Generalizable Robotic Imitation Learning

Interactive Learning and Inductive Bias

Pérez-Dattari, Rodrigo

DOI
10.4233/uuid:809581c0-5481-4586-948a-d2e739902985

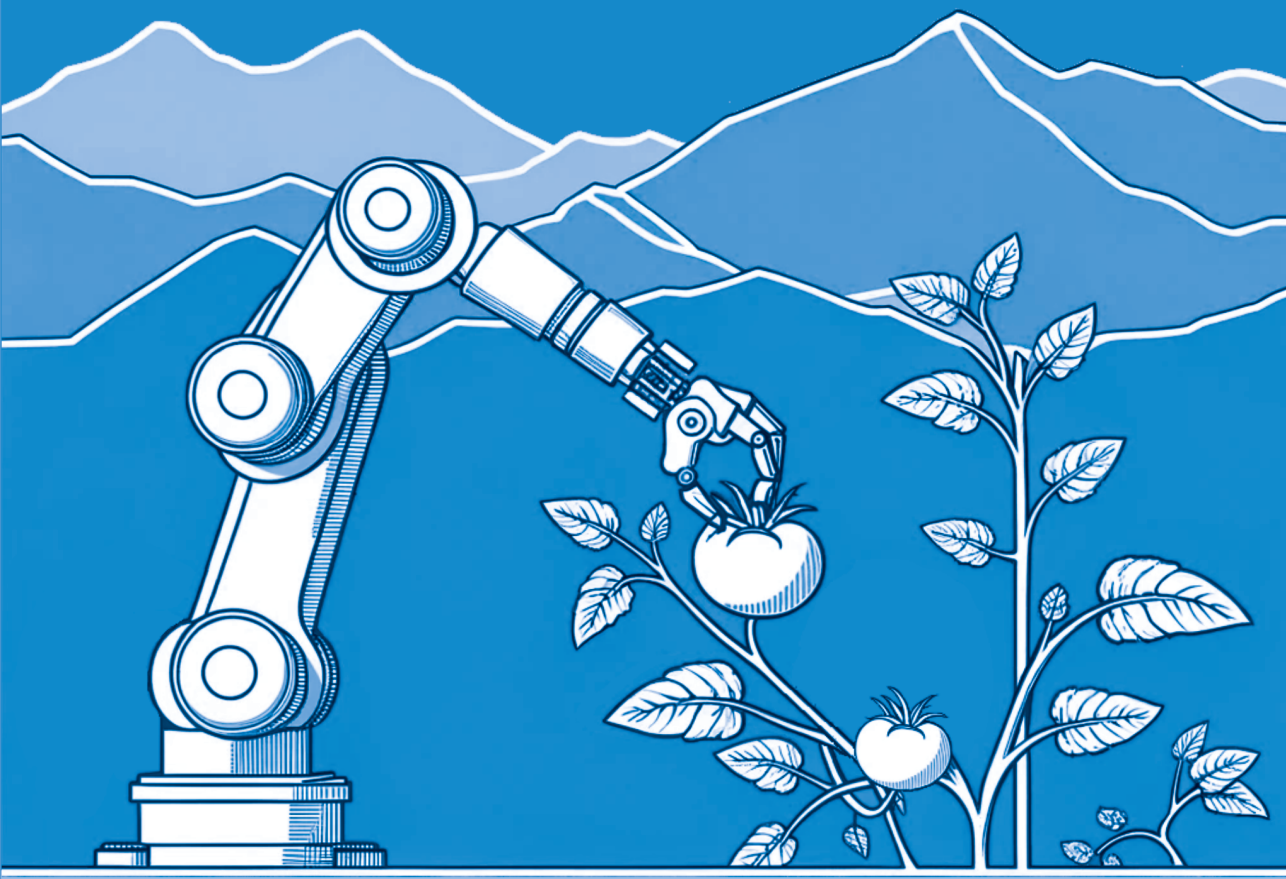
Publication date
2024

Document Version
Final published version

Citation (APA)
Pérez-Dattari, R. (2024). *Generalizable Robotic Imitation Learning: Interactive Learning and Inductive Bias*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:809581c0-5481-4586-948a-d2e739902985

Generalizable Robotic Imitation Learning

Interactive Learning and Inductive Bias



Rodrigo Javier Pérez Dattari

Generalizable Robotic Imitation Learning

Interactive Learning and Inductive Bias

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on
Wednesday 18, September 2024 at 10:00 o'clock

by

Rodrigo Javier PÉREZ DATTARI

Master of Engineering in Electrical Engineering, University of Chile, Chile
born in Santiago, Chile

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	Chairperson
Dr.-Ing. J. Kober,	Delft University of Technology, *promotor*
Prof. dr. R. Babuška,	Delft University of Technology, *promotor*
Independent members:	
Prof. dr. G. Chalvatzaki,	Technical University of Darmstadt
Dr. S.H. Mohades Kasaei,	University of Groningen
Dr. G.W. Kootstra,	Wageningen University & Research
Prof. dr.ir. M. Wisse,	Delft University of Technology
Prof. dr.ir. J. Hellendoorn,	Delft University of Technology, *reserve member*

Keywords: Imitation Learning, Human-in-the-loop, Neural Networks, State Representation Learning, Data-driven control, Dynamical Systems, Movement Primitives, Inductive Bias.

Printed by: Gildeprint

Cover by: Rodrigo Javier Pérez Dattari using AI-generated art

ISBN 978-94-6366-908-5

An electronic copy of this dissertation is available at
https://repository.tudelft.nl/.

Simplify. To know is to simplify without losing essence.

Gabriela Mistral

Contents

Summary

Robots have the potential to assume tasks across various real-world scenarios. To achieve this, we require adaptable and reactive robots that can robustly deal with products and environments that present variability. For example, in the agro-food sector, each tomato plant inside a greenhouse is unique; hence, different robotic motions are required when interacting with different plants. Unfortunately, due to their simplicity, most robotic solutions currently employed are rigid and rely on hand-crafted rules. Such solutions perform well in controlled and repetitive environments; however, they fall short when these conditions are not met. As a consequence, a large family of problems remains unsolved.

In this context, Imitation Learning (IL) presents itself as an attractive alternative, since it introduces a framework that allows modeling and adapting complex behaviors with ease. This is achieved by introducing robots that can be quickly customized by non-expert roboticists through intuitive methods similar to the ones humans use to learn from each other, e.g., demonstrations, corrections, evaluations, etc. The simplicity of utilizing these methods reduces the obstacles to real-world adoption, rendering them a practical substitute for existing solutions. Nevertheless, despite these advantages, data-driven methods often come at the cost of not being completely reliable, especially in situations that are not well represented in the training data. Hence, due to the strict task performance requirements commonly imposed in real-world environments, such limitations must be addressed for these solutions to be adopted. Therefore, in this dissertation, we study these challenges, and address them with the aim of providing tools for developing reliable and data-efficient IL methodologies that can be employed to solve problems in challenging real-world scenarios.

Historically, IL has been addressed as an offline learning problem, i.e., behaviors are transferred to robots by means of recorded demonstrations. However, one of the main limitations of such approaches is *covariate shift*. This occurs when the training data, obtained from trajectories generated by an expert demonstrator, do not align with the trajectory distribution that the robot encounters during deployment. This mismatch arises because, with respect to the training data, robots make small decision-making errors that accumulate over time. Consequently, even if a robot initially operates within a region covered by the training data, it will gradually deviate from it. This results in encountering unseen states, and, therefore, in the robot making erroneous and potentially dangerous actions, ultimately leading to failure. In this context, Interactive Imitation Learning (IIL) has emerged as an appealing alternative to overcome this problem. In IIL, training data are generated under the robot's trajectory distribution by means of online feedback. In this way, during the learning process, a human teacher observes the robot's behavior and occasionally

corrects or evaluates it. This process inherently enforces training and testing data to belong to the same distribution, as the robot learns on the distribution of states it generates itself. Consequently, IIL presents a framework that makes IL more reliable, and, therefore, it is of main interest to this thesis. Given its importance, in Chapter 2, we formalize IIL and provide an overview of this field.

Apart from IIL addressing the covariate shift problem, since humans give feedback to robots as a function of their behavior, it is possible to use feedback signals that are provided relative to the robot's actions. This can be a powerful tool, since it allows for humans to teach robots behaviors that might not be easy or feasible to demonstrate directly. In this regard, Chapter 3 introduces a method based on relative corrective feedback. This feedback modifies the robot's behavior by indicating in which direction a performed action must be modified. For instance, if a robot is moving too slow, a human can indicate the robot to go faster, without needing to provide an exact speed value for the robot to reach. Although this idea is not new, there remain multiple challenges in learning from this type of feedback. Therefore, Chapter 3 addresses one of them: tasks involving high-dimensional observations where temporal information is required to build proper state representations. Such problems can be frequently encountered in real-world scenarios. Consequently, it is of main interest to provide solutions for them. This chapter shows that this can be addressed via state representation learning strategies, which enable the extraction of spatiotemporal features using minimal, non-expert human feedback. It also contrasts various feedback modalities, underlining the particular importance of this approach in the context of relative corrections.

Although IIL introduces powerful methods for learning reliable behaviors from human feedback, these behaviors still lack interpretability and predictability. This makes it challenging for these methods to be adopted in real-world scenarios where strong requirements must be met. In this regard, it is appealing to incorporate tools from control theory into these frameworks to provide them with guarantees and make their behavior predictable. Therefore, in the context of autonomous driving, Chapter 4 introduces an approach that merges the robustness of MPC with the adaptability and efficiency of IIL. By making part of the MPC's cost function learnable, we utilize established MPC methods for achieving standard behaviors like path tracking and obstacle avoidance, while learning other parts such as velocity references. The learned parts are computed using context information, such as the vehicle's first-person view, allowing it to incorporate complex human reasoning in the computed trajectories. Experimental results, obtained using a realistic simulator, demonstrate that the proposed method achieves real-time execution and superior performance, in terms of collisions and deadlocks, compared to approaches that rely solely on optimization or solely on data-driven methods. As a result, we demonstrate that the benefits of MPC can be effectively integrated with those of IIL, leading to a reliable and powerful framework.

Then, in Chapter 5, we continue our investigation into integrating control theory knowledge with Imitation Learning (IL). The objective is to facilitate the learning of robotic motions from human demonstrations. These motions are modeled as dynamical systems, enabling the application of theories from this field to analyze

properties such as *global asymptotic stability*. In this context, globally asymptotically stable motions are those that consistently reach the task's goal state, regardless of the robot's initial conditions. This means that even when the robot encounters states not represented by the imitation data, it can still generate motions converging towards the goal. Recognizing the importance of this feature in robotic motions, we introduce a novel Deep Neural Network (DNN) loss term that enforces it. This introduction of a specialized loss term is a key distinction of our method. In contrast, other works that learn stable motions from demonstrations often do so by constraining the structure of their learning models, for example, by requiring invertibility or positive/negative definiteness. While these constraints ensure stability, they limit the applicability of the methods to a narrower range of models and tend to excessively limit the learning flexibility of function approximators. Lastly, it is important to note that this approach is compatible with IIL; however, offline IL is employed in this chapter for practical reasons related to the focus on stability loss design. We demonstrate that the proposed method, derived using Deep Metric Learning (DML) tools, effectively learns stable and accurate motions. To this end, comparisons with other methods are conducted using datasets containing demonstrations of multiple motions. Moreover, real-world validations are performed using a robot manipulator in various scenarios. These validations involve learning dynamical systems, both first and second order, in different state spaces, such as the robot's end-effector position and its 7-dimensional joint space.

Lastly, in Chapter 6, we build upon the concepts introduced in Chapter 5, with the aim of developing a more efficient and versatile loss function. This approach tackles the same challenge as the preceding chapter: enforcing global asymptotic stability in motions modeled as dynamical systems. The newly introduced loss function also incorporates principles from DML; however, it introduces fewer structural constraints in the optimization process to achieve stability. This innovation allows the robot to converge to a broader range of stable motions, thereby exhibiting improved optimization performance, particularly in learning second-order dynamical systems. Furthermore, a key feature of this methodology is its ability to ensure stability within non-Euclidean state spaces. This aspect is crucial in robotics; for example, robot orientations are often represented in non-Euclidean spaces, such as the 3-sphere for unit quaternions. Therefore, enforcing stable motions in these spaces is a critical requirement for a variety of practical applications. Similar to Chapter 5, this approach is evaluated using datasets of demonstrated motions and is further tested in the real world with robot manipulators. The real-world experiments demonstrate that robots can effectively learn complex, stable motions in the space of their end-effector pose, thereby considering their orientation.

Samenvatting

Robots hebben het potentieel om taken over te nemen in verschillende realistische scenario's. Om dit te bereiken hebben we adaptieve en reactieve robots nodig die robuust kunnen omgaan met producten en omgevingen die variabel zijn. In de agro-voedingssector is bijvoorbeeld elke tomatenplant in een kas uniek; dus zijn verschillende robotbewegingen vereist bij interactie met verschillende planten. Helaas zijn de meeste huidige robuuste oplossingen vanwege hun eenvoud star en vertrouwen ze op handgemaakte regels. Dergelijke oplossingen presteren goed in gecontroleerde en repetitieve omgevingen; echter, ze schieten tekort wanneer niet aan deze voorwaarden wordt voldaan Als gevolg hiervan blijft een groot aantal problemen onopgelost.

In deze context presenteert Imitatie Leren (IL) zich als een aantrekkelijk alternatief, aangezien het een framework introduceert dat het modelleren en aanpassen van complex gedrag gemakkelijk maakt. Dit wordt bereikt door robots te introduceren die snel aangepast kunnen worden door gebruikers zonder robotica achtergrond middels intuïtieve methoden, vergelijkbaar met de methoden die mensen gebruiken om van elkaar te leren, zoals demonstraties, correcties, evaluaties, enzovoort. De eenvoud van het gebruik van deze methoden vergemakkelijkt de toepassingin de echte wereld, waardoor ze een praktisch alternatief worden voor bestaande oplossingen. Desalniettemin, komen datagedreven methoden ondanks hun voordelen vaak met het nadeel dat ze niet volledig betrouwbaar zijn, vooral in situaties die niet goed vertegenwoordigd zijn in de trainingsdata. Vanwege de strikte prestatievereisten die vaak worden opgelegd in real-world omgevingen, moeten deze beperkingen worden aangepakt voordat deze oplossingen kunnen worden toegepast. Daarom bestuderen we in dit proefschrift deze uitdagingen en pakken ze aan met als doel het ontwikkelen van betrouwbare en datagestuurde IL-methodologieën die ingezet kunnen worden om problemen in uitdagendescenario's in de echte wereld op te lossen.

Historisch gezien is IL behandeld als een offline leerprobleem, dat wil zeggen, gedrag wordt overgebracht op robots door middel van opgenomen demonstraties. Een van de belangrijkste beperkingen van dergelijke benaderingen is echter *covariabele verschuiving*. Dit gebeurt wanneer de trainingsdata, verkregen uit trajecten gegenereerd door een geoefende demonstrator, niet overeenkomen met de trajectverdeling van de robot tijdens gebruik. Deze mismatch ontstaat doordat robots, ten opzichte van de trainingsdata, kleine fouten maken die zich in de loop van de tijd opstapelen. Daardoor zal een robot, zelfs als deze aanvankelijk binnen een gebied werkt dat door de trainingsdata wordt gedekt, geleidelijk daarvan afwijken. Dit resulteert in het tegenkomen van ongeziene toestanden, en dus in het maken van foutieve en potentieel gevaarlijke acties door de robot, wat uiteindelijk tot falen leidt. In deze context is Interactief Imitatie Leren (IIL) naar voren gekomen als een aantrekkelijk alternatief om dit probleem te overkomen. In IIL worden trainingsdata gegenereerd onder de

trajectverdeling van de robot door middel van online feedback. Op deze manier observeert een menselijke leraar tijdens het leerproces het gedrag van de robot en corrigeert of beoordeelt dit af en toe. Dit proces dwingt inherent dat trainings- en testdata tot dezelfde verdeling behoren, aangezien de robot leert met de verdeling van toestanden die het zelf genereert. Daardoor biedt IIL een framework dat IL betrouwbaarder maakt, en is daarom van hoofdbelang voor deze scriptie. Gezien het belang ervan, formaliseren we in Hoofdstuk 2 IIL en bieden een overzicht van dit veld.

Naast het feit dat IIL het probleem van covariabele verschuiving aanpakt, omdat mensen feedback geven aan robots als een functie van hun gedrag, is het mogelijk om feedbacksignalen te gebruiken die relatief zijn ten opzichte van de acties van de robot. Dit kan een krachtig hulpmiddel zijn, omdat het mensen in staat stelt om robot gedrag te leren dat misschien niet gemakkelijk of haalbaar is om direct te demonstreren. In dit opzicht introduceert Hoofdstuk 3 een methode gebaseerd op relatieve correctieve feedback. Deze feedback wijzigt het gedrag van de robot door aan te geven in welke richting een uitgevoerde actie moet worden aangepast. Bijvoorbeeld, als een robot te langzaam beweegt, kan een mens aangeven dat de robot sneller moet gaan, zonder dat een exacte snelheidswaarde voor de robot moet worden opgegeven. Hoewel dit idee niet nieuw is, blijven er meerdere uitdagingen bestaan bij het leren van dit type feedback. Daarom richt Hoofdstuk 3 zich op een van deze uitdagingen: taken die betrekking hebben op observaties met hoge dimensies waarbij tijd-afhankelijke informatie nodig is om de toestand goed te representeren. Dergelijke problemen komen vaak voor in realistische scenario's. Het is daarom van groot belang om oplossingen voor deze problemen te bieden. Dit hoofdstuk toont aan dat dit aangepakt kan worden via strategieën voor het leren van toestandrepresentaties, die het mogelijk maken om spatiotemporele kenmerken te extraheren met minimale, feedback van mensen zonder robotica achtergrond. Het contrasteert ook verschillende feedbackmodaliteiten, waarbij het bijzondere belang van deze aanpak in de context van relatieve correcties wordt benadrukt.

Hoewel IIL krachtige methoden introduceert voor het leren van betrouwbare gedrag uit menselijke feedback, mist hetnog steeds interpreteerbaarheid en voorspelbaarheid. Dit maakt het uitdagend voor deze methoden om te worden toegepast in scenario's in de echte wereld waar aan strenge eisen moet worden voldaan. In dit opzicht is het aantrekkelijk om hulpmiddelen uit de regeltheorie te integreren in deze frameworks om ze van garanties te voorzien en hun gedrag voorspelbaar te maken. Daarom introduceert Hoofdstuk 4 in de context van autonoom rijden een aanpak die de robuustheid van MPC combineert met de aanpasbaarheid en efficiëntie van IIL. Door een deel van de kostfunctie van MPC leerbaar te maken, gebruiken we gevestigde MPC-methoden om standaardgedragingen zoals padvolging en obstakelontwijking te bereiken, terwijl we andere delen zoals snelheidsreferenties leren. De geleerde delen worden berekend met contextinformatie, zoals de first-person-view van het voertuig, waardoor het complex menselijk redeneren in de berekende trajecten kan integreren. Experimentele resultaten, verkregen met behulp van een realistische simulator, tonen aan dat de voorgestelde methode real-time uitvoering en superieure prestaties behaalt, wat betreft botsingen en patstellingen, vergele-

ken met benaderingen die uitsluitend vertrouwen op optimalisatie of uitsluitend op datagestuurde methoden. Als gevolg daarvan demonstreren we dat de voordelen van MPC effectief geïntegreerd kunnen worden met die van IIL, wat leidt tot een betrouwbaar en krachtig framework.

Vervolgens zetten we in Hoofdstuk 5 ons onderzoek voort naar het integreren van kennis van regeltheorie met Imitation Learning (IL). Het doel is om het leren van robotbewegingen uit menselijke demonstraties te vergemakkelijken. Deze bewegingen worden gemodelleerd als dynamische systemen, wat de toepassing van theorieën uit dit veld mogelijk maakt om eigenschappen zoals *globale asymptotische stabiliteit* te analyseren. In deze context zijn globaal asymptotisch stabiele bewegingen diegene die consequent de doelstaat van de taak bereiken, ongeacht de initiële condities van de robot. Dit betekent dat zelfs wanneer de robot toestanden tegenkomt die niet vertegenwoordigd zijn door de imitatiedata, het nog steeds bewegingen kan genereren die naar het doel convergeren. Gezien het belang van deze eigenschap in robotbewegingen, introduceren we een nieuwe term voor de verliesterm van Deep Neural Networks (DNN) die dit afdwingt. Deze introductie van een gespecialiseerde verliesterm is een belangrijk onderscheid van onze methode. Andere werken die stabiele bewegingen uit demonstraties leren, doen dit vaak door de structuur van hun leermodellen te beperken, bijvoorbeeld door omkeerbaarheid of positieve/negatieve definitheid te vereisen. Hoewel deze beperkingen stabiliteit garanderen, beperken ze de toepasbaarheid van de methoden tot een kleiner scala van modellen en beperken ze vaak de leervrijheid van functie-approximatoren overmatig. Tot slot is het belangrijk om op te merken dat deze aanpak compatibel is met IIL; echter, bespreken we offline IL in dit hoofdstuk om praktische redenen vanwege de focus op het ontwerp van de stabiliteitsverliesterm. We tonen aan dat de voorgestelde methode, afgeleid met behulp van Deep Metric Learning (DML) tools, effectief stabiele en nauwkeurige bewegingen leert. Hiervoor worden vergelijkingen met andere methoden uitgevoerd met datasets die demonstraties van meerdere bewegingen bevatten. Bovendien worden echte wereldvalidaties uitgevoerd met behulp van een robotmanipulator in verschillende scenario's. Deze validaties omvatten het leren van dynamische systemen, zowel eerste als tweede orde, in verschillende toestandsruimten, zoals de positie van de eind-effector van de robot en zijn 7-dimensionale gewrichtsruimte.

Tot slot bouwen we in Hoofdstuk 6 voort op de concepten die zijn geïntroduceerd in Hoofdstuk 5, met als doel het ontwikkelen van een efficiëntere en veelzijdigere verliesfunctie. Deze aanpak gaat dezelfde uitdaging aan als het vorige hoofdstuk: het afdwingen van globale asymptotische stabiliteit in bewegingen gemodelleerd als dynamische systemen. De nieuw geïntroduceerde verliesfunctie integreert ook principes van DML; echter, het introduceert minder structurele beperkingen in het optimalisatieproces om stabiliteit te bereiken. Deze innovatie stelt de robot in staat om te convergeren naar een breder scala aan stabiele bewegingen, waardoor een verbeterde optimalisatieprestatie wordt vertoond, met name bij het leren van dynamische systemen van de tweede orde. Bovendien is een belangrijk kenmerk van deze methodologie de mogelijkheid om stabiliteit binnen niet-Euclidische toestandsruimten te waarborgen. Dit aspect is cruciaal in de robotica; bijvoorbeeld, robotoriënta-

ties worden vaak gerepresenteerd in niet-Euclidische ruimten, zoals de 3-sfeer voor eenheidsquaternionen. Het afdwingen van stabiele bewegingen in deze ruimten is dan ook een kritische vereiste voor een verscheidenheid aan praktische toepassingen. Net als in Hoofdstuk 5 wordt deze aanpak geëvalueerd met datasets van gedemonstreerde bewegingen en verder getest in de echte wereld met robotmanipulatoren. De experimenten in de echte wereld demonstreren dat robots effectief complexe, stabiele bewegingen kunnen leren in de ruimte van hun eindeffector pose, waarbij rekening wordt gehouden met hun oriëntatie.

1

Introduction

1

1.1. An Emerging Approach for Reactive and Adaptable Robots

In the last few decades, significant research and development have been conducted in the field of robotics [1]. However, despite these advancements, the full potential of robotics in practical real-world applications remains largely untapped. Currently, most real-world robotic systems employ simplistic methods, such as using pre-recorded waypoints to define a robot's trajectory. Although such approaches have been successful in highly structured and predictable environments, their effectiveness quickly diminishes in less controlled settings.

The agro-food industry and household settings are examples of environments where more advanced solutions are required. In these domains, each instance of a problem presents unique characteristics, necessitating distinct solutions for each case. For example, in agricultural harvesting tasks, each plant exhibits a unique morphology, necessitating robotic solutions that can adapt their behavior to these variations. Similarly, in household environments, variations in layout and resident preferences necessitate a tailored approach for each situation. See Fig. 1.1 for examples of these settings.

Furthermore, these environments require robots to adapt in real time to unforeseen changes. An example is the need for a robot to modify its planned course of action when a person unexpectedly obstructs its path, ensuring safety. Another scenario involves morphological changes a plant experiences during harvesting, where, previously inaccessible areas are revealed and must be considered in the robot's motion.

Unfortunately, implementing more advanced solutions to address these challenges often demands significant investment and time, which can be prohibitive in many scenarios. For instance, customizing robotic systems for each household requires skilled engineers, a process that becomes intractable considering the wide variety of layouts and resident preferences. In the agro-food sector, this limitation has led to a heavy reliance on human labor. However, unappealing working conditions, such as highly repetitive tasks, are making it increasingly difficult to find skilled personnel for these jobs. As a result, there is an urgent need for methods that enable programming and adapting robots to complex scenarios easily.

In this context, Imitation Learning (IL) [2, 3] emerges as a particularly promising methodology. As a data-driven approach, IL facilitates the transfer of human expertise to robots through intuitive human-robot interactions. It enables robots to learn or refine their decision-making strategies by imitating human demonstrations and/or incorporating real-time corrections or evaluations. This approach significantly reduces the complexity of programming robotic behaviors, which might otherwise require intricate rule sets or the meticulous design of reward/cost functions. Consequently, IL dramatically simplifies the adaptation of robots to novel scenarios. Furthermore, by being exposed to multiple situations in the training data, robots can learn to react successfully to real-time variations in the environment.

Additionally, IL democratizes the process of programming robots, enabling

(a) Kitchen arrangements.

(b) Tomato plants.

Figure 1.1.: Examples of variable and changing environments.
Image credits: Aleksandr Malofeev, Edgar Castrejon, and Jason Briscoe on Unsplash.

individuals without specialized robotics knowledge to impart task-specific expertise. This aspect is particularly beneficial in settings where access to robotics experts is limited, such as small to medium-sized companies or, as mentioned above, household environments.

Nevertheless, although IL offers a promising pathway for enhancing robotic capabilities, it is not without its own set of challenges that need to be addressed. Hence, to achieve IL's full potential, it is crucial to acknowledge and tackle them.

1.2. Advancing Data-Efficiency and Reliability in Imitation Learning

One of the primary challenges in IL originates from its data-driven nature, where data is obtained from human demonstrations and/or feedback. The effectiveness of IL is significantly influenced by the quantity and quality of the training data. Consequently, the more diverse and comprehensive this data is, covering a wide range of situations, the better a robot can **generalize**, and thus perform, in real-world scenarios. This aspect is crucial in IL, highlighting that, in more challenging scenarios, a reliable performance requires extensive datasets. However,

collecting large amounts of data often becomes impractical, infeasible, or costly, presenting a critical barrier to IL's practical application. Thus, the development of **data-efficient methods that perform reliably** under varied conditions stands as a foundational challenge in IL.

This challenge is central to this dissertation, and each chapter is motivated by it. To address it, two main approaches are employed throughout this work: 1) exploring an alternative formulation to traditional IL, and 2) introducing prior knowledge into our models.

1.2.1. An Alternative Formulation to Traditional IL

In traditional IL, the learning process of the robot occurs *offline* [2, 4, 5]. This means that expert demonstrations of a task are first collected, and only after this collection is complete, are they used to learn the behavior. This approach is often referred to as *behavioral cloning* [6]. However, behavioral cloning faces a critical challenge regarding data efficiency, originating from a phenomenon known as **covariate shift** [2].

Covariate shift is generally defined as the prediction problem where the probability densities in the source (training data) and target (real-world application) domains differ [7]. It manifests when a robot, trained on a specific dataset, faces situations not represented in that training data. This discrepancy often leads to the execution of erroneous actions and subsequent failures. Such scenarios are common in robotics, leading to extensive research addressing this issue.

In the field of decision making, one significant source of covariate shift is compounding errors [2, 8]. Compounding errors represent a scenario where a robot progressively deviates from the region represented in the training data. This gradual drift can eventually lead to entirely novel situations, culminating in failure. Such errors occur because learned models inherently exhibit prediction errors. While these errors might be minimal initially, they can accumulate over time. Consider, for instance, a robot navigating on a field. With each movement, the robot makes a slight deviation from the demonstrated trajectory. As these deviations add up, the robot drifts increasingly further from the intended path, eventually entering a region poorly represented by the training data. Entering this region leads to increasingly significant errors, resulting in the robot being unable to return to the original trajectory or, worse, crashing.

The above-described scenario precisely showcases the operation of behavioral cloning. To understand this method and, hence, the source of the problem further, it is essential to introduce core concepts from the field of sequential decision making [9]. In these scenarios, an entity known as the *agent* interacts within an *environment*. At each time instance, or time step, the agent receives information about the environment's current situation through the *state*. Based on its decision-making strategy, termed the *policy*, the agent executes an *action*. This action aims to modify the environment's state to achieve a predetermined goal. Consequently, the environment transitions to a new state, and this sequence of events repeats. For example, in robotics, the environment can represent the robot

and objects it interacts with, while the agent represents the algorithm or controller responsible for deciding which actions to execute, at each time step, in the robot's motors. This problem formulation is widely used in robotics for learning behaviors, where the objective is to find a policy function that maps from environmental states to actions.

Then, in behavioral cloning, the learning process starts by obtaining demonstrations from an expert agent, such as a human skilled in the task. These demonstrations consist of state-action pairs recorded at various time steps, capturing the expert's decisions for each state encountered. Multiple strategies can be employed to collect demonstrations, such as teleoperation, motion tracking, or kinesthetic teaching[1]. Subsequently, the robot's policy is modified to closely resemble the expert's policy. More specifically, for the states visited by the expert, the policy is optimized to match the expert's actions as closely as possible. This is commonly achieved by modeling the policy as a parametrized function, such as a DNN, and adjusting its parameters to meet this objective [10–12]. However, as mentioned earlier, this approach has its limitations. Small approximation errors can lead the robot to encounter state regions never visited by the expert, and hence, where the robot's behavior was never trained, i.e., covariate shift. This occurs since many regions of the state space do not require the expert to visit them when completing a task, but it is possible to reach them if mistakes about its behavior are made.

This calls for a formulation of the IL problem that overcomes this limitation. Consequently, a family of IL methods known as *Interactive Imitation Learning* (IIL) has recently gained popularity for effectively tackling this issue [13–16]. In IIL, the expert agent, also called the *teacher*, has an active role during the learning process. As the learning agent, i.e., the robot's controller, interacts with the environment, it is continually supervised and receives feedback from the teacher. When the robot's actions are erroneous, the teacher provides feedback that guides the agent to improve its policy. Hence, given this feedback, the policy is updated. This process is then repeated until a successful policy is achieved. Notably, this ensures the robot learns directly in the distribution of states it generates itself. Consequently, during testing, the robot will likely operate within regions covered by the training data, having already received feedback in the regions its policy would naturally lead to. This makes IIL drastically reduce the training data required for learning a well-performing policy, as it does not necessitate exhaustive data collection over the entire state space.

IIL represents a powerful approach with numerous untapped potentialities. Yet, it is unlikely to be a standalone solution for overcoming all the challenges of IL. Acknowledging this, the following subsection shifts focus to an alternative approach for developing reliable and data-efficient IL methods: the incorporation of prior knowledge into our learning systems.

[1] Some of these approaches generate demonstrations that contain desired state transitions rather than motor commands. This assumes the existence of a lower-level controller capable of achieving such transitions.

1.2.2. Introducing Prior Knowledge

IIL significantly increases the likelihood of robots operating in regions well-represented in their training data. However, even under IIL, there are cases where robots are forced to act in regions where training data is lacking. This can be due to practical limitations, or the nature of some problems, which makes them extremely difficult to completely represent in the training data. Consequently, assuming that covariate shift is adequately addressed, a situation can emerge where the training and testing data belong to the same distribution, yet the training data do not entirely capture this distribution, thereby harming generalization.

For example, certain scenarios can lead to *data scarcity* [17]. In such situations, limited access to human demonstrators/teachers results in a poorly represented state distribution. This challenge is particularly evident in time-limited settings, such as in household environments. Here, an IL method that enables a resident to teach a robot a new task in just 10 minutes, rather than several hours, could significantly increase the practicality of these solutions.

Furthermore, another challenge in representing state distributions arises with *long-tailed* problems [18]. These are scenarios where, despite having substantial human data, the problem presents numerous unlikely situations. In practice, this means that some of these rare events are bound to occur. However, it is not possible to cover every case in the training data. For instance, in autonomous driving, a vehicle can encounter a vast number of rare situations, and, since safety is particularly crucial in this context, addressing this problem is of critical importance.

Consequently, IL would significantly benefit from methodologies that address this issue, and, therefore, has motivated a substantial portion of this dissertation. In this work, we tackle this problem by introducing prior knowledge into the robot's behavior. The objective is to identify and integrate aspects of the robot's behavior that remain consistent across the entirety of a task, and embed them as intrinsic elements of their policies. As a result, even when encountering unfamiliar scenarios, a robot equipped with such behaviors is more likely to execute reasonable actions. Behaviors incorporated into the robot in this manner are known as **inductive bias** [19]. For instance, decision-making features like obstacle avoidance and stability, which are commonly required in robotic tasks, are excellent candidates to be incorporated through inductive bias.

Note that models constructed with these inherent properties lead to more data-efficient frameworks. Such built-in capabilities imply that certain complex behaviors, which would otherwise require extensive human data, become default features of our model. As a result, a significant portion of training data is effectively replaced by inductive bias. Moreover, this approach enables IL methods to provide guarantees, a critical aspect for learning reliable robot behaviors.

Inductive bias, then, offers a promising path to enhance data efficiency and reliability in IL techniques. However, it also introduces unique challenges. Firstly, integrating desired behaviors into a robot's policy is not straightforward and often requires extensive research and experimentation. Secondly, once integrated, the next step is to ensure that the inductive bias does not compromise

the robot's learning capabilities. These capabilities are closely tied to the function approximator representing the robot's policy. Therefore, the more expressive this function approximator is, the more adaptable the robot becomes. However, introducing inductive bias limits the range of potential solutions available to the function approximator, thereby reducing the robot's learning capacity. The challenge then lies in applying inductive bias precisely, avoiding excessive constraints on the solution set, and thus maintaining an extensive range of viable policy options. A notable example of this challenge is the *stability versus accuracy dilemma* [20]. In this scenario, stability represents the capacity of a robot to consistently reach a desired goal state, and inductive bias is utilized to ensure this property when learning motions through IL. However, this inductive bias also has the counter-effect of constraining the family of motions the robot can accurately reproduce.

To address these challenges, we explore control theory tools to model well-understood behaviors, grounded in a robust theoretical foundation. By integrating these behaviors into the robot's policy, we provide guarantees and a clear understanding of the inductive biases applied. This approach creates a synergy between the accumulated knowledge from decades of control theory research and the adaptability and practicality inherent in machine learning. We demonstrate the effectiveness of this fusion, highlighting an approach that is not only increasingly popular, but also has the potential to significantly advance the field.

1.3. This Thesis

The preceding sections have introduced the topics that form the foundation of this thesis. Bearing these in mind, we can now present a brief overview of the upcoming chapters. This overview will highlight the main ideas within these chapters while indicating how they interconnect.

1.3.1. Chapter 2: An Overview of Interactive Imitation Learning

In Chapter 2, we delve into the theoretical foundations of Interactive Imitation Learning (IIL). This section formally introduces the diverse methodologies within IIL, exploring various approaches to framing these methods. A key focus is on how the feedback from human teachers can be modeled and interpreted differently, leading to distinct IIL techniques. Furthermore, we discuss how the characteristics of a given problem influence the suitability of different IIL techniques. This aims to clarify the differences between these methods and identify the most effective feedback modalities based on the problem's parameters.

The chapter concludes with an in-depth discussion on *on-policy* and *off-policy* learning methods. Originating from the Reinforcement Learning (RL) literature, these terms have sparked disagreement over their application in IIL. However, understanding these concepts is critical for the effective use and design of IIL techniques. Therefore, we clarify them and provide examples of IIL methods that belong to either category.

1.3.2. Chapter 3: Integrating State-Representation Learning with Human Corrective Feedback

In this chapter, we explore and expand upon a specific technique for robotic learning through human feedback, known as D-COACH [21]. In this method, the human teacher signals the direction in which a robot's actions should be modified. For instance, if a robot's motion is slower than desired, the human can provide a feedback signal that implies *increase speed*. The robot then integrates this feedback to refine its actions, leading to improved performance. Note that in this approach, no specific action is provided; the feedback is more about guidance than exact numerical values, such as *move at 20 km/h*. Therefore, this technique is particularly advantageous when humans have a clear understanding of the task's solution, but find it challenging or impractical to demonstrate it directly.

D-COACH is designed to work together with DNNs, which model the robot's policy. A key application of this technique is learning behaviors directly from the raw pixels of an image. However, this method encounters limitations in problems where temporal information is essential. Take, for example, the *swing-up pendulum* problem, where a torque-limited pendulum must reach and maintain an upright position. Here, the underactuated nature of the pendulum necessitates a strategy that includes velocity information. Nevertheless, this velocity cannot be derived from a single image of the pendulum. As a result, data from multiple images must be compiled to infer this information, a process that proves difficult when relying solely on D-COACH's feedback signals.

To address this limitation, in Chapter 3, we integrate *state representation learning* into the D-COACH framework. The goal of state representation learning is to develop compact and effective state representations by utilizing auxiliary loss functions. Hence, we propose learning these representations to encapsulate temporal information, which is then integrated into the policy learned through human feedback. Via simulations and real-world experiments, we show that this integration is an effective method for addressing D-COACH's limitation while learning well-performing policies data efficiently. This is shown in problems like the swing-up pendulum, both using simulations and a real platform, and an experiment involving a 3-DoF robotic arm operating on a conveyor belt.

1.3.3. Chapter 4: Learning to Guide Model Predictive Control from Human Feedback

Chapter 4 marks the beginning of our study into the integration of control theory within IL. The context of this work is autonomous driving in unstructured environments, where we operate under the assumption of having a predetermined set of waypoints indicating a vehicle's path. Then, the objective is for the vehicle to follow the waypoints while safely interacting with other agents (e.g., vehicles, pedestrians, and bicycles), and adhering to traffic regulations. This scenario poses an interesting challenge for IL, as obtaining safe behaviors is of vital importance. However, the complexity of the task makes it intractable to rely solely on a data-driven approach.

As a result, we propose the use of a Model Predictive Controller (MPC), where we design parts of its cost function to be learnable. This approach allows us to leverage existing MPC methodologies for generating standard behaviors, such as path tracking and collision avoidance. Simultaneously, it facilitates the modeling of more complex aspects of the cost function through higher-level policies, which are learned via IIL. The vehicle's behavior emerges from the combination of the traditional elements of the MPC's cost function and these newly learned components. Consequently, our overall policy exhibits a strong inductive bias while gaining enhanced adaptability from the learnable aspect of its behavior. This strategy effectively combines the benefits of incorporating prior knowledge into learning frameworks with the advantages provided by learning from humans interactively. Due to the complexities of real-world driving scenarios, we validated this method using the CARLA Simulator [22], an advanced 3D simulator designed for autonomous driving research.

1.3.4. Chapter 5: Stable Motion Primitives via Imitation and Contrastive Learning

In Chapter 5, our focus is on the challenge of teaching robots *motion primitives*. These are fundamental actions, such as grasping or manipulating objects, that form the building blocks for more complex sequences. By integrating these primitives, robots can handle extended tasks. This approach is motivated by the concept of creating a higher-level decision-making framework where a robot can access a library of these motions and intelligently combine them for multi-step objectives, like the preparation of a meal. Interestingly, this concept has gained popularity recently, largely due to advancements in *foundation models* [23] such as ChatGPT [24] and LLaMA [25], and the research on their planning capabilities [26].

The primary objective of our research is to integrate a crucial inductive bias, specifically *global asymptotic stability*, into the learning of motion primitives from demonstrations. This concept is vital in dynamical systems theory, ensuring that irrespective of their initial conditions, systems are assured to converge to an equilibrium. Interestingly, this idea is directly applicable to the domain of motion primitives, which are commonly expected to consistently lead the robot to a specific goal state upon completing the motion. For example, in a primitive designed for grasping an object, the intended goal is a state where the object can be secured if the gripper closes. Therefore, this becomes prior knowledge that we can embed into the robot's default behavior, i.e., regardless of the trends observed in the demonstrations, the robot should always reach its goal.

To achieve this, we model motion primitives as dynamical systems and propose a novel DNN contrastive loss function that incorporates the desired stability properties into these systems. Consequently, they encode the evolution of the demonstrated motions and have a globally asymptotically stable equilibrium at the task's goal state. Importantly, this necessitates an intermediate step for translating the system's desired evolution into robotic actions. Therefore, in such settings, a lower-level controller is required, e.g., a PD-type controller with inverse dynamics, for torque-controlled robotic manipulators.

Notably, modeling motions as dynamical systems also enables reducing covariate shift by minimizing a multi-step imitation error. This approach facilitates obtaining accurate motions when employing offline IL, which is the approach we take in this chapter. Consequently, we introduce a framework that leverages dynamical systems and their underlying theory, offering a novel approach for incorporating inductive bias and reducing covariate shift. This leads to a data-efficient and reliable learning framework for motion primitives.

Lastly, experiments in two scenarios were carried out. The first scenario consisted of using datasets containing demonstrations of a variety of primitives. Then, dynamical systems were learned using this data, and simulated by integrating them to study the behavior of the learned systems and compute statistics without involving real platforms yet. The second experimental approach consisted of a real-world validation using a 7-DoF KUKA iiwa manipulator. Here, three different tasks were learned from human demonstrations, namely, writing, cleaning a table, and hanging a hammer, showcasing the successful integration of this approach in a real-world platform.

1.3.5. Chapter 6: Deep Metric Imitation Learning for Stable Motion Primitives

Finally, in Chapter 6, we delve deeper into the ideas introduced in Chapter 5, offering a more general and better-performing approach. The previous chapter presents a novel methodology for designing auxiliary cost functions, which induce stability properties in dynamical systems modeled as DNNs. Its key concept is the discovery that introducing a specific structure in the DNN's latent space leads to global asymptotic stability. This stability can then be enforced using tools from the Deep Metric Learning (DML) literature, where several approaches exist for enforcing different types of latent structures. Building on these foundations, this chapter employs a more general DML approach to introduce stability. This approach enables a more expressive learning framework, allowing the policy to converge to a broader family of solutions. As discussed in the previous section, this broader convergence directly addresses one critical challenge associated with inductive bias.

Moreover, this approach facilitates the easy integration of stability properties into dynamical systems evolving in non-Euclidean manifolds. In particular, we conduct an in-depth study of 3-dimensional spheres. These manifolds are crucial as they represent the space used to encode unit quaternions, a common and practical way to represent robot orientations. Consequently, this method enables the learning of stable motions that, for instance, encompass the complete pose of a robotic manipulator's end effector. It allows for the simultaneous consideration of Cartesian space for position and spherical space for orientation. As a result, this chapter introduces an innovative and powerful methodology for learning motion primitives, showcasing the potential of incorporating dynamical system theory and metric learning techniques into robotic behavioral learning.

The experimental validation of this approach followed the same methodology as that described above for Chapter 5. Consequently, both datasets of motions and

real-world platforms were utilized to evaluate the method. In this instance, two robot manipulators were employed for validation. A 6-DoF IRB 1200 robot was used to learn harvesting motions, and a 6-DoF Kinova Gen2 robot was employed to learn the task of placing a hammer on a table.

2

An Overview of Interactive Imitation Learning

This chapter contains relevant excerpts from the publication:

C. Celemin*, R. Pérez-Dattari*, E. Chisari*, G. Franzese*, L. de Souza Rosa, R. Prakash, Z. Ajanović, M. Ferraz, A. Valada, and J. Kober. "Interactive Imitation Learning in Robotics: A Survey". In: *Foundations and Trends® in Robotics* 10.1–2 (2022)

* Authors contributed equally.

In this chapter, we first formalize the problem of sequential decision-making using the Markov Decision Process (MDP) framework [28], and use this description to define the Interactive Imitation Learning (IIL) problem. Subsequently, we explore the various interaction modalities that humans can use to provide feedback to robots. Furthermore, we present a literature review focusing on methods that explore feedback within the agent's state-action space, as these are most pertinent to this dissertation. Finally, we delve into an in-depth discussion of *on-policy* and *off-policy* learning methods, emphasizing their significance in the context of IIL.

2.1. Theoretical Background

Many problems like solving Rubik's cube with a robotic hand, controlling the propulsion of a rocket, swinging up a pendulum, or finding the best strategy in a chess game share the necessary idea of finding the best set of actions that would successfully accomplish the task. These problems share many properties, and therefore, they can be modeled using a common framework (i.e., MDP).

2.1.1. Decision Theory

A wide variety of problems can be formalized as a sequential decision-making process, where the decision-making authority is an *agent*, operating in a certain *environment*. At each time instance t (also known as time step), the agent receives information describing the situation of the environment with the *state* vector s_t, and executes an *action* a_t, aiming to change the environment towards a desired state according to the goal of the task. The environment transitions to a new state s_{t+1}, and provides a reward r_t, which is a signal that explains the objective of the task.

When a decision-making problem has well-defined initial and terminal conditions, it is known as a *finite horizon* problem, and the period of time between its start and end is called an *episode*. The collection of states and actions experienced by the agent throughout an episode is known as a *trajectory* $\tau = (s_0, a_0, ..., s_T, a_T)$, where T corresponds to the number of time steps visited by the agent.

Decision theory provides a formal and complete framework for decision-making by combining probability and utility theory [29].

2.1.2. Markov Decision Process

Initial foundations for MDP are set by [28] and further extended by [30]. An MDP models a stochastic, sequential decision-making process in a fully observable environment as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$ with four components:

- $\mathcal{S}$: A set of all possible *states* s.
- $\mathcal{A}$: A set of all possible *actions* a. Some problems may have a state-dependent set of actions ($\mathcal{A}(s)$).

- $\mathcal{T}(s' \mid a, s)$: A *transition model* that defines $\mathbb{P}(s_{t+1} = s' \mid s_t = s, a_t = a)$, probability of reaching state s' if action a is applied in state s ($\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$).
- $\mathcal{R}(s, a)$: A *reward function*, determining the *reward* r received for applying action a in state s ($\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$).

Decisions are modeled as state-action pairs (s, a). The next state s' is determined by a probability distribution, which is defined by the transition function $\mathcal{T}$ and it is based on the current state s and the applied action a. The Markov property defines that the next state s' is dependent only on the current state and action, while the previous states and actions do not have any influence on the current transition. A deterministic policy π is a mapping from states to actions, defining which action should be chosen in that state in $\mathcal{S}$ ($\pi : \mathcal{S} \mapsto \mathcal{A}$). The policy can also have a stochastic representation, with a distribution over state-action pairs ($\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$).

In certain problems, the agent cannot directly observe the underlying state. Instead, the state can only be inferred indirectly, using an observation model (the probability distribution of different observations given the underlying state). For such problems, it is appropriate to consider the Partially Observable Markov Decision Process (POMDP) [31], where the dynamics are still described using MDPs, and the additional observation model $O(s, o)$ specifies the probability of perceiving an *observation* o in a state s.

Sequential Decision-Making Problem

The objective of solving decision-making problems modeled with MDPs or POMDPs is to find the policy π^*, that maximizes the expected accumulated reward (also known as *utility* or *value*), i.e.,

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t) \right], \tag{2.1}$$

where $p_\pi(\tau)$ corresponds to the trajectory distribution induced by π, Π to the set of possible policies, and γ is a discount factor. Since problems, in general, can have *infinite horizon* ($T \to \infty$), this sum could diverge. Therefore, the sum can be discounted with the discount factor γ, where $0 \leq \gamma < 1$. This problem formulation assumes the reward hypothesis, which claims that *"all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward)"* [9].

There exists a plethora of methods for solving sequential decision-making problems, each with unique assumptions, strengths, and weaknesses. We make rough distinctions between three different approaches: *Control*, *Planning*, and *Learning*.

The boundaries of these approaches are not always clear as many practical solutions commonly lie in between or combine these approaches. In this work,

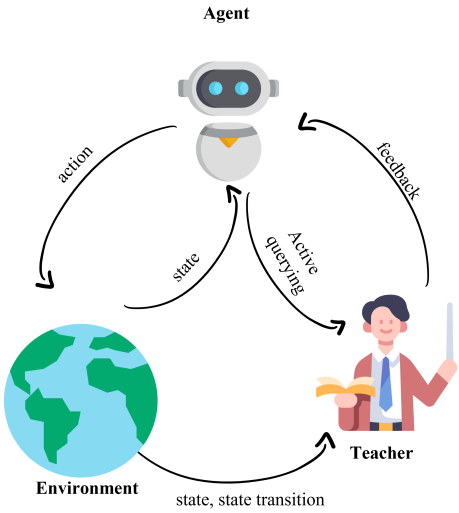


Figure 2.1.: IIL learning loop.

we focus on Interactive Robot Learning approaches, where the goal is to devise a policy by learning from interactions with humans.

2.1.3. Interactive Imitation Learning

MDPs in IIL

In IIL, a human teacher, which we will refer to as *the teacher*, aims to improve the behavior of a learning agent, which we will refer to as *the learner*, by occasionally providing feedback to it as a function of the observed behavior (see Fig. 2.1). The period of time when the teacher provides feedback to the learner is known as the *learning process*, which finishes whenever the human considers the learner's behavior appropriate or when no more improvement is observed. The human feedback can be modeled with the *feedback function* $\mathcal{H}$. Although $\mathcal{H}$ can evolve throughout a learning process (i.e., a human may modify its understanding of a task when teaching), for simplicity, the following of this chapter assumes this function does not change.

$\mathcal{H}$ is presented as a more general alternative to the reward function employed in the MDP framework. At every time step, as a consequence of the agent's behavior, $\mathcal{H}$ outputs a *feedback signal* H_t, which is defined as any type of information that can be used to improve the agent's policy (see Fig. 2.2). In IIL, feedback can be occasional; therefore, H_t consists of two values: $h_t \in \mathbb{R}^n$ and $g_t \in \{0, 1\}$. h_t provides the information employed to improve the agent's performance and g_t indicates the instances where feedback was given, i.e., h_t exists whenever $g_t = 1$. Furthermore, and differently from the reward function in Reinforcement Learning (RL), $\mathcal{H}$ may depend on previously visited

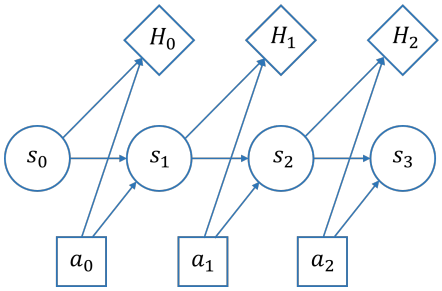


Figure 2.2.: MDP with the feedback signal H_t.

states $s_{\leq t} \equiv (s_0, s_1, ..., s_t)$ and actions $a_{\leq t} \equiv (a_0, a_1, ..., a_t)$. Hence, in the deterministic case, $\mathcal{H}(s_{\leq t}, a_{\leq t}, s'_t) : \mathcal{S}^{t+1} \times \mathcal{A}^{t+1} \times \mathcal{S} \mapsto \mathbb{R}^n \times \{0, 1\}$ (note that the domain can be a subset of $\mathcal{S}^{t+1} \times \mathcal{A}^{t+1} \times \mathcal{S}$, where X^t represents X to the power of t). Alternatively, $\mathcal{H}$ can be modeled as a probability distribution $\mathcal{H} : \mathcal{S}^{t+1} \times \mathcal{A}^{t+1} \times \mathcal{S} \times \mathbb{R}^n \times \{0, 1\} \mapsto [0, 1]$. Finally, note that this formulation can be extended to cases where the agent generates active queries, where $\mathcal{H}$ would also depend on them.

Given that h_t does not necessarily represent a reward, the problem formulation of the MDP needs to be modified accordingly. The next subsection discusses how to approach this problem.

Interactive Imitation Learning Objective

The goal of sequential decision-making problems is to find a policy π that generates trajectories $\tau \sim p_\pi(\tau)$ such that an objective function $J(\pi)$ is minimized. In RL, for instance, the objective function is defined by the policy's (negative) expected return. In IIL, however, there is not always direct access to this function, as it is commonly represented implicitly inside the teacher's mind and, therefore, it is not always possible to minimize it directly. Consequently, more generally, it is possible to formulate the problem in terms of an observable *surrogate loss* $L(\pi, \mathcal{H})$ computed as a function of the feedback function $\mathcal{H}$. We assume that the minimization of $L(\pi, \mathcal{H})$ indirectly minimizes $J(\pi)$ (or at least leads to near-optimal solutions). Note that when the true objective function of the problem is available, these two functions are the same (i.e., $L = J$). Hence, IIL aims to find a *learner's policy* π^l by solving the following optimization problem:

$$\pi^{l*} = \arg\min_{\pi \in \Pi} L(\pi, \mathcal{H}). \tag{2.2}$$

One key aspect of this equation is the approach employed to search through the space of solutions Π. In practice, when learning this sequential decision-making problem, the data used to optimize (2.2) comes from a policy that interacts with the environment, which biases the optimization problem. Hence, depending on this policy, different solutions will be obtained.

To make this idea evident, the problem can be formulated in terms of the expected immediate cost $C(s, a)$ of performing an action a for a state s [8]. Then, we can express this cost in terms of a policy π with $C_\pi(s) = \mathbb{E}_{a\sim\pi(s)}[C(s, a)]$. Consequently, the objective function becomes the accumulated expected immediate cost $J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{s\sim p_\pi^t(s)}[C_\pi(s)]$, where T corresponds to the task horizon and $p_\pi^t(s)$ is the state distribution at time step t induced by π. Once again, given that we might not have access to $C_\pi(s)$, the problem can be formulated in terms of the immediate expected surrogate loss $\ell_\pi(s) = \mathbb{E}_{a\sim\pi(s)}[\ell_\pi(s, a, \mathcal{H}(s, a))]$, yielding the following IIL optimization problem:

$$\pi^{l*} = \arg\min_{\pi\in\Pi} \sum_{t=0}^{T} \mathbb{E}_{s\sim p_\pi^t(s)}[\ell_\pi(s)]\,. \tag{2.3}$$

In practice, the expected value of the surrogate loss in (2.3) is estimated from the data collected by a policy that interacts with the environment (i.e., p_π^t is induced by this policy). For instance, Behavioral Cloning (BC) methods use the *teacher's policy* π^h to collect training data (i.e., $s \sim p_{\pi^h}^t(s)$), or, in other words, the data comes from executions of the task performed by the teacher.

It turns out that methods like BC that learn from data gathered with a policy different from the one that is later evaluated (i.e., π^l) suffer from *covariate shift*. Therefore, in IIL, the training data distribution depends on the learner's policy. In this way, these methods aim to minimize the state distribution mismatch between the data sampled at training and test time. Nevertheless, this poses a *chicken-or-the-egg* problem, since without knowing the learner's policy in advance, it is not possible to generate data from the trajectories that this policy would generate [32]. IIL methods address this by solving the problem iteratively, i.e., the learner's policy is used to collect data, improve its behavior from the data, and repeat this process N times until a well-performing policy is obtained. Hence, by noting that $\sum_{t=0}^{T} \mathbb{E}_{s\sim p_\pi^t(s)}[\ell_\pi(s)] = T\mathbb{E}_{s\sim p_\pi(s))}[\ell_\pi(s)]$, where $p_\pi(s) = \frac{1}{T}\sum_{t=0}^{T} p_\pi^t$ corresponds to the average distribution of states [8], the general IIL problem can be formulated as:

$$\textbf{IIL problem: } \pi^{l*} = \arg\min_{\pi\in\Pi} \sum_{i=1}^{N} \mathbb{E}_{s\sim p_{\pi_i^l}(s))}[\ell_\pi(s)]\,. \tag{2.4}$$

Note that in this equation there is an abuse of notation, as $p_{\pi_i^l}(s)$ represents a distribution of states that *depends* on π_i^l, but the actions taken for collecting training data do not always necessarily have to distribute exactly as π_i^l.

From (2.4) it can be observed that every IIL method has the following properties:

1. A surrogate loss ℓ_π is computed as a function of the feedback function $\mathcal{H}$.
2. The problem is formulated over state distributions that depend on the learner's policy.
3. The problem is solved iteratively by sampling, at each training iteration, from state distributions that depend on the current learner's policy.

Episodic Feedback

A family of IIL methods solves (2.2) by solving the inverse problem, i.e., $L(\pi, \mathcal{H})$ is unknown and human feedback is employed to estimate $\hat{L}(\pi, \mathcal{H})$. Then, this estimation is minimized $\hat{L}(\pi, \mathcal{H})$ with some optimization method (e.g., path planning or RL). Commonly, several trajectories are sampled from the learner's policy π^l to get Monte Carlo estimates of $L(\pi, \mathcal{H})$ and feedback is provided at the end of them, i.e, $g_t = 0$ the rest of the time. This feedback consists of an evaluation over the complete trajectory that has the form of a choice/preference [33–35], i.e., at each iteration, given the execution of two or more trajectories from the learner, the teacher provides a ranking of them. Then, the feedback is employed to gradually shift the trajectories generated by the learner in a direction where their performance will increase.

Per Step Feedback

Alternatively, many IIL methods directly solve (2.3) following approaches that were derived either from RL (value maximization) or the classical Imitation Learning (IL) (divergence minimization) literature. Therefore, in these cases, feedback is provided in a *per-step basis*, i.e., the teacher observes the behavior of the learner at each time step and provides feedback if necessary.

Value Maximization Value Maximization methods correspond to IIL approaches that employ human feedback to solve problems formulated using the RL approach (see Eq. 2.1). In other words, some part of the RL problem is modified through $\mathcal{H}$.

The most direct way of doing this is by *naively* replacing the reward function of an existing RL approach with $\mathcal{H}$ and executing the learning process as if nothing changed. However, prior research has shown that such methods may induce *positive reward cycles*, which could lead to unintended behaviors [36]. This shortcoming led to the development of approaches that built upon the RL literature but took into account this and other limitations in the method design.

Divergence Minimization The IIL methods that are derived from the literature of classical IL can be modeled as a divergence minimization problem where we assume that we have access to expert trajectories from π^h. Then, the problem is modeled as minimizing the distance between the trajectory distribution of the expert/human $p_{\pi^h}(\tau)$ and the learner $p_{\pi^h}(\tau)$. The $f - divergence$ family [37] is a class of divergences that measure distances between probability distributions. Hence, the IL problem can be seen as an *f-divergence minimization problem* [38, 39]. By denoting the f-divergence between two distributions as $D_f(\cdot, \cdot)$, IL can be formalized as:

$$\pi^{l*} = \arg\min_{\pi \in \Pi} D_f \left(p_{\pi^h}(\tau), p_\pi(\tau) \right). \tag{2.5}$$

BC methods solve (2.5) by using the *forward Kullback–Leibler divergence* (KL), which reduces the problem to the Maximum Likelihood Estimation (MLE) of the

teacher's policy from samples drawn from the trajectory distribution induced by the teacher's policy [40], i.e.,

$$\pi^{l*} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\tau \sim p_{\pi^h}(\tau)} \left[\sum_{t=0}^{T} \ln \pi(a_t|s_t) \right]. \tag{2.6}$$

Interestingly, if the *Total Variation* (TV) distance between these distributions is minimized instead, the problem reduces to the minimization of the forward KL divergence between the teacher and the learner policies from a state distribution that follows the learner's policy [39]

$$\pi^{l*} = \arg\min_{\pi \in \Pi} \mathbb{E}_{s \sim p_{\pi}(s)} \left[D_{KL} \left(\pi^h(a|s), \pi(a|s) \right) \right]. \tag{2.7}$$

Notably, if we define the surrogate loss of an IIL problem as $\ell_\pi = D_{KL}\left(\pi^h(a|s), \pi(a|s)\right)$, then we would have an IIL method that minimizes the TV divergence between the teacher's and the learner's policy. The method Data Aggregation (DAgger) [8] minimizes this objective function, which inspired a broad family of IIL methods.

In this case, the feedback function directly outputs a desired action for a given state, i.e., h_t corresponds to a sample from $\pi^h(a|s)$. The samples h_t can be employed to estimate π^h by solving the MLE problem. Alternatively, π^l can be modeled as a deterministic policy. In such cases, the samples are approximated by the minimization of a distance between π^l and h_t (e.g., Mean Squared Error (MSE) minimization). This approach can indirectly solve (2.7) if some assumptions are made; for instance, if it is assumed that π^l follows a Gaussian distribution with fixed variance, solving (2.7) is equivalent to finding a discrete policy that models the mean of this distribution through MSE minimization [2].

2.2. Modalities of Interaction

In the IIL literature, there exist various modalities of interaction that a human teacher can adopt to communicate with the learning agent. In this Chapter, we aim to provide a classification of these methods by answering the question *what kinds of feedback could a teacher use to train an agent interactively?* The feedback is the signal containing the information that human teachers explicitly communicate to the learning agent through a Human-Robot (or Human-Computer) interface. Different kinds of feedback are useful for transferring knowledge to the agent depending on factors like the task complexity, the teacher's understanding or expertise about it, the potential of the teacher to learn through the training process, or the available interface for providing feedback.

The short answer to that question provides two main categories that group the learning methods. They are based on the domain of the feedback provided by the teacher, which could be either in the *evaluative space* or in the *transition (state-action) space*. The former covers the methods in which the teacher provides a signal of assessment or evaluation about *how well* the agent performs, while the

latter category gathers the methods that require the teacher to provide feedback that let the agent learn *how to do* the task.

In both categories, there are two ways for the user to transmit the assessment or guidance to the agent. The teacher could provide feedback that is either relative or absolute. In the relative feedback case, the teacher provides a signal that contains information about the direction the agent behavior should shift to, with respect to current or other policy executions, e.g., how good a policy/transition is with respect to others, or how a transition should be modified with respect to the current one. However, since it is only a relative direction, it does not specify explicitly what the exact input-output mapping is that the model should fit to, and it might be required to gather many feedback samples to tune the final mapping, even for a specific state or state-action pair. On the other hand, the absolute feedback contains information about the current execution regarding the optimal behavior, implicitly known by the teacher. The relative feedback requires a lower cognitive load (i.e., less mental effort) for teachers because it is less informative than the absolute counterpart, which sometimes makes it less data efficient. In other words, the use of relative and absolute feedback can represent a trade-off between data efficiency and the cognitive load of the teachers during the interaction.

Table 2.1 presents the four modalities a teacher can use for interacting with a learning agent, depending on the kind of information provided and the way it is represented (absolute or relative). This dissertation focuses on methods utilizing feedback in the transition space. Consequently, the subsequent section provides a more in-depth exploration of these methods.

Table 2.1.: Modalities of interaction according to the feedback type.

	Absolute Feedback	**Relative Feedback**
Feedback in Evaluative Space	Reinforcements	Preferences
Feedback in Transition (State-Action) Space	Absolute Corrections	Relative Corrections

2.2.1. Human Feedback in Transition (State-Action) Space

Human feedback in the transition space contains information about *how to do* the task, i.e., explicit feedback that explains how a transition should be done, being it in the space of the actions, or the states. Unlike in learning from evaluative feedback, with feedback in the transition space, there is no explicit quality assessment of the policy, the feedback signals represent the teacher's insights or understanding of the task execution. This kind of feedback can be absolute, in which case the teacher is expected to demonstrate the optimal transition for the state the agent is currently visiting. Relative feedback, on the other hand, is used in cases where the teacher corrects the policy execution towards the considered right direction with respect to what the robot is executing in that time step. However, it does not assume that the correction is the optimal action, but rather

a hint in that direction. The correct action is reached after some iterations that accumulate the incremental progress of many relative corrections.

Learning from Human Absolute Corrections

In this kind of interaction, the agents are expected to receive explicit demonstrations of the task execution by the teacher, while the learning policy is controlling the agent, as shown in Fig. 2.3. Depending on the method, the teacher can provide corrective demonstrations every time step, occasionally according to the teacher's own decision, or because the learner queries them. Moreover, those demonstrations could be either only recorded, or recorded and executed. In the former case the agent executes the action from the current policy, whereas, in the latter, the agent replaces those actions with the ones demonstrated by the teacher, as in teleoperation mode. These methods are the closest to standard Learning from Demonstration (LfD) methods like Behavioral Cloning, and some of them could even be considered a generalization of BC.

Corrective Demonstrations One of the first approaches in this category is the Confidence-Based Autonomy framework presented by [13], which has two components, Confident Execution and Corrective Demonstrations. The first is a strategy that uses various thresholds to evaluate the confidence of the agent in a certain state, and in case it is too low, it stops the autonomous execution and queries the human teacher for additional demonstrations. Corrective Demonstrations are the second mechanism, which allows the teacher to provide corrections to any mistake of the agent.

The idea of Corrective Demonstrations is further investigated by [41–43], wherein they propose to leverage both prior hand-coded policies and corrective demonstrations. Instead of only obtaining directly a policy from the demonstrations, it keeps the hand-coded policy as the primary behavior, which is only replaced by the demonstrations when the robot visits states that are similar to the ones in which the corrective demonstrations were recorded. This approach was used to train the humanoid robot Aldebaran Nao to solve a complex ball dribbling task, and it shows improvement compared to a hand-coded controller.

Some of the most important methods for learning from corrective demonstrations are inspired or belong to a family of approaches based on DAgger [8], which interactively records the correct action demonstrations while a novice policy is controlling the agent. DAgger was not specifically intended for human users, the teacher could be another expert policy, like a model-based controller or a planner system. Indeed, many methods have been proposed after DAgger, since the requirement of human teacher input every time step is not the most user-friendly approach.

The idea behind DAgger is to generate roll-out trajectories with the current policy iteratively, query the expert for corrections on the visited state-action pairs, and finally add the corrected actions to the dataset used for training the policy.

As with other methods in this section, this approach enables the expert to provide corrections on the states visited by the current policy, meaning that the data

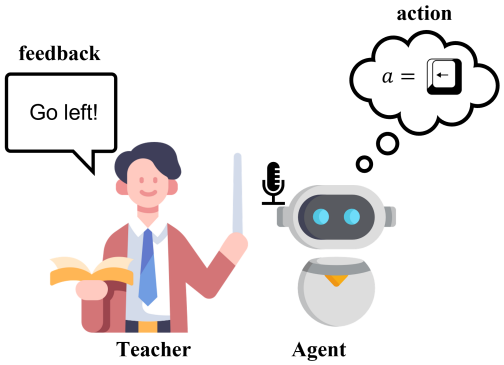


Figure 2.3.: Learning from human absolute corrections: the teacher is explicitly telling the robot to go left.

distribution is induced by the agent itself, drastically reducing compounding errors and distribution shift issues common in standard learning from demonstrations settings [8].

DAgger requires a corrective demonstration from the teacher in every state, however, it uses a gating function based on a β probability in order to control what action is actually executed, whether the action of the learner $\pi_R(s)$ or the one of the teacher $\pi_T(s)$.

At the beginning of the learning process, β is set high for the robot to execute most of the expert teacher actions because the initial robot policies could make many mistakes that lead to dangerous or irrelevant states. Through the iterations of the algorithm, β is decreased to zero in order to give full control to the learning agent. If $\beta = 1$ all the time, DAgger performs exactly as behavioral cloning because the data distribution is completely induced by the teacher.

If the expert is a human, this is often unfeasible and prone to incorrect labels for robotic tasks, which usually operate at high control frequency, generating a large number of actions for each trajectory. Most of the variants of DAgger (mentioned later) [44–48] differ from the original in i) the implementation of the gating function; ii) the way data is recorded, all aiming to improve workload, query efficiency, or safety.

The Svm-based reduction in Human InterVention (SHIV) algorithm [49] is very similar to DAgger, however, it actively requests labels in states considered risky, instead of requiring labels every time step, reducing the human burden. The risk is defined when previously unseen states are visited, or when the policy model has a high surrogate loss in the area of the visited state. The method was validated in grasping tasks, outperforming the original DAgger.

A possible alternative is to monitor the policy execution and intervene when necessary, taking over control from the agent completely. This is a more natural and intuitive approach for a human teacher compared to labeling individual

state-action pairs. This setting can be defined as learning from human intervention, and numerous studies have been presented to investigate such methods.

There exist two main types of human intervention approaches: Human-Gated and Robot-Gated [50]. Both types change the stochastic gating function based on the probability β for executing either the action of the learner or the action of the expert, with a different strategy.

Human-Gated Interventions Human-gated interventions allow the expert users to decide themselves when to intervene (control the agent). Its advantage is that safety is ensured by the expert, who is always ready to intervene in case of dangerous behavior.

Human Gated DAgger (HG-DAgger) [46] is a direct extension of the DAgger algorithm, where the human teacher is in charge of intervening when the agent drifts away from the desired behavior. Every time an intervention occurs, the expert trajectory is recorded and stored in the training data set used to optimize the policy. Additionally, HG-DAgger learns a safety threshold of a risk metric, which could be used as a policy confidence metric for different regions of the state space. The method is evaluated on both a simulated and a real-world autonomous driving task, showing better performance compared to behavior cloning and standard DAgger.

The assumption of the method is that the teacher does not intervene in the portions of the trajectory that are well executed. A different approach is used in the Intervention Weighted Regression (IWR) framework [51], where the robot's own experience is stored together with the teacher's interventions in the replay buffer. The authors show that storing such data has the advantage of reinforcing the already good behavior and improving the robustness of the policy, because more data is stored overall, and the data itself is distributed covering wider areas of the state space. Nevertheless, since the amount of intervention and non-intervention data is usually imbalanced, the authors propose a weighting parameter to prioritize the intervention samples. The method is evaluated on two challenging simulated manipulation tasks with low-dimensional observations, demonstrating better performance compared to HG-DAgger and to behavior cloning with complete demonstration.

IWR works under the assumption that the teacher is always able to correct bad behaviors, which might not be true in general, since non-expert users might be in charge of training the robot. In [52] the Corrective and Evaluative Interactive Learning (CEILing) framework combines human interventions with evaluative feedback. The use of evaluative feedback on non-corrected portions of the trajectory gives the human teacher the option to decide which part of the trajectory to use for training and which to discard. The method is shown to be able to train manipulation tasks from high-dimensional image observations directly in the real world in less than one hour of training.

Another related method is the Expert Intervention Learning (EIL) framework [53]. EIL aims to learn from the interventions as well as from the timing of the interventions since non-intervention constitutes useful information as well. They

formalize a constraint on the learner's value function, which is used to differentiate *good enough*, *bad* and *intervention* state-action pairs. The method is evaluated on a physical miniature car with a discrete action space, consisting of a library of 64 driving primitives. EIL is benchmarked against behavior Cloning and HG-DAgger, showing safe and more desirable trajectories. Another recent work in the same category is Super-Human InsErtion using Learning from Demonstration (SHIELD) [54], which focuses on the problem of industrial insertion. It extends the Deep Deterministic Policy Gradient from Demonstration (DDPGfD) [55] algorithm with a collection of different design choices, including on-policy corrections, i.e., the human can intervene to guide the agent back into the optimal region in case of deviations.

In Cycle-of-Learning [56], human-gated interventions are used for improving a policy obtained from demonstrations pre-recorded in the first stage. The experiments with a perching task using a simulated drone showed that this approach has better performance than using either only demonstrations or only human interventions.

Corrective demonstrations are not only used for learning an explicit policy, but also for learning objective functions. In Learning to Navigate from Disengagements (LaND) [57], the teacher takes over the control of autonomous navigation robots during failure situations. However, the data gathered during the interventions is not used for updating the policy, but for training a disengagement predictive model that is used as part of the cost function of the task, which is optimized during the decision-making with a model predictive control-based planner.

Robot-Gated Interventions Robot-gated interventions require the agent to estimate when an intervention is necessary, which does not require constant attention from the teacher, since the robot is the one deciding when the intervention should be performed, allowing the human to supervise multiple robots at once [48]. These methods generally require the agent to estimate a measure of performance, safety, or uncertainty about the currently observed state, which is then used to determine when to query or enable human teacher control. However, these kinds of approaches have to deal with the disengagement of the users, who do not react immediately when requested and require some time to be able to take over the system again.

One example of this approach is presented in [58], where the policy outputs a discrete vector of confidence scores for four different gripper orientations, and the one with the highest confidence is picked. An apprenticeship model is developed, which queries the teacher intervention in case of too many failures in a row or if the output confidence is lower than a certain threshold.

A variation of DAgger called Safe DAgger (SafeDAgger) [44] trains a classifier that predicts whether the learning policy deviates from the expert and, if it is the case, it switches the control to the expert in order to prevent executing unsafe actions. The authors mention that the metric used for comparing the expert and learning policy should depend on the task. Experiments with a driving simulator showed that SafeDAgger is safer and more efficient than DAgger. Ensemble Dagger

(EnsembleDAgger) [45]—a method that extends SafeDAgger—uses the deviation classifier as a discrepancy rule, along with a doubt rule that also switches control from the learning policy to the expert teacher. The doubt rule is computed based on the novelty/uncertainty of the policy, which is measured with the variance of an ensemble of neural networks. The doubt rule lets the agent prevent executing dangerous actions in unseen states, in addition to the actions of the learning policy that tend to deviate from the expert teacher.

The Lazy DAgger (LazyDAgger) [47] framework also extends SafeDAgger, in particular, it aims to reduce context switching by adding noise to the actions provided by the supervisor to improve the data distribution as well as by adopting an asymmetric switching criteria, modeled as a hysteresis function. Finally, Thrifty DAgger (ThriftyDAgger) [48] is proposed, where the switching policy is learned instead. Interventions are queried in case the encountered state is sufficiently novel or risky. Similar to EnsembleDAgger, novelty is estimated by computing the variance of each output of a set of policies, whereas the "risk" of a state is estimated by learning a Q-function to evaluate the discounted probability of success from that given state and the action proposed by the policy.

Conclusion Learning from absolute corrective demonstrations is the interactive approach most similar to standard learning from demonstration since the teacher should explicitly show what the robot has to do, i.e., she/he is required to be an expert at solving the task. However, these interactive methods have the advantage of i) reducing compound errors, because the demonstrations are given to correct the current learning policy deviations; and ii) reducing the cognitive load of the teachers since they are not required to give full demonstrations in most of the methods, but rather occasional corrections; and iii) dealing better with the mistaken demonstrations, which are not normally considered by imitation learning methods intended for non-human teachers.

Mistaken demonstrations have a direct effect that the teacher is able to predict, allowing the teachers to be aware of how to fix their mistakes. Although in most methods the mistaken feedback remains in the database used for training the policy, it is possible to compensate for them with correct labels outnumbering the mistakes, something relatively simple to do given the explicit nature of this kind of feedback (unlike with evaluative feedback).

Learning from Human Relative Corrections

Methods in this category do not require the teacher to know what the exact action or state transition should be applied by the agent in every state. However, they need to understand how a change in the action/state-transition magnitude would impact the execution of the task. In other words, the teacher should be able to roughly estimate how a transition would change if the policy is slightly modified. For instance, knowing that less power in a propeller decreases the acceleration of an aircraft or boat, or more force in the pedal brake slows down a car. With these insights, teachers could suggest how to modify a continuous policy in a more natural way (see Fig. 2.4), as it happens when a coach is instructing a student for

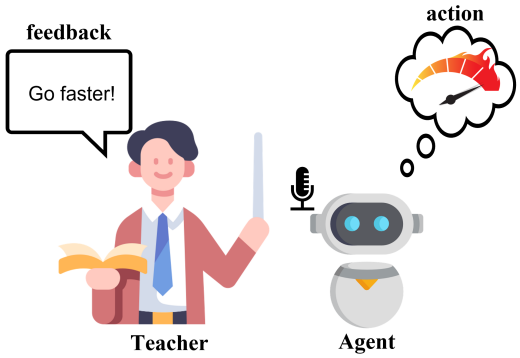


Figure 2.4.: Learning from human relative corrections: the teacher is correcting the velocity of the robot telling it that it can increase the value with respect to the current one.

learning a physical skill, e.g., in football training: *kick the ball a bit harder and more to the right side*, in a singing lesson: *slightly increase the volume of your voice in this part of the song*, during a dance move *bend the knees less*. This correction could be discrete (increase/decrease the action) as well as continuous-valued, depending on the interface used.

Advice Operators One of the first interactive methods using relative corrections is Advice-Operator Policy Improvement (A-OPI), where at each iteration, it rolls out the current policy while recording the state-action pair's trajectory. Then, in an offline phase, the teacher selects the parts of the trajectory considered to be modified, along with an associated advice operator that changes the model's action of each selected pair. Finally, there is a phase of policy re-derivation based on the updated dataset [59–61]. An advice operator can be a relative change of the current action; for instance, in a navigation task, the advice *accelerate* would change the model's current velocity request, multiplying it by 1.1. It is a relative correction because it means *increase* the current action magnitude. An advice operator can also be the demonstration of an action, being it an absolute corrective demonstration (Sec. 2.2.1). For instance, the advice *stop* changes the model's velocity request to zero. Corrective demonstrations and A-OPI were sequentially applied by [62] for improving the walking stability of a NAO humanoid robot.

The COACH framework Similarly to A-OPI, when actions are increased/decreased, the COrrective Advice Communicated by Humans (COACHc) [15] framework employs binary feedback to indicate, for a given state, the direction in which the action taken by an agent has to change, while the magnitude of the change is set as a predefined parameter in the range of the actions. A parametrized policy is directly learned in the parameter space, as in policy search

RL [63]. Differently from A-OPI, the feedback provided in COrrective Advice Communicated by Humans (COACHc) and the policy updates occur while the agent is interacting with the environment, i.e., during policy execution time, which allows the teacher to directly observe the effects of the corrections and correct again if required, speeding up the learning process. COACHc was originally formulated to model the policy as a linear combination of basis functions, which allowed to solve tasks such as teaching a NAO robot to dribble a ball [15].

The method was later extended to Deep COACH (D-COACH) [21, 64], which models the policy with Deep Neural Network (NN)s, allowing to solve tasks with high dimensional observations, like RGB images obtained from a camera such as in the problem of driving a Duckie Town car [21]. Also solving problems like balancing a real inverted pendulum swing-up, or solving a manipulation task in a conveyor belt with partial observations by incorporating memory into the NN architecture [65]. Furthermore, COACHc was combined with Policy Search RL to learn precise motor skills, solving tasks such as the ball-in-a-cup [66]. These works present experiments in which the learning agents obtained policies with higher performances with respect to the capabilities of their human teachers, who were not always able to execute the task at hand, but still managed to teach it.

COACHc is employed to learn tasks with feedback in the action space, however, corrective advice can similarly be applied to collect feedback in the state space, in tasks wherein the teacher considers that it could be more natural due to the not-so-intuitive relation or effect between the action, the current state, and the next state. With Teaching Imitative Policies in State-space (TIPS) [67], relative corrections in the state space are used for updating the policy; however, in order to find the action labels that would obtain the advised relative state correction, an additional module based on learning an inverse dynamics model is proposed. This inverse model works for translating the state space feedback into the space of the actions, such that the policy could be updated just as with COACHc. TIPS can also be considered as the interactive version of Behavioral Cloning from Observations (BCO) [68]. The method was validated with a *fishing* and a *laser drawing* task with a real KUKA LBR iiwa 7 robot, and a user study with simulated environments showed that using feedback in the state space can reduce the task load of the teachers.

Physical advice Some works that are more focused on teaching behaviors with manipulators have been proposed for letting the teachers provide kinesthetic corrections over the executed trajectories. These relative corrections are used for either updating a policy or updating the objective function that can be used in a model-based setting with a planner system.

For instance, a policy correction by the teacher on the end-effector displacement with respect to the original trajectory is detected with tactile sensors in Tactile Policy Correction (TPC) [69]. The correction could be used for policy refinement or policy reuse. In the former, the corrections are added as new data points to the training set, whereas in the latter the corrections are used to replace some already existing data points. In both cases, all the data points in the set are used

for re-deriving the policy after the execution. The approach was validated with grasping tasks using an iCub humanoid robot.

Additionally, incremental refinement of trajectories of context-dependent policies is performed with kinesthetic feedback in [70]. The corrections are not detected and computed with tactile sensors, but rather with the measured position difference between the desired trajectory and the one disturbed by the teacher. A reaching task is used in the experiments for testing the method with a BioRob arm. In [71] kinesthetic corrections are also used to reshape a movement primitive used for a feeding assistance robot application.

Relative corrections as implicit preferences The relative corrections intended to modify a manipulator trajectory are also used as implicit preference feedback, despite it being an explicit relative correction in the state space. The trajectory disturbed by the teacher is closer to what the teacher is expecting the robot to do (preferred option) than the trajectory intended by the robot. Some methods leverage this information of preference for learning a function that approximates the teacher's objective (see (2.2)), such that it could be used along with a lower-level system for computing the desired robot trajectory. Based on this concept, Trajectory Preference Perceptron (TPP) was proposed and tested in robotic tasks in a household setting and pick-and-place tasks in a grocery store checkout setting [72, 73]. Similarly, Online Learning from physical Human-Robot Interaction (HRI) was validated in household tasks with shared workspaces [74–76].

Conclusion The methods based on relative corrections allow non-expert teachers to incrementally correct the agent until the unknown desired actions are found, in a guidance setting that resembles the natural way a teacher corrects a student. Some of these methods empower the teachers, who in some cases are not able to demonstrate the task, to teach agents to perform and reach the goals successfully. Learners outperforming the teacher in IIL is similar to what we see in humans learning complex skills, e.g., when a sports coach guides the player to perform complex behaviors that they cannot do (anymore). Nevertheless, learning with this feedback modality is limited to continuous action problems.

Since this feedback is directly given in either the state or action space, methods using it are also more flexible for reverting the effect of mistaken corrections. Moreover, there are some methods that update the policies with stochastic gradient descent and do not store the feedback in a dataset, which are even less sensitive to the occasional teacher mistakes, allowing to provide a correct label that does not conflict with any previously stored wrong feedback.

2.2.2. Discussion

In this section we classified different IIL methods according to the explicit information that is given by the teacher to the learner, having two main categories: Feedback in the evaluative space, and feedback in the transition space. They are divided into subgroups of relative and absolute feedback, therefore, the

discussion sets any form of interaction within one of the four subgroups: i) Human reinforcements; ii) human preferences; iii) corrective demonstrations; iv) Relative corrections. Each of the introduced subgroups has its pros and cons which condition the situations in which they could be applied. In general, all these interaction modalities let the teachers train agents that perform better than policies obtained with standard IL, especially to reduce the problem of compound errors, since more complete data is incrementally collected with the teacher interventions during or after the policy roll-outs.

Some works have compared methods of different modalities of interaction and have found that users tend to prefer to interact with the learning agents by communicating information that explains or shows how to perform the task, rather than to provide assessments of the quality of the policy [14, 77–79]. However, this preference is not the only relevant factor that could be considered for selecting the most convenient approach to solve a specific problem.

The growing community of learning with humans in the loop research is still mostly focused on exploring new methods and evaluating their effectiveness and efficiency. Research for measuring and comparing usability will help to identify what approach or method is more convenient for each kind of problem. Usability can be assessed by analyzing how effective is a method for obtaining a successful policy, how efficient is the learning process, how pleasant the process is for the users, how sensitive it is to human mistakes, and how easy it is for the user to learn to interact with the system.

Nevertheless, there are insights that can guide the selection of the interaction modality to be used for training a policy. Depending on the used modality of interaction, the set of people who can teach a learning agent can be more or less inclusive regarding their expertise. This is correlated with the amount of information contained within the feedback signals of each modality.

With corrective demonstrations, the feedback is the informatively richest since the teacher explicitly shows what the agent should do. This means, that only users with high expertise in the task can teach the system. With the relative corrections, the set of users can be widened because not only expert demonstrators can participate, but also users are enabled to teach if they just have insights about how the transitions would change with a variation of the action. They can advise slight corrections to the agent to incrementally improve a policy. The set of possible teachers is augmented if using human reinforcements, because then, the teachers do not require to know much about what actions should be done or how transitions should be modified, as long as they can assess locally whether each part of a behavior is appropriate (assessments that implicitly happen before any intervention with the two subgroups of the transition space feedback modalities). If an action is considered wrong, the teacher does not need to know which the correct one is, he/she would just punish it for the agent to try something else until it finds the appropriate behavior. Finally, in the case of learning from preferences, the set of possible teachers is the largest one, since it includes any person who understands the objective of a task, and who can assess whether one behavior is closer to the solution than other ones, without being required to understand or

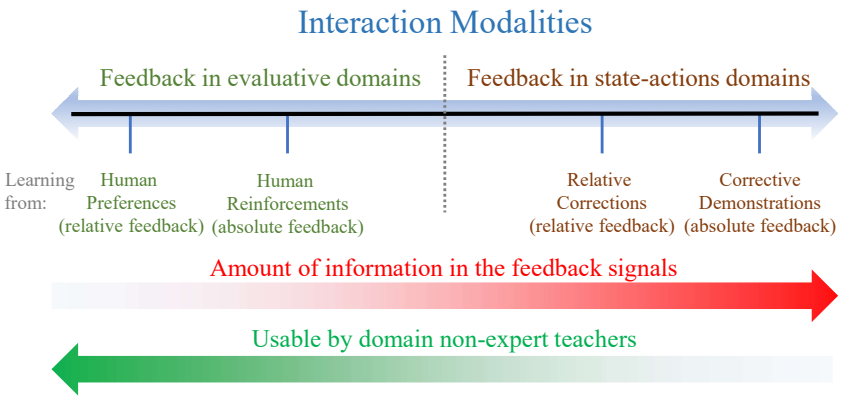


Figure 2.5.: Interaction modalities and the information contained in the feedback signals. The four modalities are organized in the plane from the right with the Corrective demonstrations modality requiring the highest expertise, to the left with the human preferences requiring the least (as shown by the green arrow). This order has a negative correlation with the amount of information shared within the feedback signals in each modality (red arrow)

specify what exactly makes the preferred behavior better.

As mentioned before, the corrective demonstrations are the most informative interactions, followed by the relative corrections that are defined in the same domain of actions or states, but they do not need to be strictly accurate since the accumulation of many corrections can gradually reach the desired behavior (Fig. 2.5). With human reinforcements, the feedback is a scalar evaluating the performance of each part of the policy execution, and it can be discrete or continuous. With human preferences, the feedback contains the least amount of information because even one discrete feedback signal (or N in the case of rankings) is used to compare full or partial trajectories, without assessing any individual decision.

Both the limitations given by human factors, or physical constraints like the ones related to learning with real physical robots that cannot be accelerated as in simulation, cannot be directly approached by adding computational power, as in the case of other Machine Learning (ML) methods. Therefore, some variables like the level of expertise of the teacher, the physical constraints given by the environment and the users, e.g., time, and the available interfaces compose the factors considered for selecting the right modality. Other variables of the interactive imitation learning problem that are discussed in the next sections consider additional nuances of the approach selection.

2.3. On/Off Policy Learning

Machine Learning methods intended to solve sequential decision-making problems (like RL or IL) feature different algorithmic properties related to what kind of data is used for learning, when and how it is generated, and how it is used for updating the policy. Depending on how these questions are approached by the method designer, learning processes could be classified as on/off-line learning and on/off-policy learning.

The chronological evolution of the focus on the way the learning data is generated for IL has been relatively opposite with respect to RL. Initially, the main idea of RL was the autonomous learning of a policy by trial and error, while the agent is interacting with the environment, i.e., collecting the data samples while testing the learning policy. However, in recent years, researchers have dedicated efforts to an additional branch for applying the MDPs properties, and RL concepts, for learning from prerecorded data without further agent-environment interaction, as is the case with offline RL [80]. On the other hand, IL was studied for many years only to find methods that could replicate behaviors contained in static datasets of expert demonstrations, and only later it has been explored the idea of incrementally collecting data from the teacher who observes the learning agent performance.

Due to the different development of these two learning paradigms, general common concepts have been independently introduced. In this section, a discussion intending to unify the definitions of these concepts given in both the RL and IL literature is presented, while trying to keep the RL definitions as the reference.

2.3.1. Online and Offline Learning

Depending on when the collection of data used for learning is carried out, the learning methods could be classified into offline or online learning. In RL, the offline learning setting is defined as the situation when *"the agent no longer has the ability to interact with the environment and collect additional transitions using the behavior policy. Instead, the learning algorithm is provided with a static dataset of transitions and must learn the best policy it can using this dataset"* [80]. In contrast, in the online learning setting, the experience the agent gathers for learning increases with new interactions with the environment, allowing it to improve the current policy.

The projection of these definitions into the world of IL matches completely with the classification of interactive and non-interactive methods. Offline learning covers the standard IL methods that sequentially record demonstrations in a static dataset, and later obtain a policy with the recorded data. Online IL methods cover the group of IIL approaches because they feature the ability to collect more data with a dynamic dataset during learning. Since in IL the data collection depends on a teacher, the continuous feedback sampling of online learning involves the teacher in the loop as it has been defined for IIL.

As mentioned in the introduction of this chapter and considering the introduced definitions, we could say that RL was initially developed for online learning, and only recently its potential for learning offline has been studied, while IL was first

formulated offline, and recently extended to the online setting.

In both RL and IL, the agent learns from the obtained feedback, provided by the environment or teacher intervention, respectively. Both learning paradigms aim at a similar objective in the offline case, since both try to obtain a policy that reproduces the behavior recorded in the data. In other words, offline RL also tries to imitate the demonstrations collected in a dataset; however, it makes use of a reward function that supports the process of defining which decisions in the demonstrated data are more relevant and which ones are less convenient for attaining the task goal.

Since, historically, offline learning has been predominately applied in IL methods, two ideas that become evident in online learning scenarios have been mostly ignored in its literature: *on-policy* and *off-policy* learning (see Fig. 2.6). These ideas have been well-defined and deeply studied by the RL community, and they play a fundamental role in the understanding and design of learning methods. In this section, we argue that the relevance that on-policy and off-policy learning has in RL also transfers to IIL. However, although some works have used these concepts in the context of IIL [81–83], they are still not clearly defined in this field. Therefore, to analyze the relevance that on/off-policy learning has in IIL, it is necessary to first clearly define it for this case.

Below, we introduce these concepts from the original definitions in the literature of RL, and thereafter they are extended to the IIL case.

On/Off-Policy Learning in Reinforcement Learning

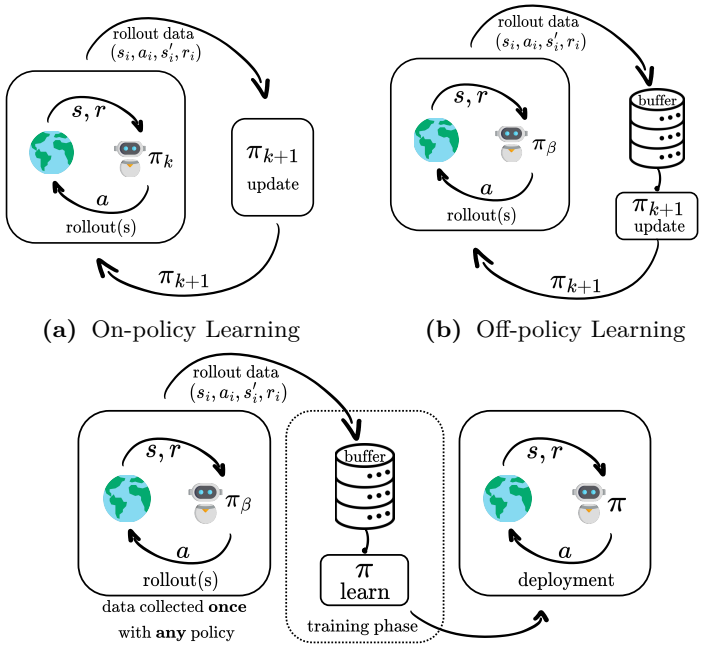
[9] define these concepts stating: *"on-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data"*. The policy that is being learned is often referred as *target policy* π^{t}, and the policy used to generate the learning data as *behavior policy* π^{b}. Then, on-policy learning occurs when the learning data comes from trajectories generated by the target policy, i.e., $\pi^{\mathrm{t}} = \pi^{\mathrm{b}}$. In contrast, if $\pi^{\mathrm{t}} \neq \pi^{\mathrm{b}}$, the learning is off-policy. Note that, consequently, offline RL requires off-policy learning.

These concepts can be formally defined from the RL objective and from how it is commonly optimized. From (2.1), we have that this objective commonly corresponds to the maximization of the discounted expected return

$$\pi^* = \arg\max_{\pi \in \Pi} \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[G(\tau) \right], \tag{2.8}$$

where $G(\tau) = \sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t, a_t)$ corresponds to the return.

In practice, since we do not have analytical access to this expectation, to find the policy π^* that maximizes the presented objective, it is necessary to empirically collect information from, ideally, every possible trajectory τ (or transition $(s_t, a_t, s_{t+1}, r_{t+1})$ given the Markov assumption) and shift the behavior of π^{t} towards the trajectory distribution that maximizes (2.8). However, in most realistic scenarios, it is not possible to sample the complete state-action space; hence, a policy is commonly chosen to sample this space as diversely and



(a) On-policy Learning **(b)** Off-policy Learning

(c) Offline Learning

Figure 2.6.: Different learning schemes used in RL that are applicable to IL. In on-policy learning, the target policy π_k is the same as the behavior policy. This policy collects the data used in the update that leads to π_{k+1} (a). In off-policy learning the behavior policy π_β is different from the target policy, allowing the use of a replay buffer. However, in practice, π_β results from combining π_k with exploration noise or input from the teacher (b). In offline learning the behavior policy π_β used for obtaining the data is completely different from the policy π obtained in the learning process, which is not considered for the data collection (c). This figure is inspired by [80], with some modifications.

exhaustively as possible while keeping the problem tractable. This policy is π^{b}. Then, at every update iteration, the data collected by π^{b} is employed to estimate the current expected return of the trajectory distribution induced by π^{t}, and π^{t} is modified such that this expectation increases. However, if data is generated by sampling trajectories induced by π^{b}, *how is the expected return computed with respect to* π^t*?* There are two options, 1) on-policy learning, i.e, directly improve π^{b} at every iteration ($\pi^{\mathrm{b}} = \pi^{\mathrm{t}}$), 2) off-policy learning, i.e., $\pi^{\mathrm{b}} \neq \pi^{\mathrm{t}}$ and employ a strategy to compute the expected return of π^{t} from trajectories collected by π^{b}. Consequently, at every learning iteration, the estimated objective of an on-policy learning method corresponds to

$$\textbf{on-policy: } \hat{\mathbb{E}}_{\tau \sim p_{\pi^{\mathrm{b}}}(\tau)} \left[G(\tau) \right], \tag{2.9}$$

where $\hat{\mathbb{E}}_{\tau \sim p_\pi(\tau)}$ corresponds to the expectation estimated by sampling data from the environment following a policy π.

In contrast, the estimated objective of off-policy methods, even though the data comes from π^{b}, corresponds to

$$\textbf{off-policy: } \hat{\mathbb{E}}_{\tau \sim p_{\pi^{\mathrm{t}}}(\tau)} \left[G(\tau) \right]. \tag{2.10}$$

Note that off-policy methods are defined as those that are able to learn off-policy, which indicates that it is also possible to learn on-policy with these methods, as they are capable of learning from data generated by any policy, which includes the target policy [9].

The RL literature provides a vast family of on-policy and off-policy learning methods, to study how the concepts of on/off-policy are applied in practice we can analyze some examples.

SARSA and Q-Learning To illustrate the difference between on-policy and off-policy methods, let us study SARSA [84] and Q-Learning [85], two seminal RL methods. SARSA is on-policy and Q-Learning is off-policy. These methods employ Temporal-Difference (TD) learning to compute estimates of the expected return and solve (2.8). TD combines ideas from Monte Carlo (MC) methods and Dynamic Programming (DP), i.e, trajectories are empirically sampled from the environment, but the final outcome is estimated based on current models of the environment (which is known as *bootstrapping*), instead of only using the sampled data. SARSA and Q-learning employ TD learning to estimate the action-value function $Q(s_t, a_t)$, which estimates the expected return of a policy given its current state and selected action and derive a policy from it. Hence, the environment can be sampled following π^{b} and bootstrapped at every time step to get the following sample/estimation of the return:

$$G(\tau)_t = \underbrace{r_{t+1}}_{\text{sample}} + \underbrace{\gamma Q(s_{t+1}, a_{t+1})}_{\text{estimation}}. \tag{2.11}$$

Then, Q can be updated by computing the error of this TD estimate with respect to the current estimation of Q for a given time step, which is known as the update rule of SARSA:

$$\textbf{SARSA: } Q^{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \underbrace{[\underbrace{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})}_{\text{TD target}} - Q(s_t, a_t)]}_{\text{TD error}}, \tag{2.12}$$

where α is the learning rate of the update. Note that the only variable that indicates that (2.12) is following π^{b} is a_{t+1}, as the other variables are a consequence of the action taken by π^{b} one time step before, which can be ignored at $t+1$ given the Markov assumption. Hence, it is possible to remove the dependence of the TD-target from π^{b} if instead of using the action a_{t+1} sampled from π^{b} in this estimate, a different one is chosen. This idea can be followed to create an off-policy variation of SARSA, known as Q-learning.

Q-learning defines its target policy as the optimal policy according to the current estimation of Q, i.e., $\pi^t(s_t) = \arg\max_a Q(s_t, a)$. Then, (2.12) can be modified by replacing the term $Q(s_{t+1}, a_{t+1})$ with the Q value of π^{t}, making the return estimation to be according to π^{t} instead of π^{b}, i.e,

Q-learning:

$$Q^{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)], \quad (2.13)$$

where the modified value with respect to (2.12) is highlighted in red. Note that in the special case where the behavior policy is greedy with respect to the current estimate of Q (e.g., when using an ϵ-greedy exploration strategy and ϵ tends to zero), the SARSA and Q-Learning update rules are equivalent [84] because the target and behavior policies are the same, i.e., Q-Learning becomes on-policy.

Importance Sampling Another well-known approach for designing off-policy learning methods is *importance sampling*. Importance sampling allows methods that in nature are on-policy, such as SARSA, to become off-policy by weighting the TD errors with the importance sampling ratio [9, 86]. The importance sampling ratio is employed to estimate the update of the Q function of the target policy from data generated by a different policy, e.g., the behavior policy. Closely related to the methods studied above, the method Temporal-Difference per Decision Importance Sampling (TD-DIS)[1] [87] can be analyzed in this case. TD-DIS method can be seen as an off-policy extension of SARSA by means of importance sampling [88, 89]. The update rule of TD-DIS is

TD-DIS:

$$Q^{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\rho_t[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)], \quad (2.14)$$

where

$$\rho_t = \frac{\pi^t(a_t|s_t)}{\pi^b(a_t|s_t)} \quad (2.15)$$

is the per-step importance sampling ratio between the target policy and the behavior policy. The similarities between SARSA and TD-DIS are evident; ρ_t is the only variable that differentiates both methods and allows the update rule of TD-DIS to be employed with data collected by the target policy. In the special case where this method is used on-policy, the behavior and target policies become the same (i.e., $\rho_t = 1$) and TD-DIS becomes equivalent to SARSA.

2.3.2. On-Policy/Off-Policy Learning in Imitation Learning

According to [2], the terms on-policy and off-policy, in the IL literature, are mentioned for the first time in [81]. The authors use the terms on-policy and

[1] The acronym TD-DIS is introduced in this work for simplicity, given that no acronym is proposed in [87].

off-policy according to which policy is used to sample data from the environment. If the current agent's policy is used to sample data, then the method is on-policy; if the teacher's policy is used to sample data, then the method is off-policy. Although this definition may seem equivalent to the one in RL, there is a difference: in RL, these definitions are about the data that is used in the evaluation or improvement of the *current* agent's policy.

This difference is important because online RL and IIL are iterative learning processes (i.e., the agent's policy is evolving over time while it interacts with the environment), which means that the distribution of the data generated by an older version of the agent's policy is not the same as the one generated by the current agent's policy. The on/off-policy definitions provided in [81] allow on-policy methods to use data generated by older versions of the agent's policy (i.e., other policies) when improving its behavior, which is not consistent with the RL definition.

As an example, DAgger [8] has commonly been defined as being on-policy and Behavioral Cloning as off-policy [2, 81, 83]. Nevertheless, in DAgger, data is constantly being aggregated in a dataset that is used to update the agent's policy iteratively. Consequently, data generated with a different policy than the target policy is used in the update rule, and, therefore, from an RL perspective, it would be an off-policy method. From this point of view, DAgger and Behavioral cloning are in the same category.

Instead, we argue that the ideas of on/off-policy learning can be transferred differently to IIL. In this section, we focus the analysis on the per-step feedback case as described in Sec. 2.1.3, as it is the case that most resembles RL.

From Reinforcement Learning to Interactive Imitation Learning

From the definition of on/off-policy learning in RL provided in Sec. 2.3.1, we can recall that off-policy learning occurs when data collected with one policy (i.e., behavior policy) is employed to update a different one (i.e., target policy). Consequently, off-policy learning allows updating a policy from data that has no dependence on it.

This same idea can be employed to study on/off-policy methods in IIL, i.e., if the data used in the update rule of the learner's policy follows a different policy, the learning method is off-policy; otherwise, it is on-policy. The only difference is that, in this case, the learner collects the feedback signal when interacting with the environment, instead of the reward signal like in RL. As mentioned in Sec. 2.1.3, the feedback signal can be understood as a generalization of the reward.

To observe this more clearly, we can analyze methods from the two paradigms that lead to IIL methods (see Sec. 2.1.3): 1) Value Maximization and 2) Divergence Minimization.

Value Maximization methods

Since these methods derive from the RL literature, they optimize the RL objective, and, therefore, the definitions provided in Sec. 2.3.1 can be directly used to define them as being on-policy or off-policy. Let us study two of these methods:

Convergent Actor-Critic by Humans (COACHe) [90] and Training an Agent Manually via Evaluative Reinforcement (TAMER) [91].

COACHe COACHe is derived employing the *policy gradient theorem* of RL [9]. This theorem allows to directly improve the parameters θ of a policy π by computing the gradient of its value function and applying stochastic gradient ascent. Consequently, COACHe applies the following update rule to its policy:

$$\theta^{\text{new}} \leftarrow \theta + \alpha \nabla_\theta \pi(s_t, a_t) \frac{h_{t+1}}{\pi(s_t, a_t)}, \tag{2.16}$$

where α is the learning rate and h_{t+1} the human feedback. Here, h_{t+1} can be interpreted as replacing the *advantage function* used in this type of policy gradient methods, which describes how much better or worse an action would perform compared to the agent's action when following the agent's policy. Consequently, to improve the agent's policy with this method, it is necessary to learn **on-policy**; otherwise, h_{t+1} would indicate the advantage of an action with respect to a policy different from the agent's policy, making its update incorrect.

TAMER TAMER can be interpreted as a method that maximizes the Q function for deriving a policy but assumes that the policy behaves *myopically* (i.e., $\gamma = 0$). Therefore, we can observe that if we assume a myopic behavior, (2.12) and (2.13) reduce to the same solution, which corresponds to the update rule employed by TAMER

$$Q^{\text{new}}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[h_{t+1} - Q(s_t, a_t)], \tag{2.17}$$

where the reward r_{t+1} is replaced by the human feedback h_{t+1}. Consequently, TAMER interprets the feedback signal as a Q-value, which does not depend on the agent's policy [92]. Moreover, it does not depend on *any* policy, but only on immediate actions. Consequently, with TAMER, it is possible to update the target policy with data collected by any policy, making it an **off-policy** learning method. Note that although it is likely that the teacher will provide feedback as a function of the learner's policy [90, 93]. For instance, TAMER is proposed considering assumptions such as *"The trainer can evaluate an action or short sequence of actions, considering the long-term effects of each"* (effects with respect to the policy) and *"a human trainer's reinforcement function, is a moving target. Intuitively, it seems likely that a human trainer will raise his or her standards as the agent's policy improves"*. However, from an algorithmic perspective, given the proposed implementation, the feedback is policy-independent.

Divergence Minimization methods

Let us recall (2.7), which summarizes IIL methods based on Divergence Minimization and rewrite it in its MLE form [39], instead of the Kullback–Leibler (KL) divergence between two policies; then, in each training iteration we solve

$$\max_{\pi \in \Pi} \mathbb{E}_{s \sim p_\pi(s), a \sim \pi^h} \left[\ln(\pi(a|s)) \right]. \tag{2.18}$$

This form of the equation is useful for analyzing the on/off-policy nature of these methods because it explicitly shows the state and action distributions. Here, we analyze two IIL methods derived from this equation: DAgger [8] and COACHc [15].

DAgger In DAgger, (2.18) is minimized by setting the feedback signal to $h_t = a_t$. Hence, h_t directly corresponds to a label of the optimal action (according to the teacher) for a given state. This method assumes that for every sampled state from which this equation is minimized, the label h_t does not depend on the current agent's policy, as it only follows π^h. Therefore, it is possible to update the agent's policy from data generated by any policy, which makes DAgger an **off-policy** learning method.

Nevertheless, there is one important remark to make. Given that the state distribution of the samples used to update (2.18) depends on the behavior policy, a different behavior policy will, inevitably, yield different solutions. However, this is also the case for RL methods, so this definition is still consistent with them.

COACHc In COACHc, the assumption is that the feedback signal corresponds to an error signal that indicates the direction in which the current agent's policy should be modified to improve its performance (feedback is only meaningful for improving the current behavior policy, and not future versions of it). Therefore, in this case, for every training iteration, an action label is generated as a function of the learner's policy with the form $a = \pi^l(s) + e \cdot h_t$, where e is a hyperparameter defined as the error magnitude. Consequently, (2.18) gets modified, since the actions do not distribute as π^h anymore, but rather as a different distribution that depends on π^l. Therefore, it is only possible to update COACHc with samples collected by the agent's policy, making this method an **on-policy** learning method.

2.3.3. Discussion

The concepts discussed in this section are as important in IIL as in RL because they are agnostic of the feedback source used for policy improvement (teacher or environment). Instead, they are related to the way the learning experience is obtained and used in the policy updates.

The replay of recorded experience and the way it is implemented is one of the main features that come into the discussion of On/Off-policy learning. But unlike RL, wherein the reward function (that could be deterministic or stochastic) is (time or policy) invariant, IIL methods could have in some cases feedback of the teacher that depends on the performance of the policy. Therefore, depending on the assumption about the teacher's feedback within a learning method, it is relevant to evaluate what kind of learning is the most convenient for the method implementation, such that it leverages that assumption.

Since in online learning the experience is incrementally collected, there are additional challenges when fitting function approximators with this kind of data. The sequential nature of these problems makes the data have spatio-temporal

correlations, therefore not following the IID assumption of most ML approaches. Additionally, when training NNs from a static dataset, the iterative process of updating the model normally reduces the error for most training data as long as there are sufficient iterations. That is because some data points require more update steps following the cost function gradient than others. However, when data samples are obtained incrementally while also learning, it is difficult to control the model to avoid either overfitting or underfitting the data. In both cases, there is the additional issue of the model being changed for other input-output mappings different from the ones used in the update, which counts as losing the already acquired knowledge, and is known as "catastrophic forgetting".

Experience replay is a technique introduced for breaking those correlations in the collected data during the policy update. It also helps to have a good balance for not overfitting to the most recent training data, while keeping the old experience in the *memory* of the model, i.e., it helps to deal with the three problems previously mentioned.

For instance, in methods wherein the teacher provides evaluative feedback at any time step, there could be two different cases:

1. When the human feedback is assumed to replace the MDP reward and used within an RL implementation, the feedback is assumed to be consistent in all the state-action space, such that the RL learning properties hold. In this case, the old feedback samples are never conflicting with the new ones, therefore, old feedback signals are always usable, and the choice of on/off-policy learning is left to the RL implementation, being both valid.

2. When it is assumed humans consider past and future in their evaluative feedback signals, and it is used as something equivalent to value function (e.g., TAMER), i.e., the feedback depends on the policy. Consequently, feedback given over state-action pairs of old policies could be contradictory with respect to the one obtained with the current policy. This assumption requires giving priority to the feedback given to the execution of the current policy, hence on-policy learning would be more appropriate.

Since the discussions of On/Off-policy learning are relatively new and not consolidated in IL, this dimension of the algorithmic features space has been neglected in some implementations of IIL methods. Some algorithms have considered a learning strategy that does not align with the assumptions of the required human feedback. It is not simple to implement methods whose algorithmic features match the feedback assumptions because the limitations created by the aforementioned problems (non-IID, over/under-fitting, catastrophic forgetting) condition the learning strategies.

The most common case of having inconsistent implementations is when the feedback is policy-dependent, but experience replay is required for stable learning, i.e., on-policy learning deals better with the assumed policy-dependent feedback, but the need for experience replay makes it necessary to learn from off-policy data. As mentioned before, importance sampling helps for decreasing the priority of data obtained with different policies [94], which is convenient for learning from

off-policy data with methods whose assumptions align with the on-policy learning conditions. The Convergent Actor-Critic by Humans (COACHe) algorithm [82] is a good example of IIL with a policy-dependent feedback assumption (naturally on-policy), which benefits of off-policy learning for stability, but using importance sampling to prioritize the data in the updates according to the distribution of the current policy.

3

Integrating State-Representation Learning with Human Corrective Feedback

This chapter is based on the publication:

R. Pérez-Dattari, C. Celemin, G. Franzese, J. Ruiz-del-Solar, and J. Kober. "Interactive learning of temporal features for control: Shaping policies and state representations from human feedback". In: *IEEE Robotics & Automation Magazine* 27.2 (2020), pp. 46–54

3.1. Introduction

The ongoing industry revolution is demanding more flexible products, including robots in household environments or medium-scale factories. Such robots should be able to adapt to new conditions and environments, and to be programmed with ease. As an example, let us suppose that there are robot manipulators working in an industrial production line that need to perform a new task. If these robots were hard coded, it could take days to adapt them to the new settings, which would stop the production of the factory. Easily programmable robots by non-expert humans would speed up this process considerably.

In this regard, we present a framework in which robots are capable to quickly learn new control policies and state representations, by using occasional corrective human feedback. To achieve this, we focus on interactively learning these policies from non-expert humans that act as teachers.

We present a Neural Network (NN) architecture, along with an Interactive Imitation Learning (IIL) method, which efficiently learns spatiotemporal features and policies from raw high dimensional observations (raw pixels from an image), for tasks in environments not fully temporally observable.

We denominate IIL as a branch of Imitation Learning (IL) where human teachers provide different kinds of feedback to the robots, like new demonstrations triggered by robot queries [13], corrections [46], preferences [95], reinforcements [93], etc. Most IL methods work under the assumption of learning from perfect demonstrations; therefore, they fail when teachers only have partial insights in the task execution. Non-expert teachers could be considered all the users who are neither Machine Learning (ML)/control experts, nor skilled to fully show the desired behavior of the policy.

Interactive approaches like COACH [96], and some Interactive Reinforcement Learning (IRL) approaches [90, 93], are intended for non-expert teachers, but are not completely deployable for end-users. Sequential decision-making learning methods (IL, IIL, IRL, etc.) rely on good state representations, which make the shaping of the policy landscape simple, and provide good generalization properties. However, this requirement brings the need of experts on feature engineering to pre-process the states properly, before running the learning algorithms.

The inclusion of Deep Learning (DL) in IL (given its popularity gained in the field of Reinforcement Learning (RL) [97]), allows to skip pre-processing modules for the input of the policies, since some architectures of NNs endow the agents with intrinsic feature extraction capabilities. This has been exhaustively tested in end-to-end settings [97]. In this regard, DL allows non-expert humans to shape policies based only on their feedback.

Nevertheless, in real-world problems, we commonly face tasks wherein the observations do not explain the complete state of the agent due to the lack of temporal information (e.g., rates of change), or because the agent-environment interaction is non-Markovian (e.g., dealing with occlusions). For these cases, it is necessary to provide memory to the learning policy. Recurrent Neural Networks (RNNs) can learn to model dependencies on the past, and map them to the current outputs. This recurrency has been used in RL and IL mostly using Long

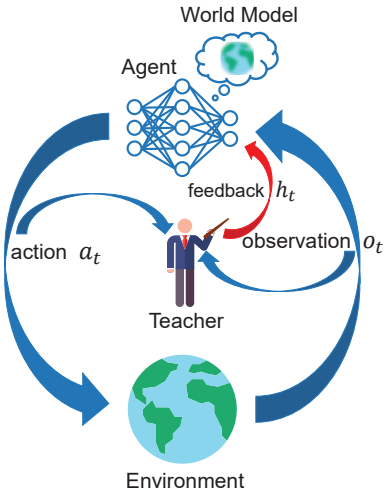


Figure 3.1.: Interactively shaping policies with agents that model the world.

Short-Term Memory (LSTM) networks [98].

Therefore, LSTMs are included in our NN architecture to learn temporal features, which contain relevant information from the past. However, DL algorithms require large amounts of data, so as a way to tackle this shortcoming, State Representation Learning (SRL) has been used to learn features more efficiently [99, 100]. Considering that real robots and human users have time limitations, as an SRL strategy, a model of the world is learned to obtain state representations that make the policy convergence possible within feasible training time intervals (see Fig. 3.1).

The combination of SRL and the teacher's feedback is a powerful strategy to efficiently learn temporal features from raw observations in non-Markovian environments.

The experiments presented in this chapter show the impact of the proposed architecture in terms of data efficiency and policy final performance within the Deep COACH (D-COACH) IIL framework [21]. Additionally, the experimental procedure shows that the proposed architecture could be even used with other IL methods, such as Data Aggregation (DAgger) [8]. The code used in this chapter can be found at: `https://github.com/rperezdattari/Interactive-Learning-of-Temporal-Features-for-Control`.

The chapter is organized as follows: background on approaches used within our proposed method, and the related work are presented in Sec. 3.2. Sec. 3.3 describes the proposed NN architecture along with the learning method. Experiments and results are given in Sec. 3.4, and finally the conclusions are drawn in Sec. 3.5.

3.2. Background and Related Work

Our method combines elements from SRL, IL and memory in NN models to build a framework that enables non-expert teachers to interactively shape policies in tasks

with non-Markovian environments. These elements are introduced hereunder.

3.2.1. Dealing with non-Markovian Environments

There are different reasons why a process could be partially observable. One of them is when the state describes time-dependent phenomena, but the observation only contains partial information of it. For instance, velocities cannot be estimated from camera images unless observations from different time steps are combined. Other examples of time-dependent phenomena are temporary occlusions or corrupted communication systems between the sensors and the agent.

For these environments, the temporal information needs to be implicitly obtained within the policy model. There are two well-known approaches for adding memory to agents in sequential decision-making problems when using NNs as function approximators:

1. **Observation stacking policies [101]**: stacking the last N observations $(o_t, o_{t-1}, ...o_N)$, and using this stack as the input of the network.

2. **Recurrent policies [102]:** including RNN layers in the policy architecture.

One of the main issues of observation stacking is that the memory of these models is determined by the number of stacked observations. The overhead increase rapidly for larger sequences in high-dimensional observation problems.

In contrast, RNNs have the ability to model information for an arbitrarily long amount of time [103]. Also, they do not add input-related overheads, because when these models are evaluated, they only use the last observation. Therefore, RNNs have lower computational cost than observation stacking. Given the more practical usage of recurrent models and their capability of representing arbitrarily long sequences, in this work we use RNN-based policies (with LSTM layers) in the proposed NN architecture.

Nevertheless, the use of LSTMs has a critical disadvantage, since its training is more complex and requires more data, something very problematic when considering human teachers and real systems. We will now introduce SRL, which helps to accelerate the LSTM converge.

3.2.2. State Representation Learning

In most of the problems faced in robotics, the state s_t, which fully describes the situation of the environment at time step t, is not fully accessible from the robot's observation o_t. As mentioned before, in several problems the observations lack temporal information required in the state description. Evenmore, these observations tend to be raw sensor measurements that can be high-dimensional, highly redundant and ambiguous. A portion of this data may even be irrelevant.

As a consequence, to successfully solve these problems a policy needs to 1) find temporal correlations between several consecutive observations, and 2) extract relevant features from observations that are hard to interpret. However, finding relations between these large data structures with the underlying phenomena

of the environment, while learning controllers, can be extremely inefficient. Therefore, efficiently building controllers on top of raw observations requires to learn informative low-dimensional state representations [104]. The objective of SRL is to obtain an observer capable of generating such representations.

A compact representation of a state is considered to be suitable for control if the resulting state representation:

- is Markovian,
- has good generalization to unseen states, and
- is defined in low dimensional space (considerably lower than the actual observation dimensionality) [99].

Along with the control objective function (e.g., reward function, imitation cost function), other objective functions can be used for SRL [100], namely:

- observation reconstruction,
- forward model or next observation prediction,
- inverse model,
- reward function, or
- value function.

3.2.3. Interactive Learning methods

This subsection introduces briefly two approaches for interactively learning from human teachers while agents are executing the task.

Data Aggregation: (HG-)DAgger

DAgger [8] is an IIL algorithm that aims to collect data with online sampling. To achieve this, trajectories are generated by combining the agent's policy π_θ and the expert's policy. The observations o_t and the demonstrator's corresponding actions a_t^* are paired and added to a database $\mathcal{D}$, which is used for training the policy's parameters θ iteratively in a supervised learning manner, in order to asymptotically approach the expert's policy. At the beginning of the learning process, the demonstrator has all the influence over the trajectory made by the agent; then, the probability of following the demonstrator's actions decays exponentially.

For working in real-world systems, with humans as demonstrators, a variation of DAgger, Human-Gated DAgger (HG-DAgger) [46], was introduced. In this approach, the demonstrator is not expected to give labels over every action of the agent, but only in places where s/he considers that the agent's policy needs improvement. Only these labels are aggregated to the database and used for updating the policy. Additionally, every time feedback is given by the human, the policy will follow the provided action. As a safety measure, in HG-DAgger the

Algorithm 1 (HG-)DAgger

Require: demonstrations database $\mathcal{D}$ with initial demonstrations, policy update frequency b
for t = 1,2,... **do**
  if mod(t, b) is 0 **then**
    update π_θ from $\mathcal{D}$
  observe state o_t
  select action from agent or expert
  execute action
  feedback provide label a_t^* for o_t, if necessary
  aggregate (o_t, a_t^*) to $\mathcal{D}$

uncertainty of the policy over the observation space is estimated; this element is omitted in this work. Algorithm 1 shows the general structure of DAgger and HG-DAgger.

Deep COACH

In this framework [21], humans shape policies giving occasional corrective feedback over the actions executed by the agents. If an agent takes an action that the human considers to be erroneous, then s/he would indicate with a binary signal h_t, the direction in which the action should be modified.

This feedback is used to generate an error signal for updating the policy parameters θ. It is done in a supervised learning manner with the cost function J using the mean squared error and stochastic gradient descent. Hence, the update rule is:

$$\theta \leftarrow \theta - \alpha \cdot \nabla_\theta J(\theta). \tag{3.1}$$

The feedback given by the human only indicates the sign of the policy error. Its magnitude is supposed to be unknown, since the algorithm works under the assumption that the user is non-expert; therefore, s/he does not know the magnitude of the proper action. Instead, the error magnitude is defined as the hyperparameter e, that must be defined before starting the learning process. Thus, the policy $error_t$ is defined by $h_t \cdot e$.

To compute a gradient in the parameter space of the policy, the error needs to be a function of θ. This is achieved by observing that:

$$error_t(\theta) = a_t^{\text{target}} - \pi_\theta(o_t) \tag{3.2}$$

where a_t^{target} is the incremental objective generated by the feedback of the human $a_t^{\text{target}} = a_t + error_t$ and a_t is the current output of the policy π_θ. From (3.1) and (3.2), and the derivative of the mean squared error, we can get the COACH update step:

$$\theta \leftarrow \theta + \alpha \cdot error_t \cdot \nabla_\theta \pi_\theta. \tag{3.3}$$

Algorithm 2 Deep COACH

Require: error magnitude e, buffer update interval b
Init: $\mathcal{B} = []$ # *initialize memory buffer*
for t = 1,2,... **do**
  observe state o_t
  execute action $a_t = \pi_\theta(o_t)$
  feedback human corrective advice h_t
  if h_t is not **0 then**
    $error_t = h_t \cdot e$
    $a_{target(t)} = a_t + error_t$
    update π using SGD with pair $(o_t, a_t^{\text{target}})$
    update π using SGD with a mini-batch sampled from $\mathcal{B}$
    append $(o_t, a_t^{\text{target}})$ to $\mathcal{B}$
  if mod(t, b) is 0 **then**
    update π_θ using SGD with a mini-batch sampled from $\mathcal{B}$

To be more data efficient and to avoid locally over-fitting to the most recent corrections, Deep COACH has a memory buffer that stores the tuple $(o_t, a_t^{\text{target}})$ and replays this information during learning. Additionally, when working in problems with high-dimensional observations, an autoencoding cost is incorporated in Deep COACH as an observation reconstruction SRL strategy. In the Deep COACH pseudo-code (Algorithm 2) this SRL step is omitted. Deep COACH learns everything from scratch in only one interactive phase, unlike other deep interactive RL approaches [90, 93], which split the learning process into two sequential learning phases. First, recording samples of the environment for training a dimensionality reduction model (e.g., an autoencoder); secondly, using that model for the input of the policy network during the actual interactive learning process.

3.3. Learning Temporal Features Based on Interactive Teaching and World Modelling

In this section, the SRL NN architecture is described along with the interactive algorithm for policy shaping.

3.3.1. Network Architecture for Extracting Temporal Features

For approaching problems that lack temporal information in the observations, the most common solution is to model the policy with RNNs as discussed in Sec. 3.2.1; therefore, we propose to shape policies that are built on top of RNNs, with occasional human feedback. In this work, we are using the terms world model and transition model interchangeably.

IIL methods can take advantage of SRL for training with other objective functions by 1) making use of all the experience collected in every time step, and

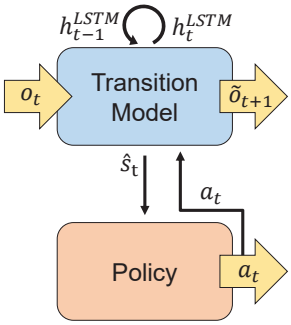


Figure 3.2.: Transition model and policy general structure.

2) boosting the process of finding compact Markovian embeddings. We propose to have a neural architecture separated into two parts: 1) transition model, and 2) policy. The transition model is in charge of learning the dynamics of the environment in a supervised manner using samples collected by the agent. The policy part is shaped only using corrective feedback. Fig. 3.2 shows a diagram of this architecture.

Learning to predict the next observation o_{t+1} forces a Markovian state representation. This has been successfully applied in RL [105]. RNNs can encode information from past observations in their hidden state h_t^{LSTM}. Thus, the objective of the first part of the neural network is to learn $\mathcal{M}(o_t, a_t, h_{t-1}^{\text{LSTM}}) = \widetilde{o}_{t+1}$, which, as a consequence, learns to embed past observations in h_t^{LSTM}. Additionally, when the observations are high-dimensional (raw images), the agents also need to learn to compress spatial information. To achieve this, a common approach is to compress this information in the latent space of an autoencoder.

For the first part of the architecture, we propose to use the combination of an autoencoder with an LSTM to compute the transition function model, i.e., predicting the next high-dimensional observation. A detailed diagram of this architecture can be seen in Fig. 3.3.

In the second part of the architecture, the policy takes as input, a representation of the state $\hat{s}_t$, that is generated inside the transition model network. This representation is obtained at the output of a fully-connected layer (FC3), that combines the information of h_{t-1}^{LSTM} with the encoder compression of the current observation $e(o_t)$. This is achieved by adding a skipping connection between the output of the encoder and the output of the LSTM.

3.3.2. Interactive Algorithm for Policy and World-Model Learning

In Algorithm 3, the pseudo-code of the state representation learning strategy is presented. The hidden state of the LSTM is denoted as h^{LSTM}, and the human corrective feedback as h. In every time step, a buffer $\mathcal{D}$ stores the samples of the transitions with sequences of length τ (line 5). The agent executes an action

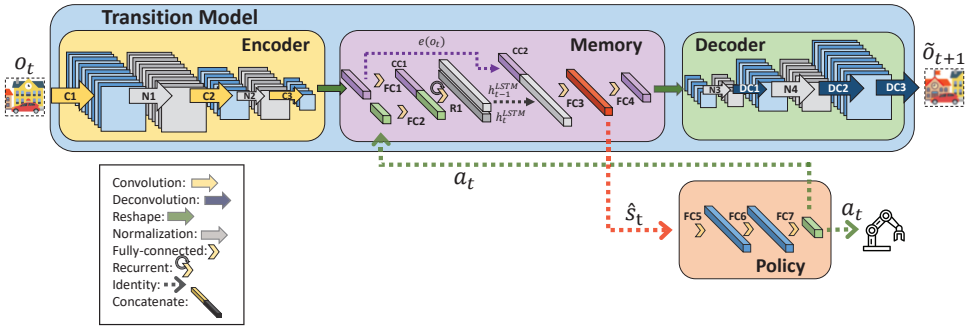


Figure 3.3.: Proposed neural network architecture. Convolutional and recurrent (LSTM) layers are included in the transition model in order to learn spatiotemporal state representations. The estimated state $\hat{s}_t$ is used as input to the policy, which is a fully-connected NN.

Algorithm 3 Online Temporal Feature Learning

1: **Require:** Policy update algorithm π^{update}, training sequence length τ, model update rate d
2: **Init:** $\mathcal{D} = []$
3: **for** t = 1,2,... **do**
4: **observe** observation o_t
5: **append** $(o_{t-1}, ..., o_{t-\tau}, a_{t-1}, ..., a_{t-\tau}, o_t)$ to $\mathcal{D}$
6: **execute** action $a_t = \pi_\theta(o_t, h_{t-1}^{\text{LSTM}})$
7: **compute** h_t^{LSTM} from $\mathcal{M}(o_t, a_t, h_{t-1}^{\text{LSTM}})$
8: **feedback** human feedback h_t
9: **call** $\pi^{\text{update}}(o_t, a_t, h_t)$
10: **if** mod(t, d) is 0 **then**
11: **update** $\mathcal{M}$ using SGD with mini-batches of sequences sampled from $\mathcal{D}$

based on its last observation and the current hidden state of the LSTM (line 6). This hidden state is updated using its previous value and the most recent observation and action (line 7). Line 8 captures the occasional feedback of the teacher, which could be a relative correction when using Deep COACH, or the corrective demonstration when using HG-DAgger. Also, depending on the learning algorithm, the policy is updated in different ways (line 9) .

$\mathcal{D}$ replays past transitions of the environment in order to update the transition function model (line 11). This is done following the *bootstrapped random updates* [102] strategy. This model is updated every d time steps.

3.4. EXPERIMENTS

In this section, experiments for validating the proposed neural network architecture and the interactive training algorithm are presented. In order to obtain a thorough

evaluation, different experiments are carried out to compare and measure the performance (return i.e., sum of rewards) of the proposed components. Initially, the network architecture based on SRL is evaluated in an ablation study, aiming to quantify the data efficiency improvement added by its different components. Then, using the proposed architecture, D-COACH is compared with (HG-)DAgger using simulated tasks and simulated teachers (oracles). The third set of experiments is carried out with human teachers in simulated environments, again comparing different learning methods. Finally, a fourth set of validation experiments is carried out in real systems with human teachers. Most of the results are presented in this chapter; however, some of them are in the supplementary material, along with more detailed information on the experiments

Two real and three simulated environments with different complexity levels were used, all of them using raw images as observations. The simulated environments are Mountain-Car, Swing-Up Pendulum, and Car Racing, whose implementations are taken from OpenAI Gym [106]. These simulations provide rendered image frames as observations of the environment. These frames visually describe the position of the system but not its velocity, which is necessary to control the system. The experiments on the real physical systems consist of a Swing-Up Pendulum and a setup for picking oranges on a conveyor belt with a 3 degrees of freedom (DoF) robot arm.

The metrics used for the comparisons are the achieved final policy performance, and the speed of convergence, which is very relevant when dealing with both, real systems and human teachers. A video showing most of these experiments can be found at: https://youtu.be/4kWGfNdm21A?si=ir_1Rw4G4E-SrmBi.

3.4.1. Ablation study

In this ablation study, the architecture of the network is the independent variable of the study. Three independent comparisons were carried out using DAgger, HG-DAgger and D-COACH. The training sessions were run using a simulated teacher to avoid any influence of human factors.

Three different architectures were tested for learning the policy from an oracle. The structure of the networks is introduced below:

1. **Full network:** Proposed architecture.
2. **Memoryless state representation learning (M-less SRL):** Similar to the full network, but without using recurrence between the encoder and decoder. The autoencoder is trained using the reconstruction error of the observation.
3. **Direct policy learning (DPL):** Same architecture as in the full network, but without using SRL, i.e., not training the transition model. The encoding, recurrent layers and policy are trained only using the cost of the policy.

The ablation study is done on a modified version of the Car Racing environment. Normally, this environment provides an upper view of a car in a racing track. In

this case, we occluded the bottom half of this observation, such that the agent is not able to precisely know its position in the track. This position can be estimated if past observations are taken into account. As a consequence, this is an appropriate setting for making a comparison of different neural network architectures. Table 3.1 shows the different performances obtained by the learning algorithms when modifying the structure of the network. These results show a normalized averaged return over 10 repetitions for each experiment, in which 5 evaluations were carried for each one of these repetitions.

Table 3.1.: Performance (return) comparison of different learning methods in the Car Racing problem. Returns were normalized with respect to the best performance (DAgger Full).

	FULL	**M-less SRL**	**DPL**
D-COACH	0.97	0.76	0.68
DAgger	1.00	0.87	0.96
HG-DAgger	0.89	0.69	0.90

As expected, DAgger with the Full architecture obtained the best performance, and given that it receives new samples every time step, it was robust against the changes in the architecture, even when it did not have memory. On the other hand, D-COACH was very sensitive to the changes in the architecture, especially with the DPL architecture. This shows how the Full model is able to enhance the performance of the agents in problems where temporal information is required. It even makes Deep COACH perform almost as well as DAgger, despite that the former does not require constant and perfect teacher feedback. Finally, HG-DAgger was more robust than D-COACH in the DPL case, but its performance with the full model was not as good.

3.4.2. Simulated tasks with simulated teachers

In the second set of experiments, a comparison between the algorithms DAgger, HG-DAgger, and Deep COACH was carried out using the proposed Full network architecture. To keep the experiments free from human factor effects, the teaching process was, once again, performed with simulated teachers. The methods were tested in the simulated problems Mountain Car (in the supplementary material), and Swing-Up Pendulum. A mean of the return obtained over 20 repetitions is presented for these experiments, along with the maximum and minimum values of these distributions.

Swing-Up Pendulum

In the case of the Swing-Up Pendulum, the results are very different for both DAgger agents (see Fig. 3.4). Both have a higher rate of improvement than Deep COACH during the first minutes, when the policy is learning the swinging behavior. Since the swinging part requires large actions, the improvement with

Deep COACH is slower. However, once the policy is able to swing the pendulum up, the second part of the task is to keep the balance in the upright position, which requires fine actions. It is at this point when learning becomes easier for the Deep COACH agent, which obtains a constant and faster improvement than the HG-DAgger agent, even reaching a higher performance. In Fig. 3.4, the expected performance upper bound is showed with a black dashed line, which is the return obtained by the simulated teacher. The purple dashed line shows the performance of a random policy, which is the expected lower bound.

3.4.3. Simulated tasks with human teachers

The previous experiments give insights into how the policy architectures and/or the learning methods perform when imitating an oracle. Most IL methods are intended for learning from any source of expert demonstrations. It does not have to be a human necessarily; it can be any type of agent. However, the scope of this work is on learning from non-expert human teachers, who are complex to model and simulate. Therefore, conclusions have to be based on results that also include validation with real users.

Experiments with the Mountain-Car (in the supplementary material), and the Swing-Up Pendulum were run with 8 human teachers. In this case, the classical DAgger approach is not used, since, as discussed in Sec. 3.2.3, it is not specifically designed for human users. Instead, HG-DAgger is validated.

Swing-Up Pendulum

This task is relatively simple from a control theory point of view. Nevertheless, it is quite challenging for humans to tele-operate the pendulum, due to its fast dynamics. Indeed, the participants were not able to successfully tele-operate the agent; therefore, unlike the Mountain-Car task, we could consider the participants as non-experts on the task.

Fig. 3.5 shows the results of this experiment, which are similar to the ones presented in Fig. 3.4. At the beginning, Deep COACH has a slower improvement when learning to swing up; however, it learns faster than HG-DAgger when the policy needs to learn the accurate task of balancing the pendulum. For the users, it is more intuitive and easier to improve the balancing with the relative corrections of Deep COACH than with the perfect corrective demonstrations of HG-DAgger, as they do not need to know the right action, rather just the direction of the correction. Unlike the performance of the simulated teacher depicted in Fig. 3.4, this plot shows the performance of the best human teacher tele-operating the pendulum with the same interface used for the teaching process. It can be seen that using both agents allowed to obtain policies that outperform the non-expert human teachers.

All the policies trained with Deep COACH were able to balance the pendulum, whereas with HG-DAgger the success rate was the half. Additionally, after the experiment, the participants were queried about what learning strategy they preferred. Seven out of eight expressed preference for Deep COACH.

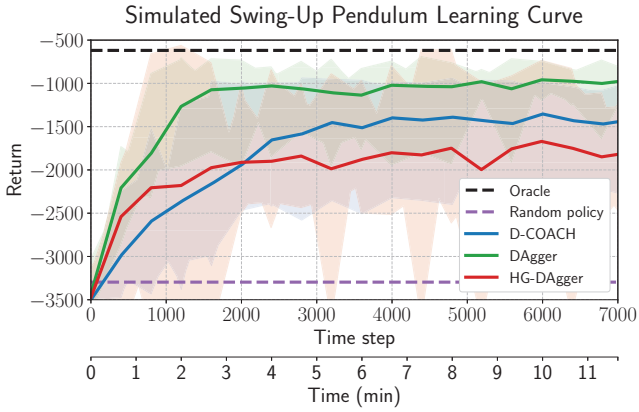


Figure 3.4.: D-COACH and (HG-)DAgger comparison in the Swing-Up Pendulum problem using a simulated teacher.

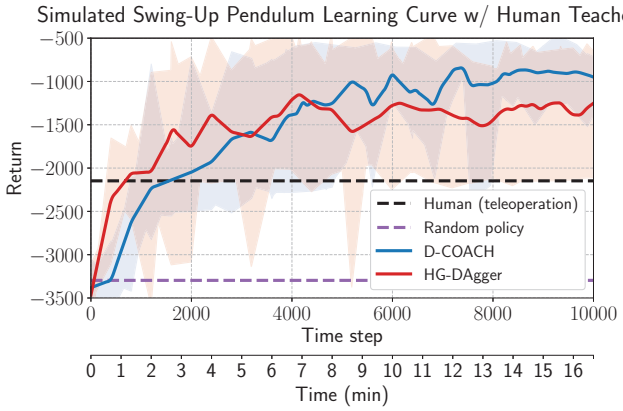


Figure 3.5.: Simulated Swing-Up Pendulum learning curve with human teachers

3.4.4. VALIDATION ON PHYSICAL SYSTEMS WITH HUMAN TEACHERS

The previous experiments performed comparison studies of the NN architectures and the learning methods under controlled conditions in simulated environments. In this section, Deep COACH is validated with human teachers and real systems in two different tasks: 1) a real Swing-Up Pendulum, and 2) a fruits classifier robot arm.

The real Swing-Up pendulum is a very complex system for a human to tele-operate. Its dynamics are faster than the simulated one of OpenAI Gym used in the previous experiments. The supplementary material provides more details

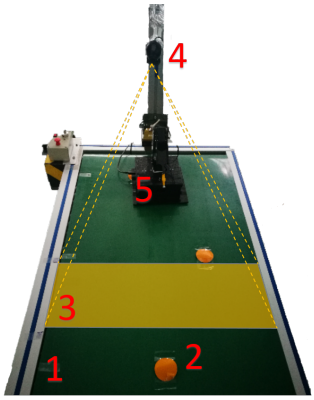


Figure 3.6.: Orange selector experimental set-up. 1) conveyor belt, 2) "orange" samples, 3) frame observed by the camera, 4) RGB camera, and 5) 3 DoF robot arm.

of this environment along with the learning curve of the agents trained by the participants of this validation experiment. Those results, along with the video, show that non-expert teachers can manage to teach good policies.

Orange selector with a robot arm

This set-up consists of a conveyor belt transporting "pears" and "oranges", a 3 DoF robot arm located over the belt, and an RGB camera with a top view of the belt (see Fig. 3.6). The image of the camera does not capture the robot arm. The robot has to select oranges with the end effector, but avoid pears. The robot does not have any tool like a gripper or vacuum gripper to pick up the oranges. Therefore, in this context, we consider a successful selection of an orange when the end effector intersects the object. The performance of the learning policy is measured using two indices: 1) rate of oranges successfully selected, and 2) rate of pears successfully rejected.

The observations obtained by the camera are from a different region of the conveyor belt than where the robot is acting. Therefore observations cannot be used for compensating the robot position in the current time step, rather they are meaningful for future decisions. In other words, the current action must be based on past observations. Indeed, the delay between the observations and its influence on the actions is around 1.5 seconds. This delay is given by the difference between the time when the object gets out from the camera range and the time it reaches the robot's operating range. This is why this task requires to learn temporal features for the policy.

The problem is solved by splitting it into two sub-tasks which are trained separately:

1. **Orange selection:** The robot must intercept the orange coordinate with the

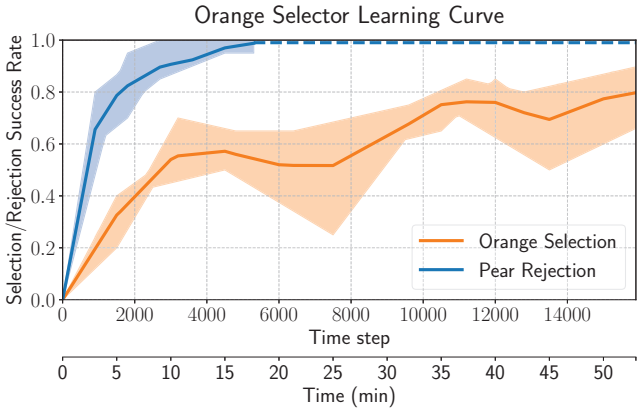


Figure 3.7.: Orange selection/pear rejection learning curve.

end effector, right when it passes below the robot.

2. **Pear rejection:** The robot must classify between oranges and pears, so when a pear is approaching under the robot, the end effector should be lifted far from the belt plane, otherwise it should get close.

These two sub-tasks can be trained sequentially. The orange selection is trained initially, with a procedure in which there are some oranges being transported by the belt with fixed positions, while some others are placed randomly. This in order to avoid over-fitting of the policy to specific sequences.

When the robot is able to track the oranges in its reach, the pear rejection learning starts. For that, pears are placed randomly throughout the sequences of oranges, and the human teacher advises corrections on the robot movement in order to make the end effector move away from the pears when they are in the operation region of the robot.

Fig. 3.7 depicts the average learning curves for this task after 5 runs of the teaching process. It is possible to see that the pear rejection sub-task is learned within 20 minutes with 100% success, while the orange selection is a harder sub-task that only reaches around 80% success after 50 minutes. Effectively, combining the two sub-tasks, the performance of the learned policies is given only by the success of the orange selection, since the pear rejection was perfectly attained in all the runs executed for this experiment.

3.5. Conclusions

This chapter has introduced and validated a SRL strategy for learning policies interactively from human teachers in environments not fully temporally observable. Results show that when meaningful spatiotemporal features are extracted, it is

possible to teach complex end-to-end policies to agents using just occasional, relative, and binary corrective signals. Even more, these policies can be learned from teachers who are not skilled to execute the task.

The evaluations with the Data Aggregation approaches and Deep COACH depict the potential of this kind of architecture to work on different IIL methods. Especially in methods based on occasional feedback, which are intended to reduce the human workload.

The comparative results between HG-DAgger and Deep COACH with non-expert teachers showed that with the former, the policy will remain biased with mistaken samples even if the teacher makes sure of not providing more wrong corrections (given that it works with the assumption of expert demonstrations); hence, it makes harder to refine the policy. On the other hand, Deep COACH proved to be more robust to mistaken corrections given by humans, since all the non-expert users were able to teach tasks that they were not able to demonstrate.

The previous mentioned shortcoming of DAgger algorithms open possibilities for future works, which are intended to study how to deal with databases with mistaken examples. Another field of study is the one of data-efficient movement generation in animation [107], which combined with our method, would make it possible to learn (non-)periodic movements using spatiotemporal features and IIL. Challenges such as the generation of smooth, precise, and stylistic movements (i.e., dealing with high-frequency details [108]) could be also addressed.

4

Learning to Guide Model Predictive Control from Human Feedback

This chapter is based on the publication:

R. Pérez-Dattari*, B. Brito*, O. de Groot, J. Kober, and J. Alonso-Mora. "Visually-guided motion planning for autonomous driving from interactive demonstrations". In: *Engineering Applications of Artificial Intelligence* 116 (2022)

* Authors contributed equally.

4.1. Introduction

Autonomous navigation in unstructured human environments (e.g., indoor and urban) poses a combination of problems, such as continuously changing conditions (e.g., sunny and cloudy), interaction/coordination with other agents (e.g., pedestrians, bicycles, human drivers and/or other automated vehicles), and ensuring the safety of people inside the vehicle and/or other agents in the environment. Consequently, building robust robotic solutions in such environments remains challenging.

The safety of Autonomous Vehicles (AV) has been a main topic of interest in the research community. Optimization-based techniques for local trajectory planning, such as Model Predictive Control (MPC), have gained popularity, since they can provide safety guarantees through the enforcement of constraints, e.g., for collision avoidance. Nevertheless, the performance of optimization-based methods is limited in complex environments, since they typically rely on geometric information and hard-coded rules to control high-level variables (e.g., switching between behaviors, controlling velocity references, etc.), which are either costly or lead to suboptimal solutions. Hence, the interaction and coordination of AVs with other agents, in unstructured environments, remains challenging.

To address this limitation, recently, there has been a growing interest in approaches that combine the strengths of optimization-based methods with the ones of learning-based methods [110, 111]. Learning techniques have shown to be a powerful tool for finding complex solutions directly from environment data, without requiring models.

In this work, we propose to learn human-like driving behaviors and encode them in a Model Predictive Contouring Control (MPCC) planner [112]. Human-like driving behaviors are desired in AVs as they produce trust in other human drivers and facilitate coordination/interaction with other agents by acting predictably [113]. However, human data can be expensive to obtain, and modeling complex environments with changing conditions may require large amounts of data. Consequently, we propose to follow an Interactive Imitation Learning (IIL) approach [13, 14, 46], which—in contrast to non-interactive Imitation Learning approaches (e.g., Behavioral Cloning)— is data efficient. IIL employs online human feedback to transfer implicit knowledge from humans to robots.

To induce the learned behavior in the solutions of the MPCC, we propose to *learn to control high-level variables used in its objective function* such that the resulting optimization process yields solutions corresponding to the desired behavior. As a first step, we focus on controlling the forward velocity reference of the MPCC. This reference has a large impact on the vehicle's behavior and it is challenging to design otherwise, given that it depends on many variables [114]. To closely match human behavior, we propose to learn to control this reference from (approximately) the same visual input that humans use, the first-person front-view of the vehicle (see Fig. 4.1).

The main *contributions* of this work are:

- Combining the state-of-the-art from control and machine learning in a unified framework and problem formulation for motion planning.

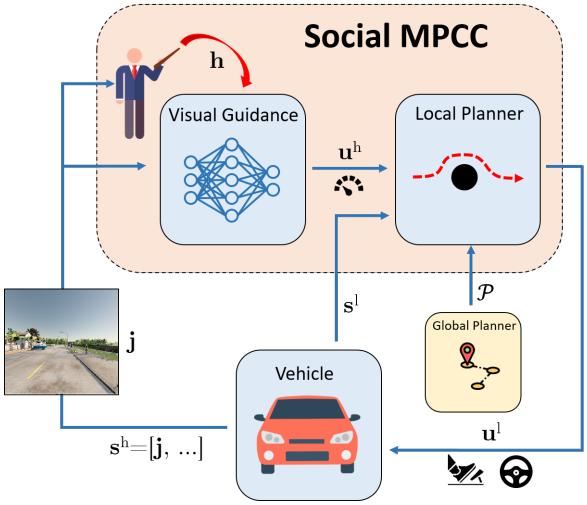


Figure 4.1.: Proposed framework, Social MPCC. The Visual Guidance observes the environment from state s^{h} and suggests the next velocity reference u^{h} to the Local Planner. Then, the Local Planner—as a function of the state s^{l}, the velocity reference u^{h} and the global path $\mathcal{P}$—computes a local trajectory and sends a control command u^{l} to the vehicle. Depending on the resulting vehicle behavior, the teacher may correct the Visual Guidance through the signal h to improve its behavior.

- A framework to generate safe and socially-compliant trajectories in unstructured urban scenarios by learning human-like driving behavior efficiently.

We present simulation results[1] in realistic driving scenarios using the CARLA simulator [115]. The presented results show that our approach can data-efficiently learn velocity references from human feedback using images as input, enhancing the performance of local trajectory planners and generating safe and socially compliant behaviors. Furthermore, we compare our approach with optimization-based-only and learning-based-only baselines, demonstrating the strength of combining both methods. Finally, qualitative results show the ability of the method to learn human-like driving behaviors.

The remaining of this chapter is organized as follows: Sec. 4.2 presents works related to decision-making algorithms for motion planning and IIL methods, Sec. 4.3 the problem formulation, Sec. 4.4 the proposed method and Sec. 4.5 the experimental results.

[1]Code available at: https://github.com/rperezdattari/Social-MPCC

4.2. Related Work

In this section, we review work from the two fields that are brought together in this chapter: 1) Motion Planning and 2) Interactive Imitation Learning.

4.2.1. Motion Planning

Classical autonomous navigation systems frequently employ a hierarchic planning architecture decomposing the navigation pipeline into a sequence of blocks performing different sub-tasks such as perception, high-level decision making and motion planning [116]. These works can be divided into three main categories: rule-based, optimization-based and learning-based.

Rule-based methods aim to translate human-driving rules and behaviors into handcrafted functions. These methods have demonstrated good performance in some real structured scenarios such as precedence at an intersection [117]. Nevertheless, these methods are scenario-specific and are prone to fail if the environment structure changes.

Optimization-based approaches typically model the decision-making problem as a Partially Observable Markov Decision Process (POMDP) as the other agents' intentions are not directly observable [118]. To model interaction, [119] proposed a joint approach for behavior prediction and planning, combining online POMDP solvers [120] for behavior prediction and nonlinear receding horizon control for trajectory planning [121]. Nevertheless, these approaches have scalability issues and assume structured navigation scenarios.

Learning-based methods can scale to cluttered and unstructured environments [122] allowing to incorporate high-dimensional data (e.g., RGB-D images, LiDAR point-clouds, etc.) into the decision-making policy [123]. For instance, Reinforcement Learning (RL) methods have been used to learn end-to-end control policies for autonomous racing [124] and indoor navigation [125] by learning a policy optimizing for long-term rewards. To generate socially compliant behaviors, [126] proposed to introduce social rules into the learning framework by designing a reward function penalizing the agent when not respecting human navigation norms. Yet, these methods do not provide any robustness or safety guarantees [127].

Recently, works combining learning-based approaches for decision-making and optimization-based methods for motion planning have demonstrated to achieve superior performance by providing guidance on high-level decision variables needed to solve the optimization [128, 129]. Closely related to our work, [130] learned a subgoal policy from visual information using Model Predictive Control (MPC) as supervisor. In contrast, we propose to learn a visual decision-making policy from human feedback. Similarly, [131] used adversarial learning to train an end-to-end decision-making module from human demonstrations. Nevertheless, it assumes a high-definition map to be available and considers a discrete set of decisions limiting the applicability of this approach only to well-constrained driving scenarios.

4.2.2. Interactive Imitation Learning

Interactive Imitation Learning (IIL) is a branch of Imitation Learning (IL) whose objective is to develop algorithms that transfer a policy from a teacher to a learning agent (learner) through interactions between the teacher and the learner [3, 46, 65]. Some examples of feedback are demonstrations [8, 13], relative corrections [15, 65], preferences [35], and evaluations [93]. In autonomous driving, it is common to find methods that work with humans as teachers, and demonstrations as feedback [46, 132–135], given that 1) it is easy to find humans that know how to drive a vehicle, and 2) human-like driving is a desired feature in autonomous vehicles [114, 133]. Therefore, building on top of this evidence, the IIL part of our work follows this same strategy.

Although, in the context of IIL, demonstrations are interpreted as feedback, they can be applied in non-interactive IL algorithms as well, i.e., Behavioral Cloning (BC) and Inverse Reinforcement Learning (IRL) [2]. However, compared to non-interactive methods, IIL poses an ideal setting to learn from humans, as it reduces human effort by being data efficient. This is achieved by providing feedback online over trajectories induced by the learner's policy, which improves its behavior only in the relevant regions of the state space (i.e., the ones that are likely to be visited) [8, 16] Furthermore, IRL, not only suffers from inefficiency in terms of amount of demonstrations, but also suffers from inefficiency in terms of interactions with the environment [2, 136], which can be a limitation when a realistic simulator of the environment is not available.

The IIL method employed in this chapter can be interpreted as a practical variation of DAgger [8], since DAgger is not designed to work interactively with humans. DAgger expects the teacher to provide demonstrations at every state visited by the learner, and the trajectories generated by the learner are a result of a mixed control setting that combines actions from the learner and from the teacher. However, humans are sensitive to timing and latency; therefore, providing good demonstrations over an agent that is partially controlled is counterintuitive and cognitively demanding [50]. Alternatively, the teacher can observe the learner's behavior and intervene whenever this behavior is not appropriate, taking control over the learner and using these actions as demonstrations, as proposed by [16, 46, 137]. The method used in this work belongs to this group of approaches. Note that this group can be extended [51] and combined with other types of feedback [52] and/or active learning [138].

4.3. Preliminaries

Throughout this chapter we use the term *ego-agent* to refer to the agent controlled by our method (e.g., autonomous vehicle or mobile robot) and *other agents* to refer to the non-controllable agents (e.g., human-driven vehicles, pedestrians, or robots) in the surrounding of the ego-agent. Moreover, the Euclidean norm of x is denoted by $\|x\|$ and $\|x\|_Q = x^T Q x$ denotes the weighted squared norm.

4.3.1. Problem Formulation

Consider the navigation scenario where an ego-agent must navigate from an initial position p_0 to a goal position g. At the beginning of an episode, the ego-agent receives a global reference path $\mathcal{P}$ to follow from a path planner consisting of a sequence of M reference way points $p_m^{\text{ref}} = [x_m^{\text{ref}}, y_m^{\text{ref}}] \in \mathbb{R}^2$ with $m \in \mathcal{M} := \{1, \dots, M\}$. Then, consider a hierarchical control structure with a high-level control policy π_θ^{h}, defined as a parametrized function with parameters θ, and a predefined optimization-based low-level controller π^{l} that follows $\mathcal{P}$. The superscripts h and l are used to denote the variables related to the high-level and low-level controllers, respectively. For each time step k, the high-level policy receives the state s_k^{h} and takes an action $u_k^{\text{h}} = \pi_\theta^{\text{h}}(s_k^{\text{h}})$. Subsequently, u_k^{h} is provided, along with the state s_k^{l} and the global reference path, to the low-level controller, which takes an action $u_k^{\text{l}} = \pi^{\text{l}}(s_k^{\text{l}}, u_k^{\text{h}}; \mathcal{P})$. This action leads to the next state $s_{k+1}^{\text{l}} = f(s_k^{\text{l}}, u_k^{\text{l}})$, under the dynamic model $f(s_k^{\text{l}}, u_k^{\text{l}})$[2].

The policy that encompasses the combination of π_θ^{h} and π^{l} is denoted as $\pi_\theta(s_k; \mathcal{P})$, where $s_k = [s_k^{\text{h}}, s_k^{\text{l}}]$. Note that the control output $u_k = \pi_\theta(s_k; \mathcal{P})$ is the same as u_k^{l}, since u_k^{l} is the output applied to the vehicle, while u_k^{h} acts on the parameters of π^{l}.

Simultaneously, we consider that for each time step, the ego-agent receives the feedback signal h_k, which provides information about a desired, expert behavior, π^{exp}. The goal is to employ h_k to find the parameters θ such that π_θ converges to π^{exp}. By doing so, π_θ^{h} learns to guide π^{l} such that a desired behavior is achieved when following $\mathcal{P}$.

Let $p_{\pi_\theta}(\tau)$ be a distribution over trajectories τ induced by π_θ, and $p_{\pi^{\text{exp}}}(\tau)$ a trajectory distribution induced by π^{exp}, then, the problem can be formulated as the minimization of the (forward) Kullback–Leibler divergence between the trajectories induced by π_θ and π^{exp} [40]:

$$
\begin{aligned}
\theta^* = \underset{\theta}{\operatorname{argmin}} \quad & D_{\text{KL}}\left(p_{\pi_\theta}(\tau) \| p_{\pi^{\text{exp}}}(\tau)\right) && \text{(4.1a)}\\
\text{s.t.} \quad & s_{k+1}^{\text{l}} = f\left(s_k^{\text{l}}, u_k^{\text{l}}\right), && \text{(4.1b)}\\
& u_k^{\text{l}} = \pi^{\text{l}}(s_k^{\text{l}}, u_k^{\text{h}}; \mathcal{P}), && \text{(4.1c)}\\
& u_k^{\text{l}} \in \mathcal{U}^{\text{l}},\ u_k^{\text{h}} \in \mathcal{U}^{\text{h}},\ s_k^{\text{l}} \in \mathcal{S}^{\text{l}},\ s_k^{\text{h}} \in \mathcal{S}^{\text{h}}, && \text{(4.1d)}\\
& \forall k \in \mathbb{R}^+.
\end{aligned}
$$

Here, (4.1b) are the kino-dynamic constraints and (4.1d) represents the state and control constraints where $\mathcal{S}^i$ and $\mathcal{U}^i$, $i \in \{\text{l}, \text{h}\}$, are the set of admissible states and control inputs (e.g., maximum agents' speed), respectively.

Note that, in this work, h_k is provided by a human; hence, θ^* will depend on the human's judgment about the task.

[2] This is identical to the Vehicle Model used in the simulation defined in Sec. 4.4.3

4.4. Method

In this section, we introduce the proposed socially-aware Model Predictive Contouring Control (Social-MPCC) framework.

4.4.1. Overview

The proposed driving system can be divided into two parts: **Visual Guidance** and **Local Motion Planner**. The Local Motion Planner π^{l} follows a given set of way points and ensures to avoid obstacles. Simultaneously, the Visual Guidance system π_θ^{h} uses images captured by the front camera view of the vehicle to command a desired forward velocity reference v^{ref} to the Local Motion Planner such that human-like driving behavior is generated. Given that the vehicle's steering commands are defined by the local planner only, it is not possible to exactly match a reference human-like behavior by means of controlling v^{ref} alone. Nevertheless, arguably, v_k^{ref} is expressive enough to accurately resemble human-like behavior in most of the situations; for instance, in the case of a vehicle in a city, the vehicle should reduce its velocity in the crossroads, stop at red lights, accelerate when overtaking other cars, etc.

4.4.2. Visual Guidance

For each time step k, the Visual Guidance system (VG), represented as the parametrized policy π_θ^{h}, translates human driving behavior and scene context into a forward velocity reference v_k^{ref}, which corresponds the high-level control output $u_k^{\mathrm{h}} \coloneqq v_k^{\mathrm{ref}}$. The state of this function s_k^{h} corresponds to the front camera view of the vehicle j_k, and, eventually, other information such as the vehicle's current speed. The objective is to find $u_k^{\mathrm{h}*}\ \forall k$ such that, given the Local Motion Planner, (4.1) is solved. As discussed in Sec. 4.1 and Sec. 4.2, given the challenges in modeling human behavior, Interactive Imitation Learning (IIL) arises as an appealing and effective approach to tackle this problem, since it allows to data-efficiently and robustly learn behaviors from humans.

Interactive Imitation Learning Formulation

In IIL, a human that acts as a teacher is involved in the learning process of an agent. Feedback signals h_k are generated by the human to modify the learner's policy towards a desired behavior in an online learning manner. The context of autonomous navigation provides a framework where humans, by driving a vehicle, are able to execute the actions which they consider to be the best for a given state. Consequently, it comes natural to use feedback in the form of demonstrations.

In this work, a Learning from Interventions scheme is employed, i.e., every time the human considers the agent to be executing an erroneous behavior, the teacher takes control over the agent's actions until it gets back into a region where the observed behavior is the desired one. The data gathered in these interventions is used as demonstrations for improving the agent's behavior following a supervised learning approach. The teacher's feedback is represented by two variables: 1) i, a Boolean that indicates if the human is intervening (if $i = 1$, the human intervenes;

if $i = 0$, s/he does not), and 2) $u_k^{\mathrm{h}*}$, which corresponds to the teacher's optimal action for the high-level controller to take (i.e., these actions follow the expert policy π^{exp}). For $i = 1$, the feedback is defined as $h_k = u_k^{\mathrm{h}*}$; for $i = 0$, it is not defined.

In practice, this works as follows: initially, the VG creates a velocity reference u_k^{h} that the Local Motion Planner tracks (along with a set of way points). The human only observes the behavior of the AV ($i = 0$), as long as s/he considers that it is adequate. Every time the planner generates undesired control commands, the human (indirectly) takes control over it ($i = 1$) by overwriting the output of the VG with the correct velocity reference $u_k^{\mathrm{h}*}$. The data from these interventions (i.e., trajectories containing every state-action pair $[s_k^{\mathrm{h}}, u_k^{\mathrm{h}*}]$) is collected, and employed to improve the agent's behavior.

Eq. (6.3) can be solved as an Imitation Learning problem $\forall h_k$ when $i = 1$ (i.e., for every state-action pair collected from the interventions). If, every time the teacher intervenes, the demonstrated trajectories are stored in a dataset $\mathcal{D}$, (6.3) can be solved iteratively by sampling B trajectories with length K from $\mathcal{D}$ in every iteration and minimizing

$$\mathcal{L}(\theta) = -\frac{1}{B}\sum_{b=1}^{B}\sum_{k=0}^{K} \ln \pi_\theta^{\mathrm{h}}(u_{b,k}^{\mathrm{h}*} | s_{b,k}^{h}) \tag{4.2}$$

through gradient descent [139]. Note that this formulation assumes that π_θ^{h} is a stochastic policy, but in this work π_θ^{h} is deterministic. However, if we assume that the optimized distribution is Gaussian with a fixed variance, the mean of this distribution can be equivalently obtained (and represented by the deterministic policy π_θ^{h}) by minimizing the Mean Squared Error (MSE) [2, 51]

$$\mathcal{L}(\theta) = \frac{1}{B}\sum_{b=1}^{B}\sum_{k=0}^{K} \left(u_{b,k}^{\mathrm{h}*} - \pi_\theta^{\mathrm{h}}(s_{b,k}^{\mathrm{h}})\right)^2. \tag{4.3}$$

This optimization process does not mention the constraints shown in (6.3) because they are implicitly captured in the demonstration data (given that it was collected following actions generated by the Local Motion Planner).

iDAgger

As depicted in Sec. 4.2, we use an IIL method based on demonstrations similar to the one described by [56] that solves (4.3); however, no name was provided by the authors to this method specifically, as they employed a subgroup of modules from a larger framework introduced by [137]. Hence, we will refer to it as iDAgger (for intervention DAgger). Note that similar ideas have been employed in other works as well [46, 51, 53].

iDAgger (Algorithm 4) generates a dataset $\mathcal{D}$ online using feedback, in the form of interventions, provided by a human teacher. The state-action pairs generated in every intervention, i.e., $[s_k^{\mathrm{h}}, u_k^{\mathrm{h}*}]$, by the human are aggregated to $\mathcal{D}$. Every b time steps, the learner updates its policy π_θ^{h} by sampling a subset of $\mathcal{D}$

and minimizing (4.3). Furthermore, π_θ^{h} can be initialized from an initial set of demonstrations collected offline. As shown by [16], the policies learned with this type of online learning algorithm are guaranteed to perform well (i.e., trajectory cost grows linearly in the task horizon and imitation error) under the intervention data distribution.

Algorithm 4 iDAgger

Require: $\mathcal{D}$ with initial demonstrations, pre-trained policy π_θ^{h} and policy update period b
for $k = 1, 2, \ldots$ **do**
  observe s_k^{h}
  get intervention signal i
  if i is True **then**
    get feedback $h_k \leftarrow u_k^{\mathrm{h}*}$
    aggregate $\{s_k^{\mathrm{h}}, h_k\}$ to $\mathcal{D}$
    $u_k^{\mathrm{h}} \leftarrow h_k$
  else
    $u_k^{\mathrm{h}} \leftarrow \pi_\theta^{\mathrm{h}}(s_k^{\mathrm{h}})$
  execute u_k^{h}
  update π_θ^{h} from $\mathcal{D}$ **if** $\mathrm{mod}(k, b)$ is 0

4.4.3. Local Motion Planner

We built upon the MPC formulation provided by the Model Predictive Contour Control (MPCC) [112] to generate control commands enabling the AV to follow a reference path provided by a global path planner (e.g., Rapidly-exploring Random Trees (RRT) [140]) and the forward velocity reference while satisfying dynamic and collision constraints when a feasible solution is found.

Vehicle Model

We use a kinematic bicycle model for the AV with state $s^{\mathrm{l}} = [x, y, \phi, v]$, where x and y are the agent's Cartesian position coordinates, ϕ the heading angle and v the forward velocity fixed in a global inertial frame $\mathcal{W}$. The model is described as follows:

$$\begin{aligned} \dot{x} &= v\cos(\phi + \beta) \\ \dot{y} &= v\sin(\phi + \beta) \\ \dot{\phi} &= \frac{v}{l_r}\sin(\beta) \\ \dot{v} &= u^a \\ \beta &= \arctan\left(\frac{l_r}{l_f + l_r}\tan\left(u^\delta\right)\right) \end{aligned} \tag{4.4}$$

where β is the velocity angle and u^{l} is the vehicle control input composed by the forward acceleration u^a and steering angle u^δ, $u^{\mathrm{l}} = [u^a, u^\delta]$. l_r and l_f are

the distances of the rear and front tires from the center of gravity of the vehicle, respectively, and are assumed to be identical.

Cost Function

The velocity reference v^{ref} generated by the VG allows controlling the AV driving behavior directly: high-velocity reference values lead to highly aggressive behavior while low-velocity reference values lead to cautious driving behavior. Hence, we design the local planner's cost function as follows:

$$\begin{aligned} J(s_k^{\text{l}}, u_k^{\text{l}}, \lambda_k) = &\left\| e_k^c(s_k^{\text{l}}, \lambda_k) \right\|_{q_c} + \left\| e_k^l(s_k^{\text{l}}, \lambda_k) \right\|_{q_l} \\ &+ \left\| v_k^{\text{ref}} - v_k \right\|_{q_v} + \left\| u_k^a \right\|_{q_u} + \left\| u_k^\delta \right\|_{q_\delta} \end{aligned} \tag{4.5}$$

where $\mathcal{Q} = \{q_c, q_l, q_v, q_u, q_\delta\}$ denotes the set of cost weights and λ_k is the estimated progress along the reference path. First, we minimize the contour error (e_k^c) and lag error (e_k^l), to track the reference path closely. The contour error quantifies how much the ego vehicle deviates from the reference path laterally, whereas lag error is the deviation of the ego vehicle from the reference path longitudinally. Please refer to [112] for more details on path parameterization and tracking error. The third term, $\|v_k^{\text{ref}} - v_k\|$, motivates the planner to follow the velocity reference provided by the Visual Guidance system closely. Finally, we add a quadratic penalty to the control commands, u_k^a and u_k^δ, to generate smooth trajectories.

Dynamic Obstacle Avoidance

First, we approximate the AV's occupied area, $\mathcal{A}^{\text{ego}}$, as union of n_c circles, i.e., $\bar{A}^{\text{ego}} \subseteq \bigcup_{c\in\{1,\dots,n_c\}} \mathcal{A}_c$, where $\mathcal{A}_c$ represents the c^{th} circle's area with radius r. For the other vehicles, the occupied area by the i^{th} vehicle, $\mathcal{A}^i$, is approximated by an ellipse of semi-major axis a_i, semi-minor axis b_i and orientation ϕ. Then, we define a set of non-linear constraints enforcing that each AV's circle c does not intersect with the i^{th} vehicle's elliptical:

$$c_k^{i,c}(s_k^{\text{l}}, s_k^{\text{l}_i}) = \begin{bmatrix} \Delta x_k^c \\ \Delta y_k^c \end{bmatrix}^{\text{T}} R(\phi)^{\text{T}} \begin{bmatrix} \frac{1}{\alpha^2} & 0 \\ 0 & \frac{1}{\beta^2} \end{bmatrix} R(\phi) \begin{bmatrix} \Delta x_k^c \\ \Delta y_k^c \end{bmatrix} > 1, \tag{4.6}$$

The parameters Δx_k^c and Δy_k^c represent x-y relative distances between the disc c and the ellipse i for planning step k. α and β are function of the AV's radius and the other vehicle's semi-major and semi-minor axis, respectively, and an enlarging coefficient ensuring collision avoidance, with $\alpha = a + r_{\text{disc}} + \epsilon$ and $\beta = b + r_{\text{disc}} + \epsilon$. We refer the reader to [121] for details on how ϵ is computed.

Road boundaries

To compute motion plans respecting the road boundaries, we employ constraints on the AV's lateral distance (i.e., contour error) with respect to the reference path ensuring that the vehicle stays within the road limits:

$$-w_{\text{left}}^{\text{road}} \leq e_k^c(s_k^{\text{l}}) \leq w_{\text{right}}^{\text{road}} \tag{4.7}$$

where $w^{\text{road}}_{\text{left}}$ and $w^{\text{road}}_{\text{right}}$ are the left and right road limits, respectively.

MPCC Formulation

We formulate the motion planner as a Receding Horizon Trajectory Optimization problem with planning horizon H conditioned on the following constraints:

$$
\begin{aligned}
u^{\text{l}*}_{0:H-1} = \min_{u^{\text{l}}_{0:H-1}} & \sum_{k=0}^{H-1} J(s^{\text{l}}_k, u^{\text{l}}_k, \lambda_k) + J(s^{\text{l}}_H, \lambda_H) \\
\text{s.t.} \quad & s_{k+1} = f(s^{\text{l}}_k, u^{\text{l}}_k), \\
& \lambda_{k+1} = \lambda_k + v_k \Delta t \\
& -w^{\text{road}}_{\text{left}} \leq e^c(s^{\text{l}}_k) \leq w^{\text{road}}_{\text{right}} \\
& c^{i,c}_k(s^{\text{l}}_k, s^{\text{l}_i}_k) > 1 \;\; \forall c \in \{1, \dots, n_c\}, \\
& u^{\text{l}}_k \in \mathcal{U}^{\text{l}}, \quad s^{\text{l}}_k \in \mathcal{S}^{\text{l}}, \\
& \forall k \in \{0, \dots, H\},
\end{aligned}
\tag{4.8}
$$

where Δt is the discretization time and $u^{\text{l}*}_{0:H-1}$ the locally optimal control sequence for H time-steps. The solver employed attempts to find a solution for the MPCC problem for a fixed number of iterations. If a feasible solution is found, we apply only the first control input for each step and recompute a new solution in the next iteration considering new observed information in a receding horizon fashion. Otherwise we employ a safety control command $u^{\text{l}}_{\text{safety}}$.

4.4.4. Social-MPCC

Overall, the Social-MPCC framework utilizes the Visual Guidance policy to provide a velocity reference that controls the vehicle's behavior through the cost function that is optimized by the Local Motion Planner. Imitation Learning is used to optimize the VG's parameters to model human behavior.

Algorithm 5 presents the overall framework. First, iDAgger (Algorithm 4) is initialized to start the training of the Visual Guidance (line 2). Then, at the beginning of each episode, the reference path $\mathcal{P}$ to be followed by the MPCC is obtained from a global planner (line 4). Afterwards, for every time step of each episode, the velocity reference $v^{\text{ref}}_k = u^{\text{h}}_k$ is received from iDAgger (lines 8-9) and fed to the MPCC to compute the control command u^{l}_k (line 10). Finally, u^{l}_k controls the AV (line 11). Each episode ends if a collision or a deadlock is detected. Moreover, the human teacher can also request the end of the episode (lines 12-15).

4.5. Experiments

This section presents simulation results in a realistic urban scenario populated with pedestrians and other vehicles (Fig. 4.2). First, we quantify the performance throughout the training procedure (Sec. 4.5.2). Then, we show a qualitative

Algorithm 5 Social-MPCC

Require: global planner, iDAgger (algorithm 4), MPCC and number of episodes n_{episodes}
run Visual Guidance training with iDAgger in separate thread
while $i_{\text{episode}} < n_{\text{episodes}}$ **do**
  get reference path $\mathcal{P}$ from a global planner
  for $k = 1, 2, \dots$ **do**
    get states s_k^{h}, s_k^{l} from environment
    send s_k^{h} to iDAgger (algorithm 4, line 3)
    receive u_k^{h} from iDAgger (algorithm 4, line 12)
    set $v_k^{\text{ref}} \leftarrow u_k^{\text{h}}$
    compute $u_k^{\text{l}} \leftarrow \pi^{\text{l}} =$ MPCC$(v_k^{\text{ref}}, s_k^{\text{l}}; \mathcal{P})$ ((4.8))
    execute u_k^{l} in vehicle
    compute done $\leftarrow$ collision/deadlock detected **or** teacher request
    if done **then**
      increment i_{episode}
      break

(a) Top view city. **(b)** Ego vehicle's back view.

Figure 4.2.: CARLA simulation environment.

evaluation of the method (Sec. 4.5.3). Finally, we present performance results (Sec. 4.5.4) of the proposed method against baselines.

4.5.1. Experimental setup

Simulation results were carried out on an Intel Core i9, 32GB of RAM CPU @ 2.40GHz. The non-linear and non-convex MPCC problem presented in Sec. 4.4.3 was solved using the ForcesPro [142] solver. The Visual Guidance was modeled with a Deep Neural Network (DNN) implemented and optimized in TensorFlow 2 [143]. We used the open-source CARLA simulator [115] to create the simulation

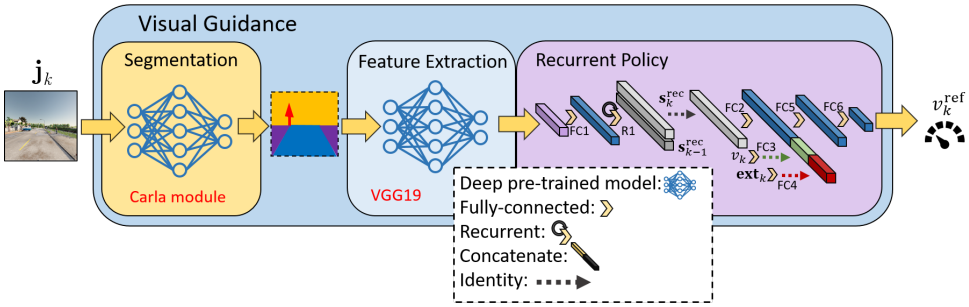


Figure 4.3.: Visual Guidance architecture (Sec. 4.5.1). The *segmentation* module receives the image j_k and provides a segmented image to the *feature extraction* module. The *recurrent policy* takes these features as an input and generates the velocity reference v_k^{ref}. In this work, a CARLA module was used for the segmentation module and a pre-trained VGG19 network was used for feature extraction. The recurrent policy consists of six fully-connected layers (FCX), and one recurrent layer (R1). FC1, FC2, FC3, FC4 and FC5 use *Leaky ReLU* [141] as activation function. FC1 has 150 neurons and the other layers have 1000 neurons. FC6 has a linear activation and one neuron, as it is the output layer. The hidden state size of R1 is 150. The variable ext_k corresponds to extensions to the input of the network, such as traffic light state and/or information about where to go when learning in an end-to-end manner.

environment where the *Traffic Manager* module was employed to simulate other vehicles and the *AI controller* to control the pedestrians. The complete framework was interfaced using the Robot Operating System (ROS) [144].

Visual Guidance: Deep Neural Network Architecture

The DNN architecture employed to represent the VG is defined by the mapping $\pi_\theta^{\text{h}} : s_k^{\text{h}} \mapsto v_k^{\text{ref}}$ (Fig. 4.3). The VG has to 1) be able to process images j_k, 2) deal with partial observability due to the absence of temporal information in j_k. Moreover, to further improve the input state of the VG, the vehicle's speed v_k can also be provided to the network. Convolutional layers were employed to process j_k and recurrent layers to deal with the mentioned partial observability [145]. Hence, the high-level state was defined as $s_k^{\text{h}} = [j_k, s_k^{\text{rec}}, v_k]$, where s_k^{rec} corresponds to the hidden state of the recurrent layers.

To increase the generalization properties and data efficiency of the network, two techniques were employed: 1) semantic segmentation [146], and 2) Transfer Learning (TL) [147]. The input image j_k was semantically segmented using a CARLA module; however, in practice, DNN models such as SegNet [148] or DeepLab [149] can be employed. For TL, we employed a VGG [150] model pretrained on ImageNet [151]. The last layer of the VGG was removed and replaced with recurrent and fully connected layers with trainable parameters to be optimized with (4.3). Hence, the VGG was used as a state representation/feature

Table 4.1.: Key hyperparameters used for local planner, learning algorithm, and simulated environment.

	Hyperparameter	Value
Simulated environment	N_{Cars}	100
	$N_{\text{Pedestrians}}$	200
	$N_{\text{Supervised}}$	5
	Camera FoV	90.0°
Visual Guidance	Interactive training time	2 hours
	Image resolution	64×64 pixels
	Feature extractor	VGG
	Recurrent layer	LSTM
	Optimizer	Adam
	Learning rate	$1e-5$
	Batch size	100
	Training iterations per episode	1000
Local planner	$Q = \{q_c, q_l, q_v, q_u, q_\delta\}$	$\{0.1, 0.2, 1.0, 1.2, 0.1\}$
	Number solver iterations	500
	u_{safety}	$[-2.0, 0.0]$
	Solver method	Primal-Dual Interior-Point Method

extraction machine and its weights were not modified during the optimization of π_θ^{h}. LSTM [98] layers were employed as the recurrent layers of the network. Finally, to optimize (4.3) with a recurrent DNN, we employed the *bootstrapped random updates* method [102].

Table 4.1 presents the values of the hyperparameters used for the local planner, training algorithm and simulation environment.

4.5.2. Training procedure

Fig. 4.2 shows the training environment. At the beginning of each episode, N_{cars} cars and $N_{\text{pedestrians}}$ pedestrians are spawned in random locations. The AV receives a sequence of way-points towards a random goal position provided by the CARLA *Route Planner*. An episode ends if the AV collides, if it reaches the goal position successfully, if a deadlock occurs, or if a teacher request is received.

Fig. 4.4 presents the VG's learning performance. The first plot shows the amount of time the teacher corrected the policy's actions, and the second plot the moving average of the mean squared error between the teacher and the policy's actions. The training procedure incorporates two phases: collection of an initial set of demonstrations used to train an initial policy, and the interactive learning process (as shown in Algorithm 4). Given that during the first $N_{\text{supervised}}$ steps the teacher provides feedback continuously, the amount of demonstration time grows linearly, as depicted in the upper plot of Fig. 4.4 from $t = 0$ s to $t \approx 1500$ s. Afterwards,

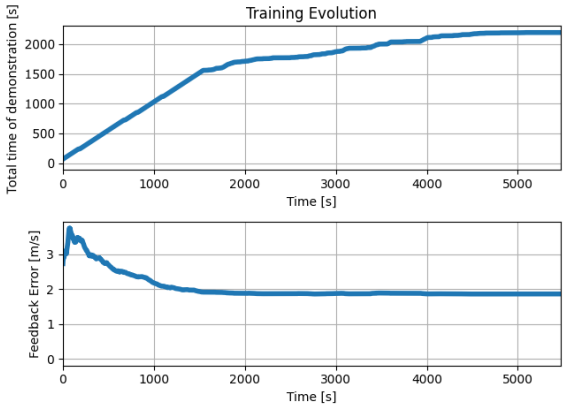


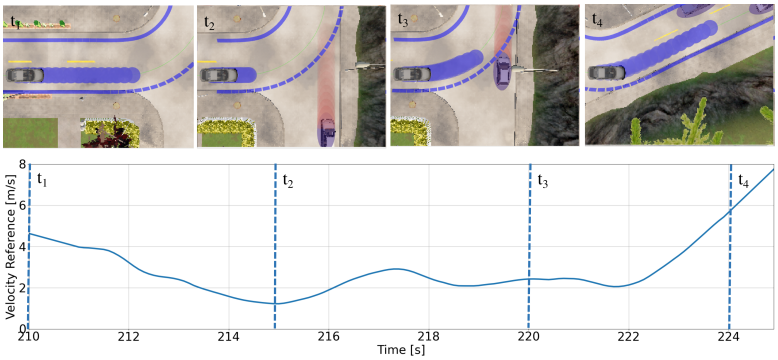
Figure 4.4.: Accumulated feedback and policy error evolution during training. The top plot shows the amount of time the teacher had to correct the policy's action and the bottom the average policy's action error.

feedback is only provided when the policy acts erroneously, which will depend on the episode's complexity and the novelty it provides. After $t \approx 4500$ s, the total demonstration time remains constant, showing that the policy is performing well and the teacher does not need to provide more feedback.

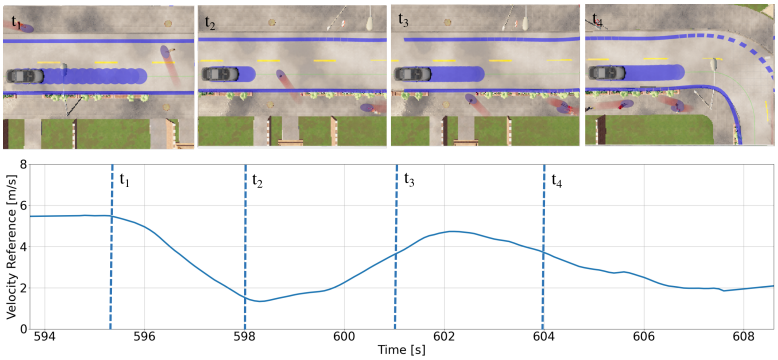
The bottom plot shows that the moving average of the root mean squared error between the VG's action and the provided supervised action reduces over time, stabilizing at around 2 m/s (note that $0 \text{ m/s} \leq u_k^{\text{h}} \leq 8 \text{ m/s}$). Although this error may seem large, this result is expected, since humans are not always consistent about the feedback they provide [152, 153], and data with irreducible error is collected. Nevertheless, this is not considered to be an issue in this experiment, as, when the mean squared error is minimized, the modes present in the data are averaged, which was not observed as being detrimental. Inconsistencies only arose in situations where their average would not jeopardize the safety of the learned behavior (e.g., cruise speed in long roads or the response time to start accelerating after a green light), while the general rules of driving (e.g., stopping at red lights or if there are pedestrians crossing the street) were always respected.

4.5.3. Qualitative Results

This section analyzes the AV behavior using our method for two driving scenarios. In the first scenario, depicted in Fig. 4.5a, the AV approaches a crossing area and has to perform a left-turn maneuver. Between t_1 and t_2, the VG continuously reduces the velocity reference as the AV approaches the crossing area yielding to the vehicle coming from the right. Then, the velocity reference initially increases, motivating the AV to cross the road, between $t = 215$ s and $t = 217$ s, and keeps a continuous reference while turning left, between $t = 217$ s and $t = 222$ s. Once the vehicle finishes turning left, approximately at $t = 222$ s, the velocity reference is



(a) Left-turn maneuver.



(b) Pedestrian crossing the road.

Figure 4.5.: The blue circles depict the MPCC's velocity reference provided by the VG and the blue lines the road constraints. The red circles depict the predicted constant velocity trajectory for the other vehicles or pedestrians.

increased again.

In the second scenario, depicted in Fig. 4.5b, a pedestrian crosses the road in front of the AV. The VG reduces the velocity reference to let the pedestrian cross, between $t \approx 595$ s and $t \approx 598$ s. Once the pedestrian finishes crossing, the reference is increased. Afterwards, to safely perform a right turn maneuver, the velocity reference is reduced.

More scenarios can be found in the attached video[3], where it is possible to appreciate that the exhibited behaviors resemble human driving.

4.5.4. Quantitative Results

The objective of this section is to study, quantitatively, the effect on the performance of a trajectory planner when optimization-based and learning-based

[3] Available at: https://youtu.be/Ph7v25mEg7c

methods are combined. To achieve this, we study three types of algorithms: 1) optimization-based only (MPCC), 2) optimization-based and learning-based combined (Social-MPCC with and without traffic information) and 3) completely data-driven (End-to-end learning with traffic information), which are described below:

- **MPCC:** Local Motion Planner with constant velocity reference.
- **Social-MPCC:** the proposed Social-MPCC framework.
- **Social-MPCC with traffic information:** Social-MPCC with traffic lights' information in its state.
- **End-to-end learning with traffic information:** same as before, but the AV's control $u^{\mathrm{l}} = [u^a, u^\delta]$ is learned using iDAgger alone.

To test the flexibility of the proposed framework, two variations of Social-MPCC are presented, one that is general to any autonomous driving scenario and one that is specific to driving in a city. In the general case, the structure of the VG is as explained in Sec. 4.5.1; in the specific case, the input is extended with the traffic lights' state (see Fig. 4.3). Note that, strictly speaking, the traffic lights' status is also fed to the neural network in the general case, as it can be perceived from few pixels in j_k when the AV approaches a traffic light. Nevertheless, to obtain real-time performance, it is necessary to limit the input's resolution; hence, the resolution of the images was not high enough to effectively use the traffic lights' information from them.

It is to be expected that Social-MPCC will perform better when traffic information is included into the system than when it is not. Therefore, to obtain a fair comparison against the end-to-end policy, the traffic lights' state is also employed in this case. Moreover, when the complete behavior is learned from data, it is also necessary to provide information to the network about where the vehicle should go, as in the other methods this information is given to the MPCC through $\mathcal{P}$. To achieve this, the network was provided with $\sin(\gamma)$ (in the same way as the traffic lights' state, see Fig. 4.3), where γ is the angle between the center of the vehicle and the next way-point located at distance of ~ 15 m from it.

Finally, to test the data efficiency of Social-MPCC, **only 2 hours** were employed for the complete learning process of the experiments, as opposed to other methods in the literature that can use $100\check{}200\times$ more human time (e.g., ~ 300 hours [154]). Table 4.2 compares the performance of the introduced methods in terms of number of collisions per traveled distance and percentage of deadlocks. In terms of computation performance, the VG (i.e., DNN) has an average computation time of 5.1 ± 0.9 ms, while the MPCC optimization problem ((4.8)) takes on average 3.0 ± 1.35 ms.

Discussion

From Table 4.2 it can be observed that Social-MPCC drastically improves the performance of MPCC with only 2 h of training. With the general Social-MPCC

Table 4.2.: Statistic results for 100 episodes of Social-MPCC compared to baselines.

	No. Collisions per km.	% of deadlocks
MPCC [112]	2.60	17
Social-MPCC	0.71	17
Social-MPCC with traffic lights	**0.37**	**13**
End-to-end with traffic lights	1.94	18

framework, it was possible to reduce the amount of collisions per kilometer by 3.66 times. Furthermore, in the case in which the traffic lights' information was provided to the VG, this value incremented to 7.03 and the percentage of deadlocks was reduced to 13%.

Deadlocks occurred when the vehicle was stationary for an extended period of time (600 time steps in this experiment). Hence, when no feasible solutions were found by the MPCC, the activation of $u^{\mathrm{l}}_{\mathrm{safety}}$ could have led to deadlocks. Interestingly, the VG also generated deadlocks. It was observed that the DNN could get stuck by constantly providing zero forward velocity reference to the local planner. Nevertheless, the combination of MPCC and VG did not increase the number of deadlocks when combined in Social-MPCC; furthermore, the number of deadlocks was reduced when the traffic lights' information was employed in the system. This occurred because an adaptive forward velocity reference can help the MPCC find solutions in cases where it would otherwise get stuck.

Finally, it was observed that the end-to-end learner achieved an acceptable performance; however, Social-MPCC showed to be superior after two hours of training. Increasing the action space of the VG to also include a steering angle reference makes the learning problem considerably harder. This is can be observed with both the number of collisions per kilometer (5.24× more) and the amount of deadlocks (1.38× more) that the end-to-end learner obtained.

Analysis Social-MPCC

Although IIL methods are very data efficient, it is still not possible to learn a flawless behavior in 2 h; moreover, assumptions in the MPCC's formulation may cause it to perform suboptimally. Hence, there are situations in which our method fails. We have visually inspected the training episodes and identified the main factors leading to failure (i.e., collisions). Table 4.3 presents the five main failure factors and their frequency considering a total of 100 episodes.

The categories in Table 4.3 are presented bellow:

- **Unusual situations:** Occasionally, the AV may get into situations that are not common, such as interacting with oddly shape vehicles or with multiple vehicles that got stuck and not moving, that are unlikely to by encountered during training. Therefore, the VG may not be trained in similar circumstances and consequently generate incorrect behaviors.

Table 4.3.: Analysis of failure episodes: number of episodes per factor leading to failure. We consider a total of 100 episodes.

Factor	№ episodes
Unusual situations	5
Outside the camera FoV	1
Wrong predictions	1
Small obstacles	8
Other agents contempt driving rules	1
Total	16/100

- **Outside the camera Field of View (FoV):** Due to the limited FoV of the first person view camera used by the VG, our system is not able to obtain all of the relevant visual information for driving in every situation. Hence, there are cases in which obstacles are not perceived on time, leaving the system too little time to react safely.
- **Wrong predictions:** The MPCC framework works under the assumption that other vehicles and pedestrians have constant velocities. This assumption does not hold in every situation, which may cause failures.
- **Small obstacles:** Small obstacles, such as children and bicycles, are not always easily perceived by the VG. Furthermore, they are not frequently encountered by the AV, which makes it more challenging to properly learn about these cases during training. Therefore, our system was not fully robust in avoiding collisions with small obstacles.
- **Other agents contempt driving rules:** In some cases, other agents, such as vehicles or pedestrians, do not respect the driving rules. Other vehicles may ignore red lights or pedestrians may cross the street in places where they are not allowed to, inducing collisions with our system.

Small obstacles and unusual situations were the two most frequent types of failures. Both cases occurred, in large part, due to the limited number of episodes during which the policy was trained. More training time or data augmentation techniques would largely help to decrease the frequency of these failures.

The rest of the failure cases did not affect the performance of the AV to a great extent, as they happened once each. However, the proposed framework could be extended to reduce these types of collisions. The failure episodes due to limited FoV can be solved by, for instance, by incorporating 360° visual information, allowing the policy to reason about the surrounding environment completely. Secondly, failures due to wrong predictions can be solved with a high-fidelity prediction model [155] reasoning about interaction and environment constraints. Lastly, in the cases where other agents contempt driving rules, the local planner's safety bounds can be increased; moreover, more training time can help make the VG be more robust.

4.6. Conclusions

In this chapter we presented a framework, Social-MPCC, that combines an optimization-based control method (MPCC) with a learning-based method (iDAgger) for learning and executing safe, human-like, driving behaviors. Learning human-like driving behaviors is a desired feature for AVs, as they produce trust in other human agents and facilitate collision avoidance by acting predictably. To achieve this, the forward velocity reference of a local trajectory planner is modified in real time by a Visual Guidance system that learns, from humans, to control this variable using first-person view images of a vehicle. The learning method follows an Interactive Imitation Learning training procedure that enables obtaining well-performing policies in only two hours of human training time, as opposed to other methods in the literature that require 100˜200× more human time.

The method was experimentally validated in a realistic simulator. Qualitative results show the capacity of the method to successfully encode human-like driving behaviors in the MPCC. Quantitative results compare the performance of Social-MPCC against baselines that are optimization-based (i.e., MPCC) or learning-based only (i.e., end-to-end iDAgger). Social-MPCC substantially improved the performance of MPCC, both in terms of number of collisions and deadlocks. Furthermore, after two hours of interactive training, the proposed method showed to be superior to the end-to-end learning method. Finally, Social-MPCC achieved real-time performance, which allows it to be implemented on a real platform.

Future works can extend Social-MPCC to control a larger family of high-level control variables of the MPCC with the Visual Guidance. For instance, way points could be locally modified to enforce specific behaviors. Furthermore, modifying the weights in the MPCC's cost function could also be employed for this purpose. Finally, the proposed framework could also be extended with other Interactive Learning techniques: for example, corrective advice could be used to teach behaviors that may be challenging to demonstrate [65].

5

Stable Motion Primitives via Imitation and Contrastive Learning

This chapter is based on the publication:

R. Pérez-Dattari and J. Kober. "Stable Motion Primitives via Imitation and Contrastive Learning". In: *IEEE Transactions on Robotics* (2023)

5.1. Introduction

Imitation Learning (IL) provides a framework that is intuitive for humans to use, without requiring them to be robotics experts. It allows robots to be programmed by employing methods similar to the ones humans use to learn from each other, such as demonstrations, corrections, and evaluations. This significantly reduces the resources needed for building robotic systems, making it particularly appealing for real-world applications (e.g., Fig. 5.1).

Nevertheless, due to their data-driven nature, IL methods often lack guarantees, such as ensuring that a robot's motion always reaches its target, independently of its initial state (e.g., Fig. 5.2). This can be a major limitation for implementing methods in the real world since it can lead to failures and/or accidents.

To tackle these challenges, we can model motions as dynamical systems whose evolution describes a set of human demonstrations [157–159]. This is advantageous because 1) the model depends on the robot's state and it is learned offline, enabling the robot to adapt to changes in the environment during task execution, and 2) dynamical systems theory can be employed to analyze the behavior of the motion and provide guarantees.

In this work, we focus on learning dynamical systems from demonstrations to model *reaching motions*, as a wide range of tasks requires robots to reach goals, e.g., hanging objects, pick-and-place of products, crop harvesting, and button pressing. Furthermore, these motions can be sequenced to model cyclical behaviors, extending their use for such problems as well [160].

A reaching motion modeled as a dynamical system is considered to be globally asymptotically stable if the robot always reaches its goal, independently of its initial state. In this work, we will refer to such systems as *stable* for short. Notably, by employing dynamical systems theory, stability in reaching motions can be enforced when learning from demonstrations, providing guarantees to these learning frameworks.

In the literature, there is a family of works that use this approach to learn stable motions from demonstrations [158, 159, 161]. However, these often constrain the structure of their learning models to meet certain conditions needed to guarantee the stability of their motions, e.g., by enforcing the learning functions to be invertible [159, 162, 163] or positive/negative definite [158, 164, 165]. Although these constraints ensure stability, they limit the applicability of the methods to a narrow range of models. For example, if a novel promising Deep Neural Network (DNN) architecture is introduced in the literature, it would not be straightforward/possible to use it in such frameworks, since, commonly, DNN architectures do not have this type of constraints. Furthermore, the learning flexibility of a function approximator is limited if its structure is restricted, which can hurt its accuracy performance when learning motions.

Hence, in the context of DNNs[1], we propose a novel method for learning stable motions without constraining the structure of the function approximator. To

[1] We understand DNNs as a collection of machine learning algorithms that learn in a hierarchical manner, i.e., the function approximator consists of a composition of multiple functions, and are optimized by means of backpropagation [166].

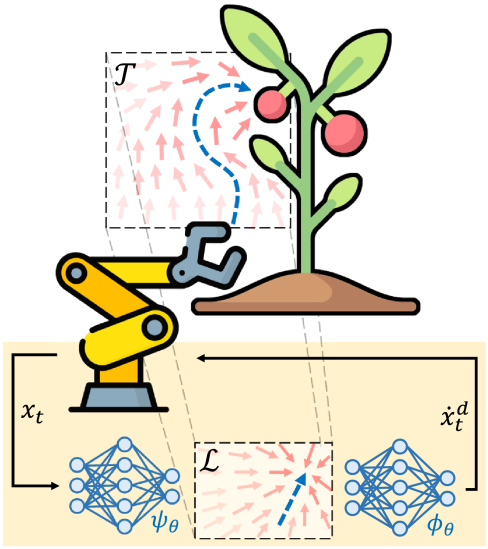


Figure 5.1.: Overview of a motion model learned using the proposed framework. The blue trajectory in the task space $\mathcal{T}$, shows the movement of the robot's end effector when starting from its current state x_t. The evolution of this trajectory is defined by the dynamical system $\dot{x}_t^d = \phi_\theta(\psi_\theta(x_t))$, which is represented with a vector field of red arrows in the rest of the space. Using Contrastive Learning, this system is coupled with a well-understood and stable dynamical system in the latent space $\mathcal{L}$ that ensures its stability.

achieve this, we introduce a contrastive loss [167, 168] to enforce stability in dynamical systems modeled with arbitrary DNN architectures. This is achieved by transferring the stability properties of a simple, stable dynamical system to the more complex system that models the demonstrations (see Fig. 5.1). To the best of our knowledge, this is the first approach that learns to generate stable motions with DNNs without relying on a specific architecture type.

We validate our method using both simulated and real-world experiments, demonstrating its ability to successfully scale in terms of the order and dimensionality of the dynamical system. Furthermore, we show its capabilities for controlling a 7DoF robotic manipulator in both joint and end-effector space. Lastly, we explore potential extensions for the method, such as combining motions by learning multiple systems within a single DNN architecture.

The chapter is organized as follows: related works are presented in Sec. 5.2. Sec. 5.3 describes the background and problem formulation of our method. Sec. 5.4 develops the theory required to introduce the contrastive loss, introduces it, and explains how we employ it in the context of Imitation Learning. Experiments and results are divided into sections 5.5, 5.6, and 5.7. Sec. 5.5 validates our method

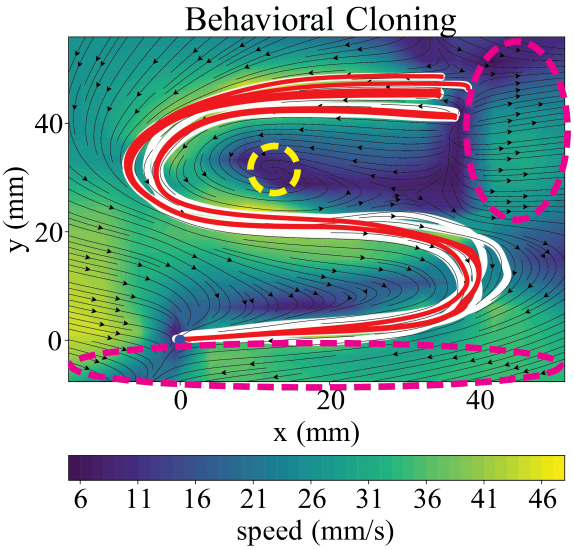


Figure 5.2.: Example of a motion learned using Behavioral Cloning. White curves represent demonstrations. Red curves represent learned motions when starting from the same initial positions as the demonstrations. The arrows indicate the vector field of the learned dynamical system. The yellow dotted line shows a region with a spurious attractor. The magenta lines show regions where the trajectories diverge away from the goal.

using datasets of motions modeled as first-order and second-order dynamical systems. These motions are learned from real data, but they are evaluated without employing a real system. Sec. 5.6 validates the method in a real robot, and Sec. 5.7 studies possible ways of extending it. Finally, the conclusions are drawn in Sec. 5.8.

5.2. Related Work

Several works have approached the problem of learning motions modeled as dynamical systems from demonstrations while ensuring their stability. By observing if these works employ either *time-varying* or *time-invariant* dynamical systems, we can divide them into two groups. In time-varying dynamical systems, the evolution of the system explicitly depends on time (or a phase). In contrast, time-invariant dynamical systems do not depend on time *directly*, but only through its time-varying input (i.e., the state of the system). The property of a system being either time-invariant or not, conditions the type of strategies that can be employed to enforce its stability. Hence, for this work, it makes sense to make a distinction between these systems.

One seminal work of IL that addresses stability for time-varying dynamical

systems introduces Dynamical Movement Primitives (DMPs) [157]. This method takes advantage of the time-dependency (via the phase of the *canonical system*) of the dynamical system to enforce its nonlinear part, which captures the behavior of the demonstration, to vanish as time goes to infinity. Then, they build the remainder of the system to be a function that is well-understood and stable by construction. Hence, since the nonlinear part of the motion will eventually vanish, its stability can be guaranteed. In the literature, some works extend this idea with probabilistic formulations [169, 170], and others have extended its use to the context of DNNs [171–173].

These time-varying dynamical system approaches are well-suited for when the target motions have clear temporal dependencies. However, they can generate undesired behaviors when encountering perturbations (assuming the time/phase is not explicitly modulated), and they lack the ability to model different behaviors for different regions of the robot's state space. In contrast, time-invariant dynamical systems can easily address these shortcomings, but they can be more challenging to employ when motions contain strong temporal dependencies. Therefore, IL formulations with such systems are considered to be complementary to the ones that employ time-varying systems [5, 174]. In this work, we focus on time-invariant dynamical systems.

An important family of works has addressed the problem of modeling stable motions as time-invariant dynamical systems. These approaches often constrain the structure of the dynamical systems to ensure Lyapunov stability by design. In this context, one seminal work introduces the Stable Estimator of Dynamical Systems (SEDS) [158]. This approach imposes constraints on the structure of Gaussian Mixture Regressions (GMR), ensuring stability in the generated motions.

Later, this idea inspired other works to explicitly learn Lyapunov functions that are consistent with the demonstrations and *correct* the transitions of the learned dynamical system such that they are stable according to the learned Lyapunov function [164, 165, 175]. Furthermore, several extensions of SEDS have been proposed, for instance by using physically-consistent priors [176], contraction theory [177] or diffeomorphisms [178].

Moreover, some of these ideas, such as the use of contraction theory or diffeomorphisms have also been used outside the scope of SEDS. Contraction theory ensures stability by enforcing the distance between the trajectories of a system to reduce, according to a given metric, as the system evolves. Hence, it has been employed to learn stable motions from demonstrations [179, 180]. In contrast, diffeomorphisms can be employed to transfer the stability properties of a stable and well-understood system, to a complex nonlinear system that models the behavior of the demonstrations. Hence, this strategy has also been employed to learn stable motions from demonstrations [159, 162, 163]. As we explain in Section 5.4.2, our method is closely related to these approaches. It is worth noting that of the mentioned strategies, only [163] models stable stochastic dynamics. However, this concept could also be explored with other encoder-decoder stochastic models, e.g., [169].

Understandably, all of these methods constrain some part of their learning

framework to ensure stability. From one point of view, this is advantageous, since they can guarantee stability. However, in many cases, this comes with the cost of reducing the flexibility of the learned motions (i.e., loss in accuracy). Notably, some recent methods have managed to reduce this loss in accuracy [159, 163]; however, they are still limited in terms of the family of models that can be used with these frameworks, which harms their scalability. Consequently, in this work, we address these limitations by enforcing the stability of the learned motions as a soft constraint and showing its effectiveness in obtaining stable, accurate, and scalable motions.

5.3. Preliminaries

5.3.1. Dynamical Systems as Movement Primitives

In this work, we model motions as nonlinear time-invariant dynamical systems defined by the equation

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

where $x \in \mathbb{R}^n$ is the system's state and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear continuous and continuously differentiable function. The evolution defined by this dynamical system is transferred to the robot's state by tracking it with a lower-level controller.

5.3.2. Global Asymptotic Stability

We are interested in solving reaching tasks. From a dynamical system perspective, this means that we want to construct a system where the goal state $x_g \in \mathbb{R}^n$ is a *globally asymptotically stable* equilibrium. An equilibrium x_g is globally asymptotically stable if $\forall x \in \mathbb{R}^n$,

$$\lim_{t\to\infty} ||x - x_g|| = 0. \tag{5.2}$$

Note that for this condition to be true, the time derivative of the dynamical system at the attractor must be zero, i.e., $\dot{x} = f(x_g) = 0$.

For simplicity, we use the word *stable* to refer to these systems.

5.3.3. Problem Formulation

Consider the scenario where a robot aims to learn a reaching motion, in a given space $\mathcal{T} \subset \mathbb{R}^n$ and with respect to a given goal $x_g \in \mathcal{T}$, based on a set of demonstrations $\mathcal{D}$. The robot is expected to imitate the behavior shown in the demonstrations while always reaching x_g, regardless of its initial state.

The dataset $\mathcal{D}$ contains N demonstrations in the form of trajectories τ_i, such that $\mathcal{D} = (\tau_0, \tau_1, ..., \tau_{N-1})$. Each one of these trajectories contains the evolution of a dynamical system with discrete-time states $x_t \in \mathcal{T}$ when starting from an initial state x_0 and it transitions for T time steps t of size Δt. Hence, $\tau_i = (x_0^i, x_1^i, ..., x_{T-1}^i)$, where T does not have to be the same for every demonstration, and here we added the superscript i to the states to explicitly

indicate that they belong to the trajectory τ_i. Note, however, that the state superscript will not be used for the remainder of the chapter.

We assume that these trajectories are drawn from the distribution $p^*(\tau)$, where every transition belonging to a trajectory sampled from this distribution follows the *optimal* (according to the demonstrator's judgment) dynamical system f^*. On the other hand, the robot's motion is modeled as the parametrized dynamical system $f_\theta^{\mathcal{T}}$, which induces the trajectory distribution $p_\theta(\tau)$, where θ is the parameter vector.

Then, the objective is to find θ^* such that the distance between the trajectory distributions induced by the human and learned dynamical system is minimized while ensuring the stability of the motions generated with $f_\theta^{\mathcal{T}}$ towards x_g. This can be formulated as the minimization of the (forward) Kullback-Leibler divergence between these distributions [40], subject to a stability constraint of the learned system:

$$\theta^* = \arg\min_\theta \quad D_{\mathrm{KL}}\left(p^*(\tau)||p_\theta(\tau)\right) \tag{5.3a}$$

$$\text{s.t.} \quad \lim_{t\to\infty} ||x_t - x_\mathrm{g}|| = 0, \tag{5.3b}$$

$$\forall x_t \in \mathcal{T} \text{ evolving with } f_\theta^{\mathcal{T}}.$$

5.4. Method

We aim to learn motions from demonstrations modeled as nonlinear time-invariant dynamical systems. In this context, we present the *CONvergent Dynamics from demOnstRations* (CONDOR) framework. This framework learns the parametrized function $f_\theta^{\mathcal{T}}$ using human demonstrations and ensures that this dynamical system has a globally asymptotically stable equilibrium at x_g while being accurate w.r.t. the demonstrations.

To achieve this, we extend the Imitation Learning (IL) problem with a novel loss ℓ_{stable} based on Contrastive Learning (CL) [167] that aims to ensure the stability of the learned system. Hence, if the IL problem minimizes the loss ℓ_{IL}, our framework minimizes

$$\ell_{\mathrm{CIL}} = \ell_{\mathrm{IL}} + \lambda \ell_{\text{stable}}, \tag{5.4}$$

where $\lambda \in \mathbb{R}$ is a weight. We refer to ℓ_{CIL} as the *Contrastive Imitation Learning* (CIL) loss.

5.4.1. Structure of CONDOR

The objective of ℓ_{stable} is to ensure that $f_\theta^{\mathcal{T}}$ shares stability properties with a simple and well-understood system. We will refer to this system as $f^{\mathcal{L}}$, which is designed to be stable by construction. Consequently, if $f^{\mathcal{L}}$ is stable, then $f_\theta^{\mathcal{T}}$ will also be stable.

Since $f_\theta^{\mathcal{T}}$ is parametrized by a DNN, we can define $f^{\mathcal{L}}$ to reside in the output of one of the hidden layers of $f_\theta^{\mathcal{T}}$. This formulation might seem arbitrary; however, it will be shown later that it enables us to introduce the *stability conditions*, which

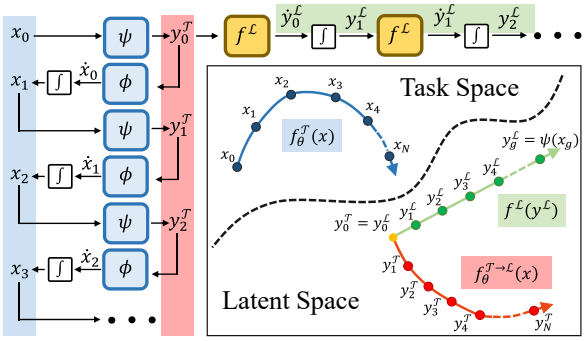


Figure 5.3.: Structure of CONDOR. We show an example of discrete-time trajectories generated with $f_\theta^{\mathcal{T}}$, $f^{\mathcal{L}}$ and $f_\theta^{\mathcal{T}\to\mathcal{L}}$ **before training** the DNN. Starting from an initial point x_0, a trajectory is generated in $\mathcal{T}$ using $f_\theta^{\mathcal{T}}$ (blue) and two trajectories are generated in $\mathcal{L}$. One of them follows $f_\theta^{\mathcal{T}\to\mathcal{L}}$ (red), and the other follows $f^{\mathcal{L}}$ (green).

5

serve as the foundation for designing ℓ_{stable}. Therefore, we define the dynamical system $f_\theta^{\mathcal{T}}$ as a composition of two functions, ψ_θ and ϕ_θ,

$$\dot{x}_t = f_\theta^{\mathcal{T}}(x_t) = \phi_\theta(\psi_\theta(x_t)), \tag{5.5}$$

$\forall x_t \in \mathcal{T}$. Note that $f_\theta^{\mathcal{T}}$ is a standard DNN with L layers. ψ_θ denotes layers $1...l$, and ϕ_θ layers $l+1...L$. We define the output of layer l as the latent space $\mathcal{L} \subset \mathbb{R}^n$. Moreover, for simplicity, although we use the same θ notation for both ψ_θ and ϕ_θ, each symbol actually refers to a different subset of parameters within θ. These subsets together form the full parameter set in $f_\theta^{\mathcal{T}}$.

Then, the dynamical system $f^{\mathcal{L}}$ is defined to evolve within $\mathcal{L}$[2]. This system is constructed to be stable at the equilibrium $y_{\text{g}} = \psi(x_{\text{g}})$, and can be described by

$$\dot{y}_t^{\mathcal{L}} = f^{\mathcal{L}}(y_t^{\mathcal{L}}), \tag{5.6}$$

$\forall y_t^{\mathcal{L}} \in \mathcal{L}$. Here, $y_t^{\mathcal{L}}$ corresponds to the *latent state variables* that evolve according to $f^{\mathcal{L}}$.

Lastly, it is necessary to introduce a third dynamical system. This system represents the evolution in $\mathcal{L}$ of the states visited by $f_\theta^{\mathcal{T}}$ when mapped using ψ_θ, which yields the relationship

$$\dot{y}_t^{\mathcal{T}} = f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t) = \frac{\partial \psi_\theta(x_t)}{\partial t}, \tag{5.7}$$

$\forall y_t^{\mathcal{T}} \in \mathcal{L}$, where $y_t^{\mathcal{T}}$ corresponds to the latent variables that evolve according to $f_\theta^{\mathcal{T}\to\mathcal{L}}$.

Fig. 5.3 summarizes the introduced dynamical systems.

[2]Note that before training, this system will not completely reside in $\mathcal{L}$, since it is allowed to evolve outside the image of ψ_θ.

5.4.2. Stability Conditions

The above-presented dynamical systems allow us to introduce the stability conditions. These conditions state that if $f_\theta^{\mathcal{T}\to\mathcal{L}}$ exhibits the same behavior as $f^{\mathcal{L}}$, and only x_{g} maps to $\psi_\theta(x_{\mathrm{g}})$, then $f_\theta^{\mathcal{T}}$ is stable. We formally introduce them as follows:

Theorem 1 (Stability conditions)**.** *Let $f_\theta^{\mathcal{T}}$, $f_\theta^{\mathcal{T}\to\mathcal{L}}$ and $f^{\mathcal{L}}$ be the dynamical systems introduced in Sec. 5.4.1. Then, x_{g} is a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}}$ if, $\forall x_t \in \mathcal{T}$,:*

1. $f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t) = f^{\mathcal{L}}(y_t^{\mathcal{T}})$,
2. $\psi_\theta(x_t) = y_{\mathrm{g}} \Rightarrow x_t = x_{\mathrm{g}}$.

Proof. Since $f^{\mathcal{L}}$ is globally asymptotically stable at y_{g}, condition 1) indicates that as $t \to \infty$, $y_t^{\mathcal{T}} = y_t^{\mathcal{L}} \to y_{\mathrm{g}}$. However, from condition 2) we know that $y_t^{\mathcal{T}} = \psi_\theta(x_t) = y_{\mathrm{g}}$ is only possible if $x_t = x_{\mathrm{g}}$. Hence, as $t \to \infty$, $x_t \to x_{\mathrm{g}}$. Then, x_{g} is globally asymptotically stable in $f_\theta^{\mathcal{T}}(x_t)$. □

Consequently, we aim to design ℓ_{stable} such that it enforces the stability conditions in the presented dynamical systems by optimizing ψ_θ and ϕ_θ.

Connection with Diffeomorphism-based methods

It is interesting to note that the stability conditions make ψ_θ converge to a *diffeomorphism* between $\mathcal{T}$ and $\mathcal{L}$ (proof in Appendix B.1). Consequently, our approach becomes tightly connected to methods that ensure stability using diffeomorphic function approximators [159, 162, 163]. However, differently from these methods, we do not require to take into account the structure of the function approximator and explicit relationships between $f_\theta^{\mathcal{T}}$ and $f_\theta^{\mathcal{T}\to\mathcal{L}}$.

5.4.3. Enforcing Stability

In this subsection, we introduce a method that enforces the stability conditions in $f_\theta^{\mathcal{T}}$.

First condition $\left(f_\theta^{\mathcal{T}\to\mathcal{L}} = f^{\mathcal{L}}\right)$

The first stability condition can be enforced by minimizing the distance between the states visited by the dynamical systems $f_\theta^{\mathcal{T}\to\mathcal{L}}$ and $f^{\mathcal{L}}$ when starting from the same initial condition. Hence, $\forall y_t^{\mathcal{T}}, y_t^{\mathcal{L}} \in \mathcal{L}$ a loss can be defined as $\ell_{\text{match}} = d(y_t^{\mathcal{T}}, y_t^{\mathcal{L}})$, where $d(\cdot,\cdot)$ is a distance function.

Second condition $(\psi_\theta(x_t) = y_{\mathrm{g}} \Rightarrow x_t = x_{\mathrm{g}})$

The second stability condition, however, can be more challenging to obtain, since we do not have a direct way of optimizing this in a DNN. Therefore, to achieve this, we introduce the following proposition:

Proposition 1 (Surrogate stability conditions)**.** *The second stability condition of Theorem 1, i.e., $\psi_\theta(x_t) = y_g \Rightarrow x_t = x_g, \forall x_t \in \mathcal{T}$, is true if:*

1. $f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t) = f^{\mathcal{L}}(y_t^{\mathcal{T}}), \forall x_t \in \mathcal{T}$ *(stability condition 1),*
2. $y_{t-1}^{\mathcal{T}} \neq y_t^{\mathcal{T}}, \forall x_t \in \mathcal{T} \setminus \{x_g\}$.

Proof. If $y_{t-1}^{\mathcal{T}} = \psi(x_{t-1}) = y_g$ the first condition implies that $y_g = y_{t-1}^{\mathcal{T}} = y_t^{\mathcal{T}}$, since $f^{\mathcal{L}}(y_g) = 0$. Consequently, given the second condition $y_{t-1}^{\mathcal{T}} \neq y_t^{\mathcal{T}}, \forall x_t \neq x_g$, this is only possible if $x_{t-1} = x_g$. □

In other words, Proposition 1 indicates that if the first stability condition is true; then, we can obtain its second condition by enforcing $y_{t-1}^{\mathcal{T}} \neq y_t^{\mathcal{T}}, \forall x_t \neq x_g$. Notably, by enforcing this, the stability conditions are also enforced. Consequently, we refer to the conditions of Proposition 1 as the *surrogate stability conditions*.

Then, it only remains to define a loss ℓ_{sep} that enforces the second surrogate stability condition in $f_\theta^{\mathcal{T}}$. However, before doing so, note that the surrogate conditions aim to push some points together (i.e., $y_t^{\mathcal{T}}$ and $y_t^{\mathcal{L}}$) and separate others (i.e., $y_{t-1}^{\mathcal{T}}$ and $y_t^{\mathcal{T}}$). Hence, this problem overlaps with the Contrastive Learning (CL) and Deep Metric Learning literature [168].

5

Contrastive Learning

The problem of pushing some points together (ℓ_{match}) and separating others (ℓ_{sep}), is equivalent to the problem that the pairwise contrastive loss, from the CL literature, optimizes [167]. This loss computes a cost that depends on positive and negative samples. Its objective is to reduce the distance between positive samples and separate negative samples beyond some margin value $m \in \mathbb{R}^+$.

In our problem, positive samples are defined as $y_t^{\mathcal{L}}$ and $y_t^{\mathcal{T}}$, and negative samples are defined as $y_{t-1}^{\mathcal{T}}$ and $y_t^{\mathcal{T}}$. The loss for positive samples is the same as ℓ_{match}. Differently, for negative samples, this method separates points by minimizing $\ell_{\text{sep}} = \max(0, m - d(y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}}))$, $\forall y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}} \in \mathcal{L}$. If their distance is smaller than m, $m - d(y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}}) > 0$, which is minimized until their distance is larger than m and $m - d(y_{t-1}^{\mathcal{T}}, y_t^{\mathcal{T}}) < 0$.

Commonly, the squared l^2-norm is used as the distance metric. Moreover, this loss is optimized along a trajectory starting at $t = 1$, which is a state sampled randomly from the task space $\mathcal{T}$. Then, we define a contrastive loss for motion stability as

$$\ell_{\text{stable}} = \sum_{b=0}^{B^s-1} \sum_{t=1}^{H^s} \underbrace{||y_{t,b}^{\mathcal{L}} - y_{t,b}^{\mathcal{T}}||_2^2}_{\ell_{\text{match}}} + \underbrace{\max(0, m - ||y_{t,b}^{\mathcal{T}} - y_{t-1,b}^{\mathcal{T}}||_2)^2}_{\ell_{\text{sep}}}, \tag{5.8}$$

where $B^s, H^s \in \mathbb{N}^+$ are the batch size corresponding to the number of samples used at each training iteration of the DNN and H^s is the trajectory length used for training, respectively.

Note that (5.8) does not take into account the fact that ℓ_{sep} should not be applied at y_g. However, in practice, it is very unlikely to sample x_g, so we do

not deem it necessary to explicitly consider this case. Furthermore, the loss ℓ_{match} enforces $f^{\mathcal{T}}(y_g) = 0$, which also helps to keep $y^{\mathcal{T}}_{t-1}$ and $y^{\mathcal{T}}_t$ together when $y^{\mathcal{T}}_{t-1} = y_g$.

Relaxing the problem

We can make use of the CL literature to use other losses to solve this problem. More specifically, we study the triplet loss [181] as an alternative to the pairwise loss. We call this version CONDOR (relaxed), since, in this case, the positive samples $y^{\mathcal{T}}_t$ and $y^{\mathcal{L}}_t$ are pushed closer, but it is not a requirement for them to be the same. Hence, we aim to observe if learning a specific structure in $\mathcal{L}$ is enough to enforce stability in $f^{\mathcal{T}}_\theta$, even though (5.8) is not solved exactly. This allows to compare different features between losses, such as generalization capabilities.

5.4.4. Boundaries of the Dynamical System

We enforce the stability of a motion in the region $\mathcal{T}$ by randomly sampling points from it and minimizing (5.8). Since this property is learned by a DNN, stability cannot be ensured in regions of the state space where this loss is not minimized, i.e., outside of $\mathcal{T}$. Therefore, it is crucial to ensure that if a point belongs to $\mathcal{T}$, its evolution will not leave $\mathcal{T}$. In other words, $\mathcal{T}$ must be a *positively invariant set* w.r.t. $f^{\mathcal{T}}_\theta$ [165, 182].

To address this, we design the dynamical system such that, by construction, is not allowed to leave $\mathcal{T}$. This can be easily achieved by projecting the transitions that leave $\mathcal{T}$ back to its boundary, i.e., if a point $x_t \in \mathcal{T}$ transitions to a point $x_{t+1} \notin \mathcal{T}$; then, it is projected to the boundary of $\mathcal{T}$. In this work, $\mathcal{T}$ is a hypercube; consequently, we apply an orthogonal projection by saturating/clipping the points that leave $\mathcal{T}$.

Note that this saturation is always applied, i.e., during the training and evaluation of the dynamical system. Hence, the stability conditions of Theorem 1 are imposed on a system that evolves in the positively invariant set $\mathcal{T}$.

5.4.5. Designing $f^{\mathcal{L}}$

So far, we presented a method for coupling two dynamical systems such that they share stability properties; however, we assumed that $f^{\mathcal{L}}$ existed. In reality, we must design this function such that it is stable by construction. Although several options are possible, in this work, we define $f^{\mathcal{L}}$ as

$$\dot{y}_t = \alpha \odot (y_g - y_t), \tag{5.9}$$

where $\alpha \in \mathbb{R}^n$ corresponds to the gains vector and $\odot$ to the element-wise/hadamard product. If $\alpha_i > 0$[3], this system monotonically converges to y_g [183], where α_i corresponds to the i-th element of α.

[3]This holds for the continuous-time case. However, we approximate the evolution of this system via the forward Euler integration method. Then, the system can be written for the discrete-time case as $y_{t+1} = Ay_t$, where $A = I + \text{diag}(-\alpha)\Delta t$, assuming $y_g = 0$ without loss of generality. To ensure stability, the absolute value of the eigenvalues of A must be less than one; then, $0 < \alpha_i < 2/\Delta t$.

Adaptive gains

In the simplest case, α is a fixed, pre-defined, value; however, the performance of the learned mappings ψ and ϕ is susceptible to the selected value of α. Alternatively, to provide more flexibility to the framework, we propose to define α as a trainable function that depends on the current latent state y_t, i.e., $\alpha(y_t)$. Then, the parameters of α can be optimized using the same losses employed to train ψ and ϕ, since it is connected to the rest of the network and the training error can be propagated through it.

Note that this system is stable under the same condition for α as before, as shown in Appendix B.2.

5.4.6. Behavioral Cloning of Dynamical Systems

Finally, we need to optimize an Imitation Learning loss ℓ_{IL} such that the learned dynamical system $f_\theta^{\mathcal{T}}$ follows the demonstrations of the desired motion. For simplicity, we opt to solve a Behavioral Cloning (BC) problem; however, in principle, any other IL approach can be used. As described in Sec. 5.3, this can be achieved by minimizing the (forward) Kullback-Leibler divergence between the demonstration's trajectory distribution and the trajectory distribution induced by the learned dynamical system. Note that this problem formulation is equivalent to applying Maximum Likelihood Estimation (MLE) between these distributions [40]; hence, we can rewrite it as

$$f_\theta^{\mathcal{T}*} = \underset{f_\theta^{\mathcal{T}} \in \mathcal{F}}{\arg\max}\, \mathbb{E}_{\tau \sim p^*(\tau)} \left[\ln p_\theta(\tau) \right]. \tag{5.10}$$

If we note that $p_\theta(\tau)$ is a product of conditional transition distributions $p_\theta(x_{t+1}|x_t)$, we can rewrite it as $p_\theta(\tau) = \Pi_{t=0}^{T-1} p_\theta(x_{t+1}|x_t) p(x_0)$, where $p(x_0)$ is the initial state probability distribution. Replacing this in (5.10) and ignoring constants we obtain

$$f_\theta^{\mathcal{T}*} = \underset{f_\theta^{\mathcal{T}} \in \mathcal{F}}{\arg\max}\, \mathbb{E}_{\substack{x_{t+1} \sim p^*(x_{t+1}|x_t), \\ x_t \sim p^{t*}(x_t)}} \left[\sum_{t=0}^{T-1} \ln p_\theta(x_{t+1}|x_t) \right], \tag{5.11}$$

where $p^{t*}(x_t)$ is the probability distribution of states at time step t, and $p^*(x_{t+1}|x_t)$ is the distribution of transitioning to state x_{t+1} given that the system is in some state x_t. Both of these distributions are induced by the dynamical system f^*.

In practice, however, we do not have an analytical representation of the distributions $p^{t*}(x_t)$ and $p^*(x_{t+1}|x_t)$. Therefore, the problem has to be estimated through empirical evaluations of this objective, which is achieved using the demonstrations present in the dataset $\mathcal{D}$. Then, we can solve this problem iteratively [139] by randomly sampling batches of B^i trajectories from $\mathcal{D}$ at each iteration and maximizing

$$f_\theta^{\mathcal{T}*} = \underset{f_\theta^{\mathcal{T}} \in \mathcal{F}}{\arg\max} \sum_{b=0}^{B^i-1} \sum_{t=0}^{T-1} \ln p_\theta(x^*_{t+1,b}|x_{t,b}), \tag{5.12}$$

where the subscript b has been added to the states indicating their correspondence to the different trajectories of B.

To solve this problem, we can assume the transition distribution of the learning system to be a Gaussian with fixed covariance, and a mean corresponding to the *forward Euler integration* [184] of $f_\theta^{\mathcal{T}}$ for the given state $x_{t,b}$, i.e., $x_{t+1,b} = x_{t,b} + f_\theta^{\mathcal{T}}(x_{t,b})\Delta t$, where Δt corresponds to the time step size. Furthermore, the same Gaussian assumption is made for the demonstration's distribution $p^*(x_{t+1}|x_t)$; however, since its transitions are obtained directly from the demonstrations, it is not necessary to integrate in this case. Then, (5.12) reduces to the Mean Squared Error (MSE) minimization between the mean of the demonstration's distribution $p^*(x_{t+1}|x_t)$, and the mean of the learning distribution $p_\theta(x_{t+1}|x_t)$ [2], i.e.,

$$f_\theta^{\mathcal{T}*} = \underset{f_\theta^{\mathcal{T}} \in \mathcal{F}}{\arg\min} \sum_{b=0}^{B^i-1} \sum_{t=0}^{T-1} ||x^*_{t+1,b} - \left(x_{t,b} + f_\theta^{\mathcal{T}}(x_{t,b})\Delta t\right)||_2^2. \tag{5.13}$$

In practice, however, if the trajectories of the demonstrations are too long, due to computation or complexity limitations, it might not be convenient to optimize this objective for the complete trajectories. Therefore, this problem can be simplified by allowing the initial conditions of the demonstration batches to be at any time step $t' \in \{0, ..., T-1\}$, and optimizing the problem for some time horizon $H^i \leq T$. Consequently, we get the loss ℓ_{IL} that we employ to solve the BC problem in this work:

$$f_\theta^{\mathcal{T}*} = \underset{f_\theta^{\mathcal{T}} \in \mathcal{F}}{\arg\min} \underbrace{\sum_{b=0}^{B^i-1} \sum_{t=t'}^{H^i-1} ||x^*_{t+1,b} - x_{t,b} - f_\theta^{\mathcal{T}}(x_{t,b})\Delta t||_2^2}_{\ell_{\text{IL}}}. \tag{5.14}$$

5.4.7. COMPOUNDING ERRORS AND MULTI-STEP LEARNING

Commonly, (5.14) is solved as a single-step prediction problem (i.e., $H^i = 1$) by computing only one transition from $x_{t',b}$ using $f_\theta^{\mathcal{T}}$ and comparing it against $x^*_{t'+1,b}$. Nevertheless, in practice, the learned dynamical system is applied recursively, i.e., assuming perfect tracking, every prediction is computed as a function of a previously computed output using the following equation:

$$x_h = x_{h-1} + f_\theta^{\mathcal{T}}(...x_1 + f_\theta^{\mathcal{T}}(x_0 + f_\theta^{\mathcal{T}}(x_0)\Delta t)\Delta t...)\Delta t, \tag{5.15}$$

where h is the evolution horizon. Therefore, the prediction error of $f_\theta^{\mathcal{T}}$ compounds and grows multiplicatively by every new prediction [185, 186]. This makes the dynamical system diverge away from the states present in the demonstration's trajectories, requiring the system to make predictions in states that are not supported by the training data, which is known as the *covariate shift* problem [2]. Consequently, the prediction error grows even larger.

An important reason for this issue to occur is that the learned system is expected to act over multiple steps when it is only being trained for predicting single steps.

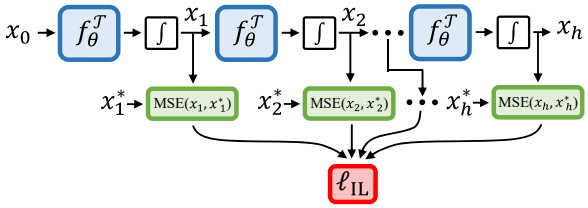


Figure 5.4.: Multi-step IL loss for one sample when using backpropagation through time, where $h = H^i$.

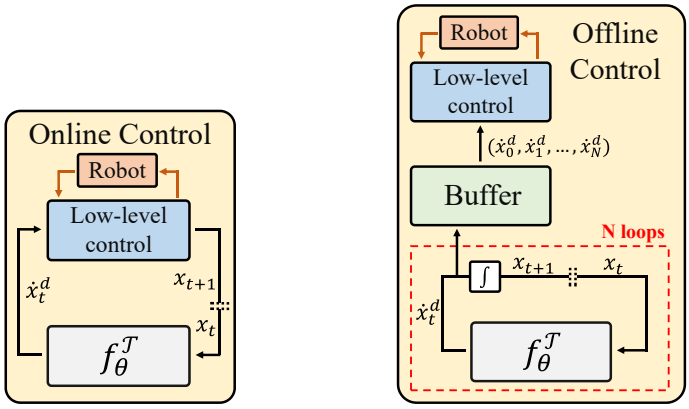
To alleviate this problem, the dynamical system must be trained for predicting multiple steps, by setting $H^i > 1$ and computing $x_{t,b}$ in (5.14) recursively, as shown in Fig. 5.4. In practice, however, the single-step loss is commonly employed, as the multi-step loss has been regarded as being challenging to optimize, even for short prediction horizons [187]. Nevertheless, these challenges can be addressed with current DNN optimization techniques.

Consequently, we optimize the multi-step loss by noting that this can be achieved using *backpropagation through time* [188, 189], which has become popular and improved given its use in Recurrent Neural Networks (RNNs) [190]. Hence, its limitations such as *exploding/vanishing gradients* or *ill-conditioning* [191] have been alleviated. Furthermore, specifically for this case, we can observe that every forward integration step in (5.15) can be interpreted as one group of layers inside a larger DNN that computes x_h. Then, each one of these groups has the same structure as the *residual blocks* in *ResNet* [192], which have also shown to be beneficial for alleviating vanishing/exploding gradients issues [193].

5.5. Simulated Experiments

In this section, we employ datasets of human handwriting motions to validate our method. Although these datasets contain human demonstrations, our evaluation of the learned motions is *simulated*, since no real system is involved in this process. This can be better understood with Fig. 5.5, where we show two different control strategies that can be employed with CONDOR. More specifically, Fig. 5.5b presents an offline control strategy where a trajectory is computed and stored in a buffer by applying CONDOR recursively. Afterwards, this trajectory is tracked by a low-level controller. In our evaluation, however, we ignore the low-level controller part and evaluate CONDOR using only the trajectory provided by the buffer, i.e., we assume that the trajectory is tracked perfectly. Despite this assumption, this methodology with this dataset has been extensively used in the literature, since it allows to test if the learning method generates adequate state transition requests [158, 159, 163].

The DNN architecture and hyperparameter optimization process of the models used in this section are described in appendices B.3 and B.4, respectively.



(a) Online control. At every time step, $f_\theta^{\mathcal{T}}$ receives x_t from the sensors of the robot, and its output is fed to a low-level controller that tracks it.

(b) Offline control. $f_\theta^{\mathcal{T}}$ is applied recursively N times to create, offline, a trajectory that is stored in a buffer. Afterward, the complete trajectory is tracked with a low-level controller.

Figure 5.5.: Control strategies that can be used with CONDOR.

5.5.1. LASA dataset validation: first-order 2-dimensional motions

We validate our method using the LASA dataset[4], which comprises 30 human handwriting motions. Each motion, captured with a tablet PC, includes 7 demonstrations of a desired trajectory from different initial positions. The state is represented as 2-dimensional positions, and the learned systems are of first order, i.e., the output of $f_\theta^{\mathcal{T}}$ is a desired velocity. Although the demonstrations may have local intersections due to human inaccuracies, the shapes contained in this dataset can be well represented using first-order dynamical systems, which cannot represent intersections. Consequently, we employ the LASA dataset to evaluate motions modeled as first-order dynamical systems, which is the same approach that was taken by the paper that introduced this dataset [158].

Fig. 5.6 shows three examples of dynamical systems learned with CONDOR. These motions share three features:

1. **Adequate generalization:** motions generated in regions with no demonstrations smoothly generalize the behavior presented in the demonstrations.

2. **Accuracy:** the learned models accurately reproduce the demonstrations.

3. **Stability:** the vector fields suggest that, independently of the initial conditions, every motion reaches the goal.

[4] https://cs.stanford.edu/people/khansari/download.html

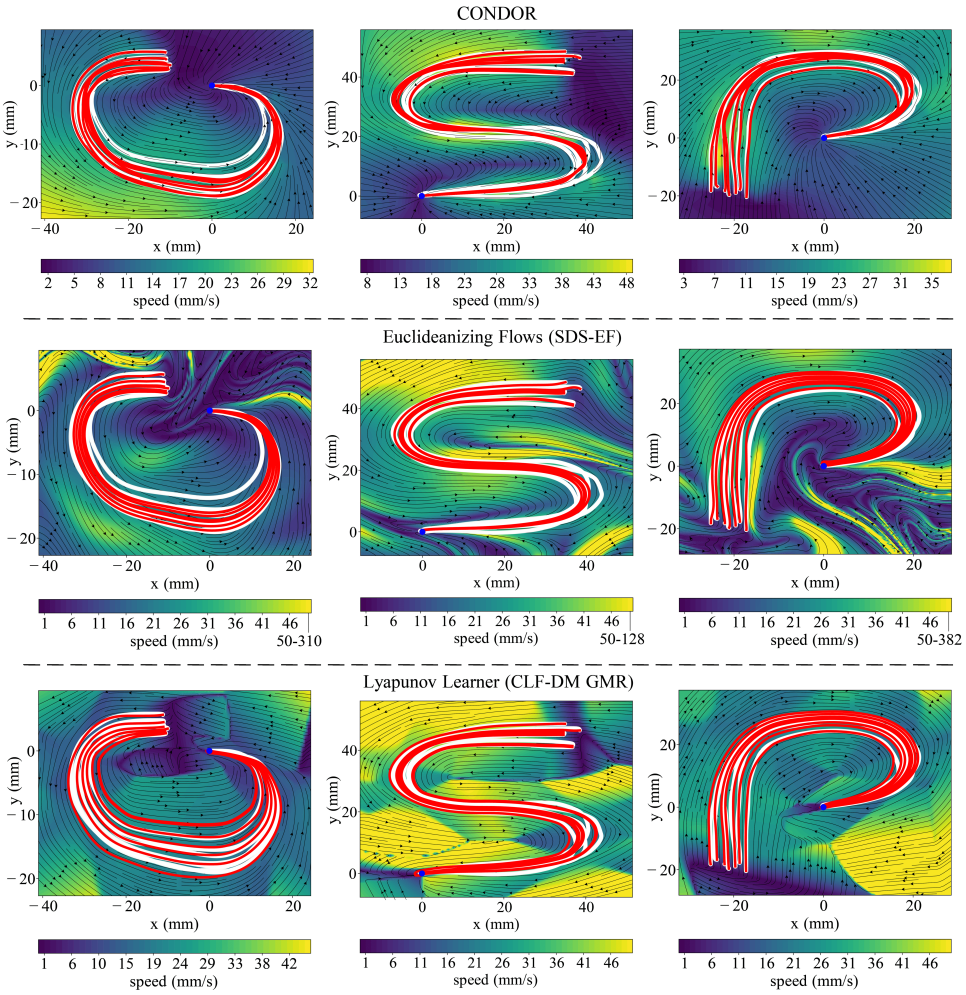


Figure 5.6.: Examples of LASA dataset motions learned using CONDOR, SDS-ES, and CLF-DS (GMR). White curves represent the demonstrations. Red curves represent the executed motions by the learned model when starting from the same initial positions as the demonstrations. The arrows indicate the vector field of the learned dynamical system (velocity outputs for every position). In SDS-ES, every speed greater than 50 mm/s is saturated to this value.

In the following subsections, we provide further details regarding each one of these points. Moreover, we compare CONDOR[5] with two other state-of-the-art methods for stable motion generation: 1) Control Lyapunov Function-based

[5]CONDOR code repository: https://github.com/rperezdattari/Stable-Motion-Primitives-via-Imitation-and-Contrastive-Learning

Dynamic Movements (CLF-DM) using Gaussian Mixture Regression (GMR)[6] [164], and 2) Stable Dynamical System learning using Euclideanizing Flows (SDS-EF)[7] [159]. CLF-DM (GMR) learns a dynamical system using a GMR and corrects its behavior whenever it is not stable according to a learned Lyapunov function. SDS-EF is a diffeomorphism shaping method, as introduced in Sec. 5.4.2.

Generalization

Fig. 5.6 also depicts the performance of CLF-DM and SDS-EF on three motions. Here, we observe that even though the stability of these methods is guaranteed, unlike CONDOR, the behavior that they present in regions without demonstrations might not always be desired.

In the case of SDS-EF, unpredictable motions can be generated[8] (e.g., bottom image, bottom-right quadrant), which, furthermore, can reach very high speeds (e.g., 382 mm/s, while the demonstrations exhibit maximum speeds of around 40 mm/s). Note, however, that this issue can be alleviated by optimizing SDS-EF for a shorter period of time, but this also makes it less accurate. Such unpredictability and high speeds can be a limitation in real-world scenarios. For instance, when humans interact with robots and must feel safe around them, or due to practical limitations, e.g., it is unfeasible to track the requested motions with a low-level controller.

Differently, in the case of CLF-DM, nonsmooth transitions are present in some regions of the state space due to the corrections applied by the Lyapunov function. This can also be a limitation, since robotic systems commonly avoid nonsmooth trajectories to minimize the risk of damage [5].

Lastly, Fig. 5.6 evidences that, in real-world scenarios, CLF-DM and SDS-EF are susceptible to making robots leave their workspaces. These methods do not constrain their trajectories to reside inside a specific space, they only guarantee that, eventually, these will converge to the goal. In practice, however, the learned trajectories might need to leave a robot's workspace to reach the goal. Then, in Fig. 5.6, if we assume that the observed regions are a robot's workspace, the vector fields of CLF-DM and SDS-EF indicate that some motions depart from it. In contrast, in CONDOR the workspace is a positively invariant set w.r.t. the learned dynamical system (see Sec. 5.4.4); consequently, motions stay inside it.

Accuracy

Fig. 5.6 indicates that every method is able to accurately reproduce the demonstrations. However, CLF-DM is clearly less accurate than CONDOR and SDS-EF. For instance, in the bottom-left image, the inner red trajectory drifts

[6] CLF-DM code repository: https://github.com/rperezdattari/Learning-Stable-Motions-with-Lyapunov-Functions

[7] SDS-EF code repository: https://github.com/mrana6/euclideanizing_flows

[8] These results are not completely consistent with the ones reported in [159], since we removed additional preprocessing (smoothing and subsampling) to compare every method under the same conditions.

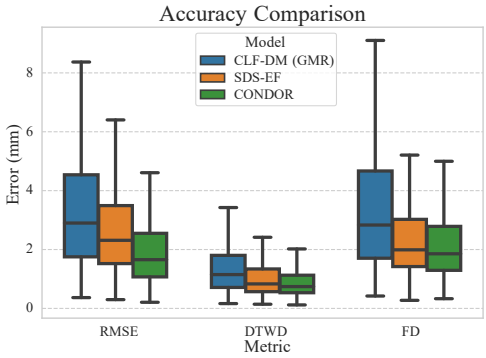


Figure 5.7.: Accuracy comparison of CONDOR against state-of-the-art methods. Each box plot summarizes performance over the 30 motions of the LASA dataset.

away from the demonstrations, coming back to them at the end of the motion due to its stability properties.

Quantitatively speaking, we can employ different metrics to evaluate the accuracy of the learned trajectories (see Fig. 5.7). Commonly, a distance between two trajectories is minimized; one trajectory corresponds to a demonstration, and the other corresponds to the one generated by the learned dynamical system when starting from the same initial condition as the demonstration. However, different distances between trajectories can be computed depending on the features that we aim to evaluate from the trajectories. To have a more complete view of the accuracy performance of our method, we compare CONDOR, CLF-DM (GMR)[9] and SDS-EF[10] under three distance metrics: 1) Root Mean Squared Error (RMSE), 2) Dynamic Time Warping Distance (DTWD) [194], and 3) Frechet Distance (FD) [195].

We can observe that CONDOR clearly achieves better results against CLF-DM (GMR) under every metric, while a smaller gap, yet superior, is achieved against SDS-EF.

Stability

Lastly, we quantitatively study the stability properties of CONDOR. As mentioned in Sec. 5.4, the stability of the motions it learns depends on the optimization problem being properly minimized. Therefore, we need to empirically test this after the optimization process finishes.

To achieve this, we integrate the dynamical system for L time steps, starting from P initial states, and check if the system converges to the goal (i.e., fixed-point iteration, where the fixed point corresponds to the goal). The larger the P, the more accurate the results we obtain. If L is large enough, the system should

[9] Each GMR consisted of 10 Gaussians and each Lyapunov function was estimated using 3 asymmetric quadratic functions.
[10] Results were extracted from [159].

converge to the goal after L steps. Hence, by computing the distance between the last visited state and the goal, and checking that it is below some predefined threshold ϵ, it is possible to evaluate if a trajectory is successful or not (i.e., if it converges to the goal).

We evaluated CONDOR using all of the motions present in the LASA dataset with $L = 2000$ and $P = 1225$ with $\epsilon = 1$mm, and observed that 100% of the trajectories reached the goal. Hence, CONDOR is able to successfully learn stable motions.

5.5.2. Ablation Study

To better understand CONDOR and the relevance of its different parts, we perform an ablation study where we compare four variations of this method:

1. **CONDOR:** the base method studied in Sec. 5.5.1.
2. **CONDOR (relaxed):** the contrastive loss for stability is replaced with the triplet loss. This variation is presented to observe the importance of minimizing the exact loss presented in (5.8) or whether it is enough to enforce this type of structure in the latent space of the NN to obtain stable motions.
3. **CONDOR (fixed gains):** as explained in Sec. 5.4.5, having adaptive gains in the latent dynamical system described in (5.9) should help obtaining more flexible motions. Therefore, this model, with fixed gains, is studied to observe the relevance and effect of using adaptive gains.
4. **Behavioral Cloning (BC):** the stability loss is removed and only the BC loss is employed to learn motions. This model is used to study the effect that the stability loss can have on the accuracy of the learned motions. To observe the behavior of BC, we refer the reader to Fig. 5.2.

Fig. 5.8 presents examples of motions learned with both CONDOR (relaxed) and CONDOR (fixed gains). In both cases, the motions display accuracy and stability. However, these models differ in their generalization. CONDOR (relaxed) has a generalization behavior similar to the one of CONDOR shown in Fig. 5.6. In these cases, the regions of the state space without demonstrations exhibit a trend that resembles the one observed in the demonstrations. In contrast, the generalization of CONDOR (fixed gains) does not follow this trend as closely. For example, within certain regions, the velocity of the motions decreases, and as they approach the demonstrations, their direction becomes nearly orthogonal, indicating a discrepancy between the generalized behavior and the pattern presented in the demonstrations.

Accuracy

In this subsection, we compare the accuracy of the different variations of CONDOR (see Fig. 5.9). Intuitively, BC should perform better than any variation of CONDOR, since it only optimizes the BC loss. In contrast, the other variations

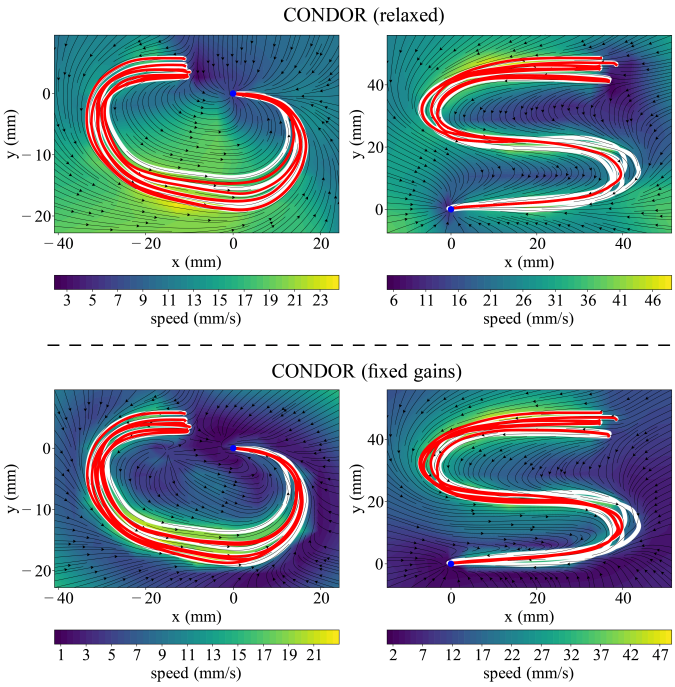


Figure 5.8.: Examples of LASA dataset motions learned using two variations of CONDOR: 1) relaxed and 2) fixed gains.

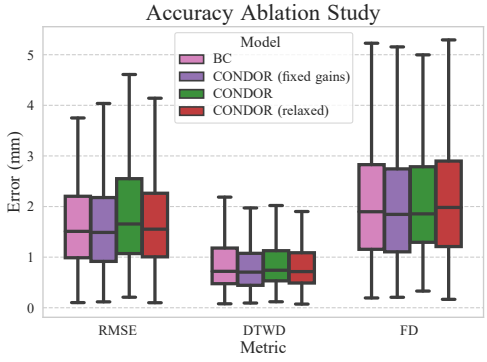


Figure 5.9.: Accuracy comparison of different variations of CONDOR. Each box plot summarizes performance over the 30 motions of the LASA dataset.

also optimize the stability loss, which could harm/limit the minimization of the BC loss. Consequently, BC is the lower bound for the accuracy performance, i.e., best case scenario, a variation of CONDOR performs as well as BC does.

In Fig. 5.9, we observe that the accuracy performance of all of the variations

Table 5.1.: Percentage of unsuccessful trajectories over the LASA dataset ($L = 2000$, $P = 1225$, $\epsilon = 1$mm).

Behavioral Cloning	CONDOR (fixed gains)	CONDOR (relaxed)	CONDOR
36.4653%	0.0054%	**0.0000**%	**0.0000**%

of CONDOR is very similar, including the BC case. This result shows that CONDOR, and its variations, is able to effectively minimize the BC and stability loss together without harming the accuracy performance of the learned motions.

Stability

Table 5.1 shows the results of stability tests on the presented methods. We can observe that when stability is not enforced (i.e., Behavioral Cloning), the percentage of unsuccessful trajectories is significant, being larger than one-third of the total amount of trajectories. In contrast, when stability is enforced using adaptive gains, the system achieves perfect performance (i.e., every trajectory reaches the goal), as it is observed with the results of CONDOR and CONDOR (relaxed). Interestingly, this result also shows that the relaxed variation of CONDOR can be employed for achieving stable motions without having a loss in performance. In contrast, when fixed gains are employed, although the percentage of unsuccessful trajectories is very low ($< 0.01\%$), the performance degrades. This result suggests that the stability loss is not being as effectively minimized as when the gains are adaptive.

Dynamical Systems Mismatch

So far, we observed that every variation of CONDOR that minimizes the stabilization loss is able to learn accurate and stable motions. The case of CONDOR (fixed gains) showed a slightly worse stability performance and poorer generalization capabilities than CONDOR and CONDOR (relaxed). However, CONDOR has not shown to be clearly superior to its variations, especially in the CONDOR (relaxed) case.

Since CONDOR (relaxed) approximates the loss that minimizes the distance between $y_t^{\mathcal{L}}$ and $y_t^{\mathcal{T}}$, the trajectories that it obtains with $f^{\mathcal{L}}$ and $f_\theta^{\mathcal{T}}$ in the latent space should diverge faster than the ones generated with CONDOR. To investigate this idea, we evaluate the optimization of this loss by separately simulating $f_\theta^{\mathcal{T}}$ and $f^{\mathcal{L}}$ when starting from the same initial conditions. If the stabilization loss is perfectly minimized, these simulations should yield the same trajectories when mapping the evolution of $f^{\mathcal{L}}$ to task space[11]; otherwise, they should diverge from each other.

Fig. 5.10 presents motions learned using the different variations of CONDOR and shows motions generated in task space when following $f_\theta^{\mathcal{T}}$ and $f^{\mathcal{L}}$. We

[11] Trajectories from $f^{\mathcal{L}}$ with known initial conditions in $\mathcal{T}$ (hence, $y_0^{\mathcal{L}} = \psi_\theta(x_0)$), can be mapped to $\mathcal{T}$ by recursively applying $x_{t+1} = x_t + \phi_\theta(y_t^{\mathcal{L}})\Delta t$.

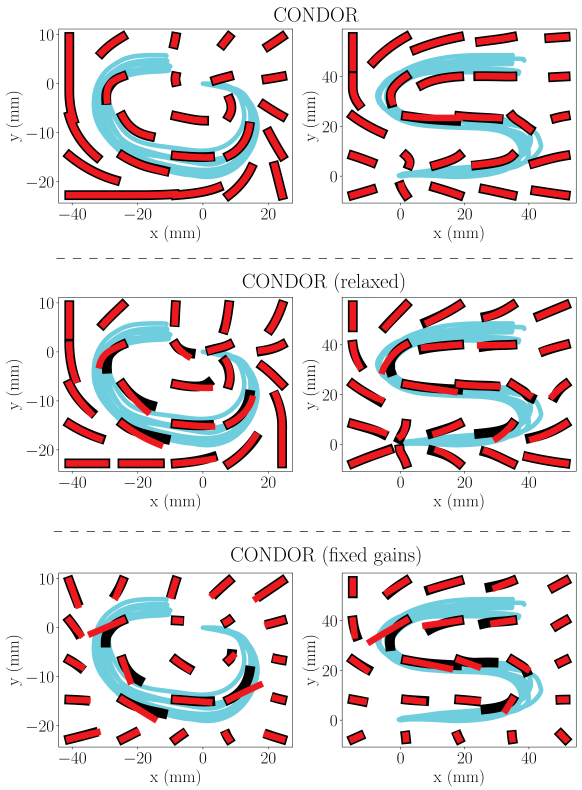


Figure 5.10.: Dynamical systems mismatch comparison. Blue curves represent the demonstrations, black curves represent trajectories generated using $f_\theta^{\mathcal{T}}$ of the dynamical system, and red curves represent trajectories generated using $f^{\mathcal{L}}$ of the dynamical system. The black and red trajectories were obtained by integrating the dynamical system through 80 time steps.

can observe that CONDOR performs well in the complete state space, where, for most trajectories, it is not possible to detect a difference between the results obtained using $f^{\mathcal{L}}$ and $f_\theta^{\mathcal{T}}$. In contrast, we can observe that, for the other cases, trajectories diverge more pronouncedly. Interestingly, the divergent trajectories seem to overlap with the regions where demonstrations are provided. This suggests the stabilization loss is not properly minimized in this region, indicating that these variations of CONDOR struggle to find good solutions in the regions of the state space where the imitation and stabilization losses are optimized together, i.e., in the demonstrations. Finally, it is also possible to observe that CONDOR (relaxed) obtains trajectories that are slightly more similar to the ones obtained with CONDOR (fixed gains), although it is not conclusive.

Quantitatively, we can analyze this trajectory difference by computing the accumulated error between the trajectories generated using both dynamical

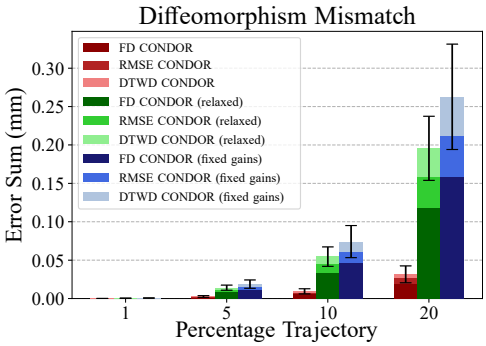


Figure 5.11.: Trajectory mismatch error. Results are presented as a function of the length of a trajectory with respect to the complete length of the demonstrated trajectories, corresponding to 1000 transitions. 100 trajectories in the task and latent spaces were simulated, whose initial positions were uniformly distributed in the motion's state space. The results show the mean and half of the standard deviation of the error computed with these simulations.

systems. Fig. 5.11 shows this error as a function of the trajectory length. As expected, this error grows for the dynamical systems as a function of their length. However, CONDOR obtains a significantly lower error than its variations. Furthermore, Fig. 5.11 clearly shows that CONDOR (relaxed) outperforms CONDOR (fixed gains). Finally, since this accumulated error is a consequence of how well the stability loss is minimized, these results might explain why the stability performance of CONDOR (fixed gains) is not perfect, i.e., this variation is not able to successfully minimize the stability loss in the complete state space.

As a final remark, we can note that generating trajectories in the latent space and then mapping them to task space can have other applications, such as predicting future states efficiently and employing them, for instance, in Model Predictive Control frameworks. It is considerably faster to generate trajectories in the latent space than using the complete DNN architecture to compute them in task space, since the number of parameters and layers required to do so is smaller. Then, once the trajectory is generated in the latent space, it can be mapped to task space as one batch in one forward pass.

5.5.3. LAIR dataset validation: second-order 2-dimensional motions

In this section, we introduce the *LAIR handwriting dataset*. The objective is to test the accuracy and stability performance of the proposed method for second-order motions, where the state comprises both position and velocity. This dataset contains 10 human handwriting motions collected using a mouse interface on a PC. The state here is 4-dimensional, encompassing a 2-dimensional position and velocity, and the output of $f_\theta^{\mathcal{T}}$ is the desired acceleration. The dataset's

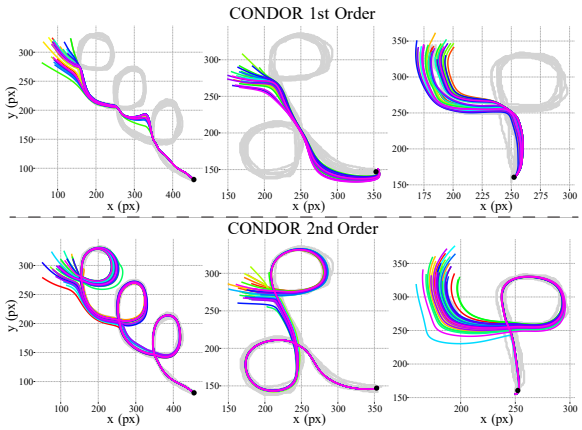


Figure 5.12.: Motions modeled using CONDOR with first order and second order systems. The shapes in grey correspond to the demonstrations. The colored curves correspond to different instances of trajectories generated when starting from different initial states. Every trajectory was initialized with zero velocity and the initial positions were obtained by sampling from a Gaussian distribution around the initial positions observed in the demonstrations. 36 trajectories were sampled per plot.

shapes present several position intersections that have been designed to require, at least, second-order systems to model them. This dataset is employed to test the scalability of the proposed method in terms of the order of the motion.

Unlike the LASA dataset, the LAIR dataset contains raw demonstrations without any type of postprocessing. Hence, the ending points of the demonstrated trajectories might not always coincide exactly. To account for this, the goal of a motion is computed by taking the mean between these ending points.

Accuracy

Fig. 5.12 shows three examples of motions of the LAIR dataset. These motions can only be modeled using a dynamical system of, at least, second order. First-order systems only employ position information to generate a trajectory; hence, visiting the same position two times will generate an ambiguity for the learning algorithm. This makes the learned system collapse to a solution that lies in between the multiple demonstrated options. Therefore, we observe that first-order systems with CONDOR are not able to appropriately model the shown motions.

In contrast, we observe that second-order systems are able to appropriately capture the dynamics of the demonstrated motions and execute them as they were intended. However, some trajectories (especially those coming from the tail of the initial-state sampling distribution) do not go through the first intersection, since they start from a position that, given its distance from the initial states of the demonstrations, directly follows the trend of the motion after this intersection. If

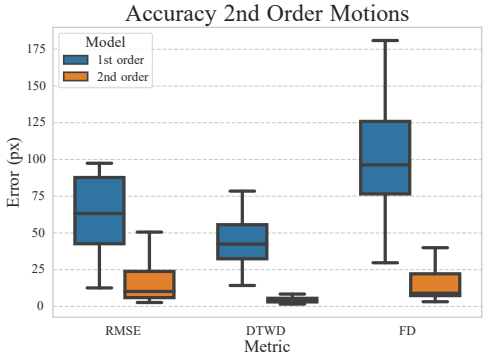


Figure 5.13.: Accuracy comparison of CONDOR when modeling motions using first order and second order systems. Each box plot summarizes performance over the 10 motions of the LAIR dataset.

Table 5.2.: Percentage of successful trajectories over the LAIR dataset ($L = 2000$, $P = 1225$, $\epsilon = 10$px).

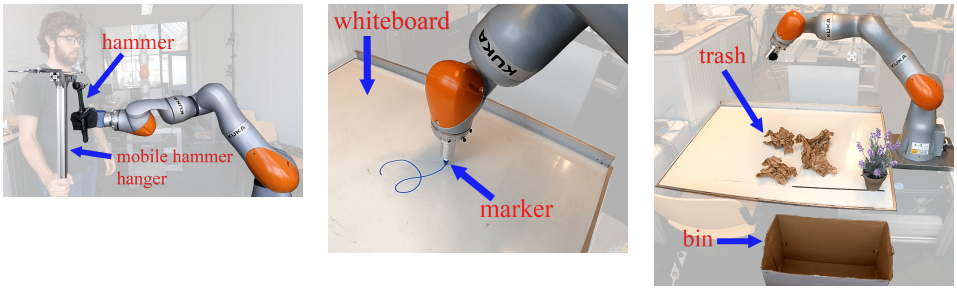
CONDOR (1st order)	CONDOR (2nd order)
9.3388%	**0.0000**%

this is a limitation for a specific application, providing demonstrations in those regions would make the system behave as expected. Finally, another interesting feature of these motions, is that the different trajectories, eventually, seem to collapse to the same position, overlapping with each other. This comes as an artifact when incorporating the stability loss, where the systems find these solutions to ensure stability.

Quantitatively, the same conclusions can be drawn when observing Fig. 5.13. This figure presents the results of the accuracy of both CONDOR variations under the same metrics employed in Sec. 5.5.1. As expected, the second-order systems outperform the first-order systems by a large margin.

Stability

Finally, we study the stability of the motions generated with CONDOR over the LAIR dataset when using first-order and second-order systems. Table 5.2 shows that when using first-order systems, CONDOR struggles to generate stable motions with second-order demonstrations. For instance, when a demonstration has a *loop*, the optimization of the DNN might not find a proper solution, since trajectories inside the loop do not have a way of reaching a region outside the loop without ignoring the demonstrations. In contrast, CONDOR with second-order systems is always able to learn stable motions.



(a) Hammer hanging. **(b)** Writing number two. **(c)** Table cleaning.

Figure 5.14.: Setup of real-world experiments.

5.6. Real-World Experiments

To validate the proposed framework in more realistic scenarios, we design three real-world experiments using a 7-DoF KUKA iiwa manipulator: 1) hammer hanging, 2) writing the number two, and 3) cleaning a table (see Fig. 5.14). Throughout these experiments, four important characteristics of the learning problem are changed: 1) dimensionality of the motion, 2) order of the motion, 3) control strategy, and 4) data collection method. These characteristics define different Imitation Learning scenarios that can be found in real-world robotic problems. Hence, by testing CONDOR in these scenarios we aim to show the applicability, flexibility, and robustness of our method. Furthermore, if we compare these scenarios with the simulated ones studied in the previous sections, we can observe that our method is not restricted to 2-dimensional motions only and that it can also work in higher-dimensional problems. Table 5.3 shows a summary of the real-world experiments, which are explained in detail in the following subsections.

Similarly to the LAIR dataset, the demonstrations are not postprocessed in these experiments. Hence, in this section, the goal of the motions is also computed by taking the mean between the ending points of each demonstration.

5.6.1. Hammer hanging: First-order 3D motions

This experiment consists of learning to control the end-effector's position of a robot such that it hangs a hammer (see Fig. 5.14a), allowing us to test the behavior of CONDOR for first-order 3-dimensional motions. This problem is interesting since it shows that implicit knowledge, that otherwise requires modeling, can be transferred to the robot via human demonstrations. In this case, this knowledge includes information about the geometry of the hammer and the hanger that is required to hang the hammer.

Control

We employ the online control strategy depicted in Fig. 5.5a, which allows the robot to be reactive to perturbations and adjust its motion *on the fly* if the environment

Table 5.3.: Characteristics of the real-world experiments.

Task	Dim state	Order motion	Control	Data collection	# demos
Hammer hanging	3	1	online, end eff.	motion capture	10
Writing two	4	2	offline, end eff.	computer mouse interface	6
Table cleaning	6	1	online, joints	kinesthetic teaching	2*

* Two motion models were learned, and one demonstration was used per model.

changes. Hence, at every time step, the robot obtains its position with respect to the goal (i.e., the hanger) and sends an end effector's velocity request to a low-level controller. For details regarding the low-level controller, the reader is referred to Appendix B.5.2.

Demonstrations

We used a motion capture system to collect demonstrations. The demonstrator had to wear a glove whose position was tracked by the tracking system. We recorded 10 demonstrations and used them to train CONDOR.

This approach has the advantage of it being comfortable for the human, since it does not require the human to adjust to any specific interface nor interact with the robot, which can require training. Nevertheless, since the robot embodiment is not being employed to collect the demonstrations, there is no guarantee that the collected motions will feasible for the robot to execute. Therefore, it is necessary to record motions that can be executed by the robot, which, depending on the problem, might require knowledge about the robotic platform.

Moving goal

To test the reactive capabilities of this approach, and the generalization properties of motions modeled as dynamical systems, we made this problem more challenging by making the hanger movable. To achieve this, we added tracking markers to the top of the hanger and fed the hanger's position to CONDOR in real time. Consequently, while the robot was executing the hanging motion, the hanger could be displaced and the robot had to react to these changes in the environment.

Notably, no extra data is required to achieve this, since the motion of CONDOR is computed as a function of the relative position of the robot w.r.t. the goal. Hence, by displacing the goal, the position of the end effector with respect to the hanger changes, making CONDOR provide a velocity request according to this new position.

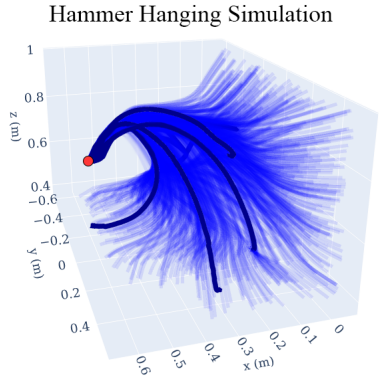


Figure 5.15.: The blue trajectories represent the learned model's evolution of the robot end effector's position when starting from different initial conditions. The larger and darker trajectories correspond to demonstrations. Some demonstrations are occluded and others were removed for visualization purposes. The red point corresponds to the goal.

Results

Fig. 5.15 shows a 3D plot with 1250 simulated trajectories generated with CONDOR when starting from different initial positions. We can observe that all of the trajectories reach the goal while following the shape in the demonstrations. The performance of this model on the real robot can be observed in the attached video.

5.6.2. Writing: Second-order 2D motions

We also tested CONDOR in a writing scenario (see Fig. 5.14b). The objective is to control the robot's end effector to write the number two on a whiteboard. To write the number two, it is necessary to use second-order motions, since this character has one intersection. Therefore, in this experiment, we aim to validate the ability of CONDOR for modeling second-order motions. Finally, note that for writing it is only necessary for the robot to move in a 2-dimensional plane; however, since the motion is of second order, the state space of the robot is 4-dimensional (the same as the motions in the LAIR dataset).

Control

We employ the offline control strategy as depicted in Fig. 5.5b. This approach is suitable for writing since in this task it is important that the trajectory that the robot executes is consistent with the one that CONDOR predicts from the initial state. For instance, if the robot, while executing the motion is perturbed by its interaction with the whiteboard in some direction, it would transition to a state that is not consistent anymore with the character that has been written so far.

In an offline control approach this is not very critical, because, given that the reference of the motion is pre-computed, it would make the robot move back to a state that is consistent with the motion that is being written. For details regarding the low-level controller, the reader is referred to Appendix B.5.3.

Demonstrations

The same PC mouse interface developed to collect the LAIR dataset is employed here. 6 demonstrations were collected and used to train CONDOR.

Results

The simulated results of this experiment follow the same behavior as the ones presented in Sec. 5.5.3. To observe its behavior on the real robot, the reader is referred to the attached video.

5.6.3. Table cleaning: First-order 6D motions

Finally, we test CONDOR in a cleaning task. The objective is to use the robot's arm to push garbage, which is on top of a table, to a trash bin. Differently from the other scenarios, in this case, the robot's joint space is directly controlled with CONDOR. Hence, we learn a 6-dimensional motion. Note that the robot has 7 degrees of freedom, but we keep the last joint fixed as it has no influence on the task.

Since the motions learned by CONDOR can be used as primitives of a more complex motion, in this experiment we highlight this capability by learning two motions that are sequenced together to generate the complete cleaning behavior. Each motion is trained with only one demonstration.

This scenario allows us to test two features of our method: 1) its behavior in a higher-dimensional space (6D), and 2) its capability to learn motions from only one demonstration.

Control

Similarly to the hanging hammer experiment, we use the online control strategy (Fig. 5.5a). Differently than before, in this case, the joint space of the robot is directly controlled with CONDOR, i.e., a reference velocity for the joints is provided to the low-level controller. Joint-space control is suitable for this task because the configuration of the robot is important for completing the task successfully since its body is used to push the trash. For details regarding the low-level controller, the reader is referred to Appendix B.5.1.

Demonstrations

For this experiment, kinesthetic teaching was used to collect demonstrations. This approach consists of collecting demonstrations by physically interacting with the robot and guiding it along the desired trajectory. To make this task easier, the gravitational forces of the robot were compensated such that it would not move unless the human interacted with it.

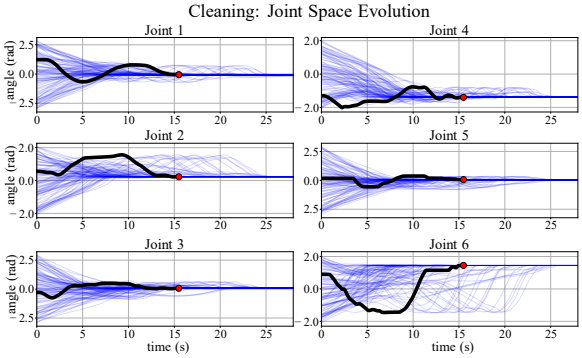


Figure 5.16.: Simulated trajectories, as a function of time, of the second motion of the cleaning task. Blue trajectories correspond to evaluations of the model under different initial conditions and the black trajectory corresponds to the demonstration. The red point is the goal.

Results

Fig. 5.16 presents simulated results of the second cleaning primitive learned in this experiment. Since the motion is 6-dimensional and, hence, it is very challenging to visualize in one plot, we use six different plots to separately show the evolution of each state dimension as a function of time.

In this case, we simulate 100 trajectories using CONDOR, since more make the plots difficult to analyze. From them, we observe that as time increases, every trajectory eventually reaches the goal. Note that, given that their initial states are random, they can start further away or closer to the goal than the demonstration; therefore, it might take them a longer/shorter time to reach the goal. Lastly, we can observe that the demonstrated trajectory and some simulations, either overlap or have the same shape with a phase shift, which showcases that the demonstrated behavior is captured by CONDOR.

The reader is referred to the attached video to observe the behavior of the cleaning primitives on the real robot.

5.7. Extending CONDOR

One of the advantages of our proposed framework is that we can extend it to address more complex problems. Therefore, there are interesting areas of research that can be studied with CONDOR. In this section, we aim to show the steps that we have taken in this direction, which we plan to study deeper in future work. More specifically, we tested two extensions:

1. **Obstacle avoidance:** multiple obstacle avoidance methods have been proposed for motions modeled as dynamical systems, and it is an active field of research [196–199]. We test CONDOR with one of these extensions [196] and observe that it works properly. However, apart from this validation, we

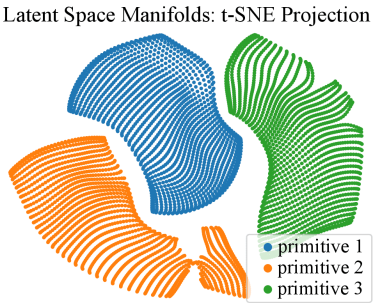


Figure 5.17.: t-SNE projection of the motion manifolds present in the latent space of the DNN.

do not provide a contribution to this problem. Hence, the reader is referred to Appendix B.6 for more details regarding obstacle avoidance in this research.

2. **Multi-motion learning and interpolating:** another interesting field of research is learning multiple motions together in one Neural Network model. This allows interpolating between these motions, generating novel behaviors that are not present in the demonstrations. This can, for instance, reduce the number of human demonstrations required to learn and generalize a problem to a different situation.

In the next subsection, we study the interpolation capabilities of motions learned with CONDOR.

5.7.1. Multi-motion Learning and Interpolation

We aim to provide preliminary results regarding the multi-motion learning capabilities of CONDOR, and its behavior in terms of interpolation and stability. To learn multiple motions in one Neural Network we extend its input with a one-hot code that indicates which motion is selected. For instance, if we learn three motions, we have three codes $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$. Then, each code is used together with a different set of demonstrations to optimize ℓ_{bc}. To interpolate between these motions, we select an input code of the DNN that has an intermediate value between the ones of the motions, e.g., $[0.5, 0.5, 0.0]$.

To ensure stability for all of the interpolated motions, we can minimize ℓ_{stable} for each motion and also for the ones in the interpolation space, which should create a bijective mapping between the complete input of the DNN (state and code) and its latent space. Part of this can be observed in Fig. 5.17, which shows a t-SNE [200] projection of three manifolds corresponding to the mapping of the state space of three motions to the DNN's latent space. Since each motion is mapped to a different region of the latent space, it is possible to move in between these regions to create interpolated motions.

Fig. 5.18 shows these motions and some examples of their interpolation. We can note that the interpolation works properly, where features of different motions are

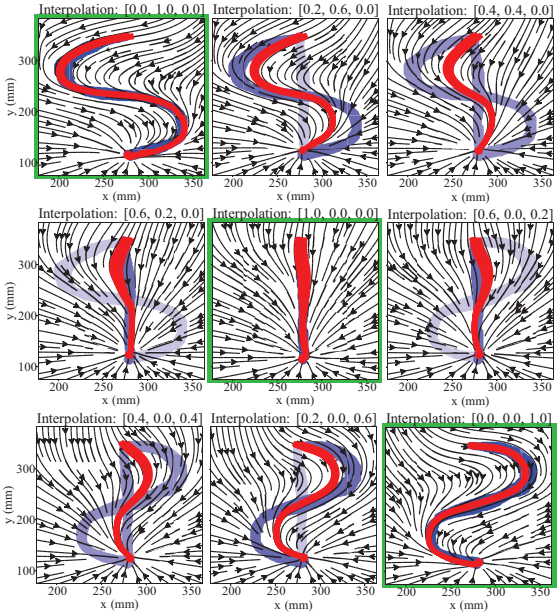


Figure 5.18.: Different motions learned in one DNN. Blue curves correspond to demonstrations, the darker their color, the more they influence each motion. Arrows represent the vector field of the motion, and red curves correspond to simulations of trajectories executed by the model. The figures highlighted in green correspond to the cases where the BC loss is minimized. The remaining figures correspond to interpolated motions. In the title of each plot, we can observe the code provided to the network to generate each motion.

combined to create novel behaviors. Furthermore, we observe that, as expected, the closer to a motion we interpolate, the more features of this motion the interpolated one presents. Finally, regarding stability, every motion has zero unsuccessful trajectories with $L = 2000$, $P = 1225$, and $\epsilon = 10$px.

5.8. Conclusions

In the context of robotic reaching motions modeled as dynamical systems, we introduce a novel contrastive loss that extends current Imitation Learning frameworks to achieve globally asymptotically stable behaviors. We optimize this loss together with a Behavioral Cloning loss, which, despite its practical limitations due to the covariate shift problem, can achieve state-of-the-art results by minimizing the multi-step loss instead of the single-step loss, as observed in our experiments. Importantly, our stability loss can also be employed with other Imitation Learning approaches, though its effectiveness with other losses remains untested.

Further experiments demonstrate that our framework, CONDOR, can effectively learn stable and accurate motions across various scenarios. These experiments were conducted in both simulated settings and with a real robot. We observed that CONDOR learns successful behaviors in 1) 2-dimensional first-order motions (LASA dataset), 2) 3-dimensional first-order motions (hammer hanging), 3) 4-dimensional second-order motions (LAIR dataset and writing two), and 4) 6-dimensional motions (cleaning table). Lastly, we observe that CONDOR can be extended to learn multiple motions and interpolate between them, allowing it to generate more stable behaviors without requiring more demonstrations. This interesting area of research will be explored further in future work.

While this chapter's findings are promising, they also reveal limitations that inspire other future research directions. Firstly, our method has only been tested on relatively low-dimensional state spaces; its applicability in higher-dimensional spaces remains unexplored. Additionally, we assume that the employed state representations used are minimal, a condition not always met in robotics. For instance, orientation representations often employ non-Euclidean manifolds (e.g., unit quaternions or rotation matrices) that introduce constraints, making the state representations non-minimal [201]. A further assumption is the existence of a low-level controller capable of generating the state transitions requested by CONDOR. While this is reasonable for manipulators, it can be a limiting factor in highly underactuated robots. Finally, CONDOR assumes that a proper state estimation (e.g., robot pose, goal location, or obstacles) is achieved by other modules.

6

Deep Metric Imitation Learning for Stable Motion Primitives

This chapter is based on the publication:

R. Pérez-Dattari, C. Della Santina, and J. Kober. "Deep Metric Imitation Learning for Stable Motion Primitives." In: *Advanced Intelligent Systems* (2024)

6.1. Introduction

Imitation Learning (IL) provides a powerful framework for the intuitive programming of robotic systems. Its strength lies in its ability to leverage human-like learning methodologies, such as demonstrations and corrections, making it accessible to non-robotics experts. This attribute significantly reduces the resources needed to build robotic systems. However, the data-driven nature of these methodologies presents a challenge: providing guarantees about the learned behaviors.

In the context of reaching motions, it is crucial for the robot's motions to consistently reach the intended target, irrespective of the robot's initial conditions. Thus, modeling motions as dynamical systems proves beneficial. This approach turns the problem into a question of ensuring *global asymptotic stability* (or *stability*, for short) at the goal, and tools from dynamical system theory can then be applied to guarantee this property.

Numerous methods have been proposed to ensure stability in motions represented by dynamical systems. However, they often exhibit at least one of the following limitations: 1) constraining the structure of their function approximators, and/or 2) being designed with the assumption that the robot's state space is Euclidean. To elaborate:

1. **Constrained function approximators.** To ensure stability guarantees, methods often constrain the structure of their function approximators. For instance, some approaches necessitate invertibility [159, 162, 163], while others require positive or negative definiteness [158, 164, 165]. However, these methods do not enable the full exploitation of modern Deep Neural Network (DNN) architectures, as these constraints are not typically present in DNNs. This limitation hinders their broader application in more complex models, where integrating these constraints is challenging. Furthermore, inherently constraining function approximators can overly restrict the range of solutions to which they can converge, resulting in less flexible models than necessary. This leads to suboptimal IL capabilities. In our context, this problem is known as the *stability versus accuracy dilemma* [20].

2. **Euclidean assumption.** Learned motions must integrate with the geometry of the space used to represent a robot's state. For example, the end-effector of a manipulator should always reach the intended target in both position and orientation space. However, orientations are often represented in non-Euclidean spaces, such as $SO(3)$ or $\mathcal{S}^3$. This is because Euclidean representations like Euler angles may not always provide a continuous description of motions that are inherently continuous[1] [201]. Such continuity is often a requirement for motions modeled as dynamical systems. Furthermore, the generalization capabilities of function approximations are compromised by non-continuous representations. To enable proper generalization, states that are close in the real world should be represented as closely related in the function approximator's input, i.e., the state space representation.

[1] This issue stems from singularities and non-unique representations.

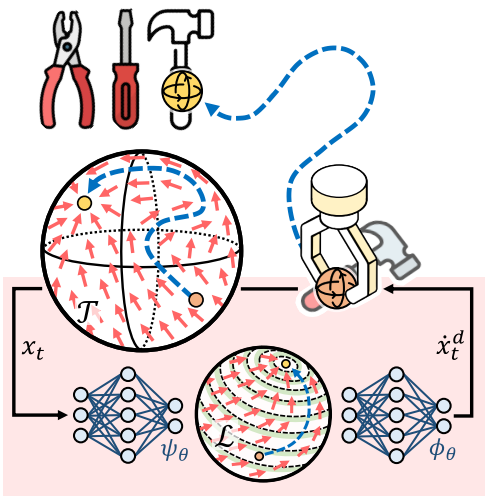


Figure 6.1.: Motion learned using the proposed framework. The blue trajectory in the task space $\mathcal{T}$ demonstrates the evolution of the robot's end effector state x_t when represented in a spherical manifold. The evolution of this trajectory is governed by the dynamical system $\dot{x}_t = \phi_\theta(\psi_\theta(x_t))$, depicted as a vector field of red arrows in the remaining of the space. Through Deep Metric Learning, this system is stabilized by deriving a simpler representation in the latent space $\mathcal{L}$.

6

Nevertheless, previous methods for stable motion generation have been initially designed under the assumption that the state space is Euclidean [156, 158, 159, 163]. Consequently, some have later been explicitly adapted to account for the geometry of motions that consider orientations [203–205], resulting in rather convoluted learning frameworks.

In this work, we present a DNN framework capable of learning accurate, stable motions in state spaces with arbitrary geometries without constraining the DNN's architecture. To accomplish this, we introduce a novel loss function that repurposes the *triplet loss*, commonly used in deep metric learning literature [168, 181]. We prove that this loss imposes conditions on the DNN's latent space that enforce the learned dynamical system to have a globally asymptotically stable equilibrium at the motion's goal state (see Fig. 6.1). To account for the geometry of the state space, it is sufficient to consider its corresponding metric during the computation of the loss, the choice of which depends on the specific task at hand. We validate our method in various settings, including Euclidean, non-Euclidean, first-order, and second-order motions. Additionally, real-world experiments controlling the 6-dimensional pose (x-y-z position and unit quaternion orientation) of a robot manipulator's end effector demonstrate the method's practical applicability and potential.

The rest of this chapter offers a thorough discussion of our developments. Following a review of the relevant literature, we delve into foundational concepts

and problem formulation. We then present the details of our methodology, provide a proof regarding the stability of the learned motion, and discuss the integration of the triplet loss function within the context of non-Euclidean state representations. We validate our approach through several experiments and conclude by considering potential directions for future research based on our findings.

6.2. Related Work

Three categories of works are relevant to this chapter. First, there are papers that focus on learning stable motions, assuming these motions occur in Euclidean state spaces. Second, others explore learning motions in non-Euclidean state spaces, but do not consider the stability of the learned motions. Finally, a third set of papers addresses both learning motions in non-Euclidean state spaces and their stability. Importantly,

In every case, works use either *time-varying (non-autonomous)* or *time-invariant (autonomous)* dynamical systems for motion modeling. In time-varying systems, evolution explicitly depends on time (or a phase). Conversely, time-invariant systems do not directly depend on time; instead, they rely on their time-varying input (i.e., the state of the system). The property of a system being time-invariant or not dictates the strategies we can use to ensure its stability. Thus, making a distinction between these systems is important. Notably, both types of formulations have been shown to be complementary in the context of IL [5, 174].

This work focuses on time-invariant dynamical systems. Consequently, although we consider both methodologies for motion learning, we delve deeper into the literature on time-invariant systems. Furthermore, while we concentrate on methods that ensure asymptotic stability, we acknowledge that there are studies imposing other conditions, such as via-point conditioning [206, 207], which can be significantly relevant in some robotic contexts.

6.2.1. Stability in Euclidean State Spaces

Regarding time-varying dynamical systems, a seminal work in IL that addresses the problem of learning stable motions introduces Dynamical Movement Primitives (DMPs) [157]. DMPs take advantage of the time-dependency (via the phase of the *canonical system*) of the dynamical system to make it evolve into a simple and well-understood system as time goes to infinity. The simple system is designed to be stable by construction. As a consequence, the stability of the learned motions can be guaranteed. This concept has been extended in multiple ways, for instance through probabilistic formulations [170, 208], or adapted to the context of DNNs [171–173].

Conversely, IL approaches based on time-invariant dynamical systems often constrain the function approximator used to model the dynamical systems. Such constraints ensure stability by construction. One seminal work introduces the Stable Estimator of Dynamical Systems (SEDS) [158]. This approach imposes constraints on the structure of a Gaussian Mixture Regression (GMR) such that the conditions for Lyapunov stability are always met. Multiple extensions of

SEDS have been proposed, for instance by using physically-consistent priors [176], contraction theory [177] or diffeomorphisms [178].

Other works propose explicitly learning Lyapunov functions that are consistent with the demonstrations. These functions are then used to correct the transitions learned by the dynamical systems, ensuring that they are always stable according to the learned Lyapunov functions [164, 165, 175]. Moreover, some papers have employed concepts such as contraction metrics [179, 180] and diffeomorphisms [159, 162, 163, 209] to impose stability, in the sense of Lyapunov, in time-invariant dynamical systems.

Understandably, all of these methods constrain some part of their learning framework to ensure stability. From one perspective, this is advantageous as it guarantees stability. However, in many cases, this comes at the expense of reduced accuracy in the learned motions. Notably, some recent methods have managed to mitigate this loss in accuracy [159, 163]. Nevertheless, they are still limited in terms of the family of models that can be used with these frameworks, which harms their scalability.

In Chapter 5, we addressed this issue by employing tools from the deep metric learning literature [168]. There, we introduced CONDOR, which uses a *contrastive loss* to enforce stability in learned motions through the optimization process of a DNN. This approach proved effective in learning stable, accurate, and scalable motions from human demonstrations. However, this method presents two important limitations. First, it requires the design of a stable and well-understood system for computing the contrastive loss (referred to as $f^{\mathcal{L}}$ in Sec. 6.3.3), which can significantly impact learning performance. Second, similar to the other methods described in this subsection, CONDOR is limited to operating within Euclidean state spaces.

In this work, we introduce a novel deep metric learning loss that ensures stability while addressing the limitations of CONDOR, specifically, the need for the function $f^{\mathcal{L}}$ and the inability to learn motions in non-Euclidean manifolds. Moreover, this loss requires weaker conditions for imposing stability, leading to, for instance, more robust performance when learning second-order dynamical systems.

6.2.2. Stability in Non-Euclidean State Spaces

As noted previously, it is crucial to consider the geometry of our state spaces when learning motions requiring pose control. Consequently, many studies have adapted methods that originally assumed data from Euclidean spaces to work with data from non-Euclidean manifolds, such as $SO(3)$ and $\mathcal{S}^3$. In the context of IL, pioneering approaches utilized time-varying dynamical systems (extensions of DMPs) to incorporate the geometry of orientation representations into their models [201, 210, 211]. Because DMPs inherently ensure stability, these methods are stable as well.

In contrast, although several geometry-aware IL approaches based on time-invariant dynamical systems have been introduced, such as Gaussian process regressions [212, 213], GMRs [214–217], and kernelized movement primitives [218, 219], such methods are not inherently stable. As a result, in these cases, models

need to be explicitly endowed with stability properties. This limitation has been addressed in recent works, where [220] uses GMRs and employs contraction theory to ensure stability considering the robot's orientation geometry. Furthermore, recent methods have extended the use of diffeomorphism-based techniques for stability to generate motions in non-Euclidean manifolds as well [203–205].

These diffeomorphism-based methods are of particular interest to our work, as our method is grounded in similar concepts for achieving stability. In both approaches, we transfer the stability properties of a simple system in the latent space of a DNN to the dynamical system that models motion in task space. However, unlike these methods, our approach learns this property, whereas the other works constrain the DNN structure to ensure its satisfaction. Moreover, our method is not constrained to diffeomorphic solutions for transferring the stability properties of the simple system to task space. Lastly, our method allows us to seamlessly incorporate the geometrical aspects of the robot's state space into the learned model, as this integration is factored into the DNN's optimization process.

6.3. Preliminaries

6.3.1. Dynamical Systems for Reaching Tasks

In this work, we model motions as nonlinear time-invariant dynamical systems represented by

$$\dot{x}_t = f(x_t), \tag{6.1}$$

where $x_t \in \mathcal{T}$, with $\mathcal{T} \subseteq \mathbb{R}^n$ being the task space, is the robot's state, and $f : \mathcal{T} \to \mathbb{R}^n$ is a differentiable function. Additionally, the subscript t indicates the time instance to which the state corresponds. Since we are interested in solving reaching tasks, we aim to construct systems with a globally asymptotically stable equilibrium at a goal state x_g. This implies that

$$\lim_{t\to\infty} ||x_\mathrm{g} - x_t|| = 0, \qquad \forall x_t \in \mathcal{T}. \tag{6.2}$$

6.3.2. Problem Formulation

We consider a robot learning a reaching motion in the space $\mathcal{T}$ towards the goal state $x_\mathrm{g} \in \mathcal{T}$. Based on a set of demonstrations $\mathcal{D}$, the robot is expected to imitate the behavior shown in the demonstrations while always reaching x_g, regardless of its initial state. The dataset $\mathcal{D}$ contains N trajectories τ. These trajectories show the evolution of a dynamical system's state x_t when starting from some initial condition x_0.

We assume that the demonstrations are drawn from an *optimal* distribution over trajectories, denoted as $p^*(\tau)$, that adheres to the *optimal* dynamical system f^*. Here, "optimal" refers to the behavior that the demonstrator deems best.

The robot's motion follows the parametrized system $f_\theta^{\mathcal{T}}$, inducing the distribution $p_\theta(\tau)$, where θ is a parameter vector. Then, the objective is to find the vector θ^* that minimizes the difference, expressed as the (forward) Kullback-Leibler

divergence, between $p_\theta(\tau)$ and $p^*(\tau)$, while ensuring that x_g is a globally asymptotically stable equilibrium:

$$\theta^* = \arg\min_{\theta\in\Theta} \quad D_{\mathrm{KL}}\left(p^*(\tau)||p_\theta(\tau)\right) \tag{6.3a}$$

$$\text{s.t.} \quad \lim_{t\to\infty} d(x_g, x_t) = 0, \forall x_t \in \mathcal{T}. \tag{6.3b}$$

Here, Θ is the parameter space of a DNN, and $d(\cdot,\cdot)$ is a distance function.

6.3.3. Stability Conditions

In [156], the *stability conditions* are formulated to enforce stability in $f_\theta^{\mathcal{T}}$. This is achieved by ensuring that $f_\theta^{\mathcal{T}}$ inherits the stability properties of a simple and stable system, referred to as $f^{\mathcal{L}}$. Thus, if $f^{\mathcal{L}}$ is asymptotically stable, $f_\theta^{\mathcal{T}}$ will also be asymptotically stable. To accomplish this, the system $f^{\mathcal{L}}$ is designed to define the evolution of a state y_t with an initial condition derived from mapping x_0 to the output of a hidden layer in the DNN that parameterizes $f_\theta^{\mathcal{T}}$. As a result, the dynamical system $f_\theta^{\mathcal{T}}$ is expressed as a composition of two functions, ψ_θ and ϕ_θ,

$$\dot{x}_t = f_\theta^{\mathcal{T}}(x_t) = \phi_\theta(\psi_\theta(x_t)). \tag{6.4}$$

Here, $f_\theta^{\mathcal{T}}$ is a standard DNN with L layers. ψ_θ denotes layers $1, ..., l$, and ϕ_θ layers $l+1, ..., L$. We define the output of layer l as the latent space $\mathcal{L} \subset \mathbb{R}^m$, which is where y_t resides, i.e., $y_t \in \mathcal{L}$. Note that $\mathcal{L} \subset \mathbb{R}^m$ implies the dimensionality of the vectors used to represent $\mathcal{T}$ and $\mathcal{L}$ **does not need to be the same**. Moreover, for simplicity, although we use the same θ notation for both ψ_θ and ϕ_θ, each symbol actually refers to a different subset of parameters within θ. These subsets together form the full parameter set in $f_\theta^{\mathcal{T}}$.

Additionally, we introduce a third dynamical system. This system denotes the evolution in $\mathcal{L}$ of the states visited by $f_\theta^{\mathcal{T}}$ when mapped using ψ_θ, which yields the relationship

$$\dot{y}_t = f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t) = \frac{\partial \psi_\theta(x_t)}{\partial t}. \tag{6.5}$$

Fig. 6.2 provides an example of the introduced dynamical systems.

Then, the stability conditions of [156] can be written as:

Theorem 2 (Stability conditions: v1)**.** *Let $f_\theta^{\mathcal{T}}$, $f_\theta^{\mathcal{T}\to\mathcal{L}}$ and $f^{\mathcal{L}}$ be the introduced dynamical systems. Then, in the region $\mathcal{T}$, x_g is a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}}$ if, $\forall x_t \in \mathcal{T}$,:*

1. $f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t) = f^{\mathcal{L}}(y_t)$,
2. $\psi_\theta(x_t) = y_g \Rightarrow x_t = x_g$.

These conditions imply that if the system $f_\theta^{\mathcal{T}\to\mathcal{L}}$ behaves like $f^{\mathcal{L}}$, and any point other than x_g is excluded from mapping to the latent goal y_g, then $f_\theta^{\mathcal{T}}$ is stable. Note that the latent goal is defined by the mapping $\psi_\theta(x_g) = y_g$.

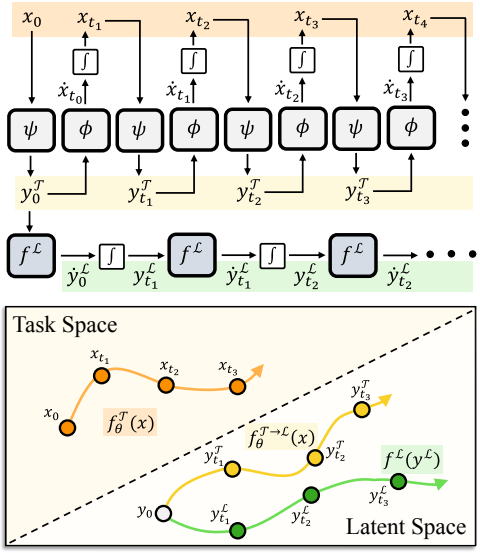


Figure 6.2.: Example of trajectories generated by simulating the systems $f_\theta^{\mathcal{T}}$, $f_\theta^{\mathcal{T}\to\mathcal{L}}$ and $f^{\mathcal{L}}$ for different time instants. The stability conditions are not met in this case, as $f_\theta^{\mathcal{T}\to\mathcal{L}}$ differs from $f^{\mathcal{L}}$.

In this work, we reformulate these conditions and propose a novel way to optimize them. This adaptation endows the learning framework with increased flexibility, enabling it to tackle a wider array of problems (e.g., non-Euclidean state spaces) and achieve improved performance.

6.3.4. Deep Metric Learning: the Triplet Loss

Commonly employed in the Deep Metric Learning literature, the *triplet loss* [181] has been utilized for learning and structuring latent state representations [168]. Its function is to cluster similar observations together and differentiate dissimilar ones within the latent space of a DNN. In this work, however, the triplet loss is used differently. Although we continue to use this loss to impose a certain structure on the DNN's latent space, its purpose is to enforce the stability conditions, thereby ensuring that $f_\theta^{\mathcal{T}}$ is stable.

Let us recall the triplet loss:

$$\ell_{\text{triplet}} = \max(0, m + d(a, p) - d(a, n)), \tag{6.6}$$

where $m \in \mathbb{R}_{>0}$ is the margin, a is the anchor sample, p the positive sample, and n the negative sample This loss enforces positive samples to be at least m distance closer to the anchor than the negative samples. Notably, this is enough for the loss to become zero, i.e., it does not require the anchor and positive samples to have the same value.

6.3.5. Stability Analysis through Comparison Functions

In this work, we employ comparison functions, namely class-$\mathcal{KL}$ functions, to prove global asymptotic stability at the equilibrium x_g. These functions are used to formulate a general approach for stability analysis in the sense of Lyapunov. As described in [221, 222], these functions are defined as follows:

Definition 1 (class-$\mathcal{K}$ function)**.** A continuous function $\alpha : [0, a) \to \mathbb{R}_{\geq 0}$, for $a \in \mathbb{R}_{>0}$, is said to belong to class $\mathcal{K}$ if it is strictly increasing and $\alpha(0) = 0$.

Definition 2 (class-$\mathcal{L}$ function[2])**.** A continuous function $\sigma : \mathbb{R}_{\geq 0} \to \mathbb{R}_{>0}$, is said to belong to class $\mathcal{L}$ if it is *weakly decreasing*[3] and $\lim_{s\to\infty} \sigma(s) = 0$.

Definition 3 (class-$\mathcal{KL}$ function)**.** A function $\beta : [0, a) \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, for $a \in \mathbb{R}_{>0}$, is said to belong to class-$\mathcal{KL}$ if:

- for each fixed s, the mapping $\beta(r, s)$ belongs to class-$\mathcal{K}$ with respect to r,
- for each fixed r, the mapping $\beta(r, s)$ belongs to class-$\mathcal{L}$ with respect to s.

Then, we can describe global asymptotic stability in terms of class-$\mathcal{KL}$ functions as:

Theorem 3 (Global asymptotic stability with class-$\mathcal{KL}$ functions)**.** *The state x_g is a globally asymptotically stable equilibrium of (6.1) in $\mathcal{T}$ if there exists a class-$\mathcal{KL}$ function β such that, $\forall t \in \mathbb{R}_{\geq 0}$ and $\forall x_0 \in \mathcal{T}$,*

$$||x_g - x_t|| \leq \beta(||x_g - x_0||, t). \tag{6.7}$$

Note that this theorem seamlessly integrates the concepts of *stability* and *attractivity* within a single function β, which acts as an upper bound of $||x_g - x_t||$. As the initial condition of the system moves further away from x_g, β correspondingly increases. Moreover, as the system evolves over time and β decreases, it follows that the system's distance to x_g will eventually decrease.

6.4. Method

We introduce the *Policy via neUral Metric leArning* (PUMA) framework, which learns motion primitives from human demonstrations parametrized as the dynamical system $f_\theta^{\mathcal{T}}$. Furthermore, it enforces the goal of the motion x_g to be a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}}$ while maintaining accuracy with respect to the demonstrations. To achieve this, we augment the Imitation Learning (IL) problem with the stability-enforcing loss ℓ_{stable}. Hence, our framework minimizes the loss:

$$\ell_{\text{PUMA}} = \ell_{\text{IL}} + \lambda \ell_{\text{stable}}, \tag{6.8}$$

where ℓ_{IL} is an imitation learning loss and $\lambda \in \mathbb{R}_{>0}$ is a weight factor.

[2] Note that the symbol $\mathcal{L}$ is used in two distinct contexts. While it represents the class-$\mathcal{L}$ functions, it is also used to denote the set $\mathcal{L}$ introduced in Sec. 6.3.3. In subsequent sections of the chapter, the $\mathcal{L}$ referring to class-$\mathcal{L}$ functions is exclusively used in the form $\mathcal{KL}$.

[3] We use this term to denote functions that either remain constant or strictly decrease within any interval of their domain.

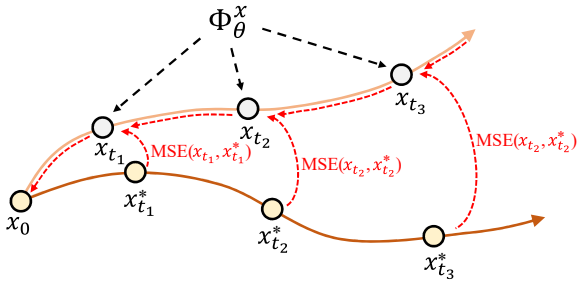


Figure 6.3.: Illustration of the behavioral cloning loss computation. Starting from an initial condition x_0, the system $f_\theta^\mathcal{T}$ evolves to various time instants via Φ_θ^x. At each instant, the estimated state is compared with a demonstrated state. The red arrows show the gradient path used to update the DNN's weights using BPTT.

6.4.1. Behavioral Cloning

To minimize ℓ_{IL}, and thereby address (6.3a), we adopt the behavioral cloning loss used in [156]. This loss tackles (6.3a) by modeling the output of the deterministic dynamical system $f_\theta^\mathcal{T}$ as the mean of a Gaussian distribution with fixed covariance. Furthermore, it mitigates *covariate shift* by minimizing the multi-step error over trajectory segments using *backpropagation through time* (BPTT), as depicted in Fig. 6.3.

In every training iteration, we sample a batch $\mathcal{B}^i$ of trajectory segments $\mathcal{H}^i$ from the dataset $\mathcal{D}$. These segments can start at any point within a given demonstration, with the start time defined as $t = 0$. Then, by introducing the *evolution function* $\Phi_\theta^x(t, x_0) : \mathbb{R}_{\geq 0} \times \mathcal{T} \to \mathcal{T}$, which defines the value of x_t by integrating $f_\theta^\mathcal{T}$ between 0 and t, with initial condition x_0, we can construct the loss

$$\ell_{\text{IL}} = \sum_{\mathcal{H}^i \in \mathcal{B}^i} \sum_{(t, x_t^*) \in \mathcal{H}^i} ||x_t^* - \Phi_\theta^x(t, x_0)||_2^2. \tag{6.9}$$

Here, the states x_t^* along the trajectory segment $\mathcal{H}^i$ serve as labels for the states predicted by the DNN from the initial condition x_0 using $\Phi_\theta^x(t, x_0)$. Note that the initial condition is obtained from $\mathcal{H}^i$, i.e., $x_0 = x_0^*$.

Since we do not have an analytical solution of Φ_θ^x, we approximate it using the *forward Euler method*, i.e.,

$$\Phi_\theta^x(t, x_0) = \Phi_\theta^x(t', x_0) + f_\theta^\mathcal{T}\left(\Phi_\theta^x(t', x_0)\right) \Delta t, \tag{6.10}$$

where $t' = t - \Delta t$ and $\Delta t \in \mathbb{R}_{>0}$ is the time step size. This integration starts with the initial state x_0, i.e., $\Phi_\theta^x(0, x_0) = x_0$. It is important to note that the recursive nature of $\Phi_\theta^x(t, x_0)$ necessitates the use of BPTT for the optimization of the DNN.

6.4.2. Triplet Stability Loss

Reformulating the Stability Conditions

The stability conditions of Theorem 2 involve three dynamical systems: $f_\theta^{\mathcal{T}}$, $f_\theta^{\mathcal{T}\to\mathcal{L}}$, and $f^{\mathcal{L}}$. Note, however, that its first condition, namely, $f_\theta^{\mathcal{T}\to\mathcal{L}} = f^{\mathcal{L}}\ \forall x_t \in \mathcal{T}$, essentially states that $f_\theta^{\mathcal{T}\to\mathcal{L}}$ must exhibit global asymptotic stability. In other words, if the behavior of $f_\theta^{\mathcal{T}\to\mathcal{L}}$ is identical to that of another system, $f^{\mathcal{L}}$, for which stability is verified, then the stability of $f_\theta^{\mathcal{T}\to\mathcal{L}}$ is also verified. Nonetheless, if we can enforce its stability through a different method, these conditions can be more generally written as follows:

Theorem 4 (Stability conditions: v2)**.** *Let $f_\theta^{\mathcal{T}}$ and $f_\theta^{\mathcal{T}\to\mathcal{L}}$ be the introduced dynamical systems. Then, in the region $\mathcal{T}$, x_{g} is a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}}$ if, $\forall x_t \in \mathcal{T}$,:*

1. *y_{g} is a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}\to\mathcal{L}}(x_t)$,*
2. *$\psi_\theta(x_t) = y_{\mathrm{g}} \Rightarrow x_t = x_{\mathrm{g}}$.*

Surrogate Stability Conditions

Theorem 4 introduces stability conditions for $f_\theta^{\mathcal{T}}$; nevertheless, it does not specify how to enforce these conditions in the system. Therefore, we introduce the *surrogate stability conditions* of Theorem 4. These conditions, when met, imply that the stability conditions of Theorem 4 are also satisfied. Unlike the stability conditions, the surrogate conditions can be directly transformed into a specific loss function, ℓ_{stable}, for optimizing the DNN to enforce their satisfaction.

To formulate the surrogate stability conditions, we note that Theorem 4 can be expressed in terms of relative distances. We define the distance between any given latent state y_t and the goal state y_{g} as $d_t = d(y_{\mathrm{g}}, y_t) = \|y_{\mathrm{g}} - y_t\|$. Then, according to Condition 1 of Theorem 4, which addresses global asymptotic stability, the value of d_t should, generally speaking, decrease over time. Moreover, the second condition specifies that the value of d_t should remain constant only for $y_{\mathrm{g}} = \psi_\theta(x_{\mathrm{g}})$. Formally, we introduce the conditions as:

Theorem 5 (Surrogate stability conditions)**.** *Let two dynamical systems be governed by the equations $\dot{x}_t = f_\theta^{\mathcal{T}}(x_t)$ and $\dot{y}_t = f_\theta^{\mathcal{T}\to\mathcal{L}}(y_t)$, such that $y_t = \psi_\theta(x_t)$. Assume both $f_\theta^{\mathcal{T}}$ and $f_\theta^{\mathcal{T}\to\mathcal{L}}$ are continuously differentiable. Then, in the region $\mathcal{T}$, x_{g} is a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}}$ if, $\forall t \in \mathbb{R}_{\geq 0}$:*

1. *$d_t = d_{t+\Delta t}$, for $y_0 = y_{\mathrm{g}}$,*
2. *$d_t > d_{t+\Delta t}$, $\forall y_0$ with $x_0 \in \mathcal{T} \setminus \{x_{\mathrm{g}}\}$,*

where $\Delta t \in \mathbb{R}_{>0}$.

Proof. To prove this theorem, we demonstrate that if the surrogate stability conditions are satisfied, then the stability conditions from Theorem 4 must also hold. This leads to x_{g} being a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}}$.

Second stability condition of Theorem 4 Let us begin by analyzing the fulfillment of the second stability condition from Theorem 4. This condition states that only x_g can map to y_g via ψ_θ. Then, since y_g is defined as $\psi_\theta(x_g)$, we need to establish that no other x_t maps to y_g. To achieve this, we note that both x_t and x_0 belong to the same state space $\mathcal{T}$. Therefore, showing that this statement holds $\forall x_0 \in \mathcal{T}$ implies that it also holds $\forall x_t \in \mathcal{T}$.

Now, suppose for a contradiction that there exists some x_0 such that $x_0 \neq x_g$ and $y_0 = \psi_\theta(x_0) = y_g$. Then, according to the second surrogate condition, $d_t > d_{t+\Delta t}$, which contradicts the first surrogate condition. Consequently, the second stability condition must be satisfied.

First stability condition Theorem 4 Let us now study the first stability condition. Our goal is to prove that y_g is a globally asymptotically stable equilibrium of $f_\theta^{\mathcal{T}\to\mathcal{L}}$ within the region $\mathcal{L}$, where the system is defined. Surrogate condition 2 hints that the system's stability could be verified through a Lyapunov candidate defined using d_t. This is because the condition enforces the distance d_t to strictly decrease within the interval defined by Δt. However, this does not necessarily imply that the Lyapunov candidate strictly decreases with time, as it is possible for it to strictly increase locally while adhering to this condition, as exemplified in Fig. 6.4. As a result, we proceed to demonstrate the global asymptotic stability of y_g using Theorem 3. Specifically, we aim to do so by employing a class-$\mathcal{KL}$ upper bound β that fulfills (6.7).

To achieve this, we first define an evolution function for the distance d_t, for a given y_0 and t, as $\delta : \mathcal{L} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, with $\delta(y_0, t) = ||y_g - \Phi_\theta^y(t, y_0)||$. Here, Φ_θ^y represents the evolution function of y_t under the dynamical system $f_\theta^{\mathcal{T}\to\mathcal{L}}$. Then, we can express the upper bound β as

$$\delta(y_0, t) \leq \beta(d_0, t), \tag{6.11}$$

where $d_0 = \delta(y_0, 0)$. Recall that for ensuring asymptotic stability, β must also satisfy the following properties: i) $\beta(0, t) = 0$, ii) β weakly decreases with t, iii) β is continuous with respect to d_0 and t, iv) $\beta \to 0$ as $t \to \infty$, and v) β strictly increases with d_0.

It is important to note that verifying all these properties can lead to a lengthy proof. Thus, while we introduce β and discuss the key concepts behind its design here, the complete proof and details are provided in Proposition 3, Appendix C.1.

We now proceed to describe the three main aspects considered in the design of β.

1. *Upper bound in time:* First, we need to identify a β that weakly decreases with time. For this purpose, we introduce a function that computes the maximum of δ over the window $[t, t + \Delta t]$ for any given y_0 and t, i.e.,

$$\delta_{t+\Delta t}^{\max}(y_0, t) = \max_{s \in [t, t+\Delta t]} \delta(y_0, s). \tag{6.12}$$

   Clearly, this function serves as an upper bound for δ. Moreover, this function must weakly decrease with time. Considering surrogate condition 2, which

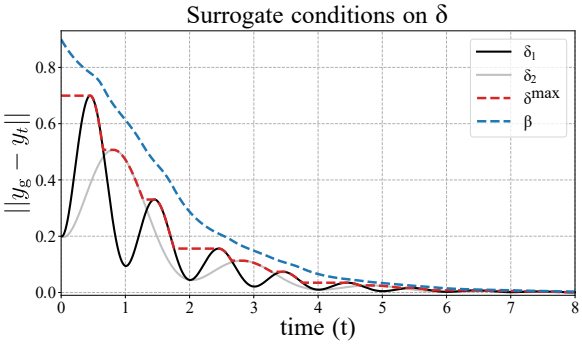


Figure 6.4.: Time evolution of two functions δ, δ_1 and δ_2, starting with different initial conditions y_0, but same d_0. Both satisfy the surrogate stability conditions with $\Delta t = 2$. Additionally, the values of $\delta^{\max}$ and β, computed using these functions, are shown. In this representation, $\delta = e^{-a \cdot t}\left(\sin^2(\omega t) + d_0 \cdot \cos^2(\omega t)\right)$ with $a = 0.75$, $d_0 = 0.2$, and $\omega = [\pi, \pi/2]$.

indicates that $\forall s \in [t, t+\Delta t]$, we have $\delta(y_0, s+\Delta t) < \delta(y_0, s)$, it follows that there is no δ greater than $\delta^{\max}_{t+\Delta t}$ in the interval $[t+\Delta t, t+2\Delta t]$. By extending this observation for every interval $[t+n\cdot\Delta t, t+(n+1)\cdot\Delta t]$, with $n \in \mathbb{N}$, we can conclude that $\delta^{\max}_{t+\Delta t}$ weakly decreases with time.

2. *Upper bound in space:* The function $\delta^{\max}_{t+\Delta t}(y_0, t)$ provides an upper bound of δ for a given y_0. However, $\beta(d_0, t)$ depends on d_0 rather than directly on y_0. To address this, for any specified d_0, we must ensure that $\beta \geq \delta$ for every y_0 located at this particular distance from the equilibrium. The set of initial conditions y_0 fulfilling this condition can be defined as $\mathcal{Y}_0(d_0) = \{y_0 \in \mathcal{L} : ||y_g - y_0|| = d_0\}$. Considering this, we can introduce an upper bound dependent on d_0 by computing the maximum of each $\delta^{\max}_{t+\Delta t}(y_0, t)$, where $y_0 \in \mathcal{Y}_0$:

$$\delta^{\max}(d_0, t) = \max_{y_0 \in \mathcal{Y}_0(d_0)} \left(\delta^{\max}_{t+\Delta t}(y_0, t)\right). \tag{6.13}$$

3. *Strictly increasing/decreasing function:* Similarly to $\delta^{\max}_{t+\Delta t}$, the function $\delta^{\max}$ also weakly decreases as a function of time (see Appendix C.1 for details). This is not inherently problematic, as it verifies the properties of class-$\mathcal{KL}$ functions. However, β must strictly increase as a function of d_0, and, for this to be the case, in our formulation, β must also strictly decrease as a function of time, $\forall d_0 \in \mathcal{L} \setminus \{y_g\}$.

   To achieve this, we consider β as the evolution function of a first-order linear dynamical system with state z_t, using $\delta^{\max}$ as the reference. This leads to the equation

$$\dot{z}_t = \alpha(z_t - \delta^{\max}), \tag{6.14}$$

   where $\alpha < 0$. With an initial condition z_0 greater than $\delta^{\max}_0 = \delta^{\max}(d_0, 0)$, we ensure that β strictly decreases with time. Moreover, β remains above $\delta^{\max}$, and

consequently above δ, for all $d_0 \in \mathcal{L} \setminus \{y_g\}$. Hence, by choosing $z_0 = \delta_0^{\max} + d_0$, we ensure that β exhibits the desired increasing/decreasing properties. We can express β as

$$\beta(d_0, t) = z_0 + \int_0^t \dot{z}_s ds. \tag{6.15}$$

Based on Proposition 3, Appendix C.1, we can confirm that our formulation for β is a valid class-$\mathcal{KL}$ upper bound of δ. This indicates that the first stability condition of Theorem 4 is satisfied, concluding our proof. □

Loss Function

Crucially, the surrogate stability conditions from Theorem 5 can be enforced in a DNN by minimizing the following expression, $\forall y_0 \in \mathcal{L}$ and $\forall t \in \mathbb{R}_{\geq 0}$,

$$\max\big(0, m + d(y_g, y_{t+\Delta t}) - d(y_g, y_t)\big), \tag{6.16}$$

where $m \in \mathbb{R}_{>0}$. This function resembles the form of the triplet loss introduced in Sec. 6.3.4. However, in our context, y_g serves as the anchor sample, $y_{t+\Delta t}$ as the positive sample, and y_t as the negative sample. In the following discussion, we explore how this expression induces the fulfillment of the surrogate stability conditions within a DNN.

Second surrogate condition For any y_0 where $x_0 \neq x_g$, minimizing (6.16) enforces transitions to progressively approach y_g, as its minimization implies

$$d(y_g, y_{t+\Delta t}) + m \leq d(y_g, y_t). \tag{6.17}$$

Hence, $d(y_g, y_{t+\Delta t}) < d(y_g, y_t)$, i.e., the second surrogate condition (see Fig. 6.5).

First surrogate condition In the case where $y_0 = y_g = \psi_\theta(x_g)$, that is, when the system is initialized at the equilibrium, enforcing a value of $y_{t+\Delta t}$ different from y_g would only increase the function in (6.16). Consequently, for $y_0 = y_g = \psi_\theta(x_g)$, (6.16) achieves its minimum when $y_{t+\Delta t} = y_g = y_0$, equal to m. Therefore, this equation also enforces the first surrogate condition.

Finally, since $\delta(y_0, t) = d(y_g, y_t)$, we present the stability loss function as

$$\ell_{\text{stable}} = \sum_{y_0 \in \mathcal{B}^s} \sum_{t \in \mathcal{H}^s} \max(0, m + \delta(y_0, t + \Delta t) - \delta(y_0, t)). \tag{6.18}$$

In this equation, $\mathcal{B}^s$ denotes a batch of initial latent states y_0. These states are derived by mapping initial states x_0 (**sampled randomly** from $\mathcal{T}$) via the function ψ_θ. Meanwhile, $\mathcal{H}^s$ represents a set of time instants t at which the loss is minimized.

To compute this loss, we must recall that the evolution function of y_t is represented as Φ_θ^y. Consequently,

$$\delta(y_0, t) = ||y_g - \Phi_\theta^y(t, y_0)||. \tag{6.19}$$

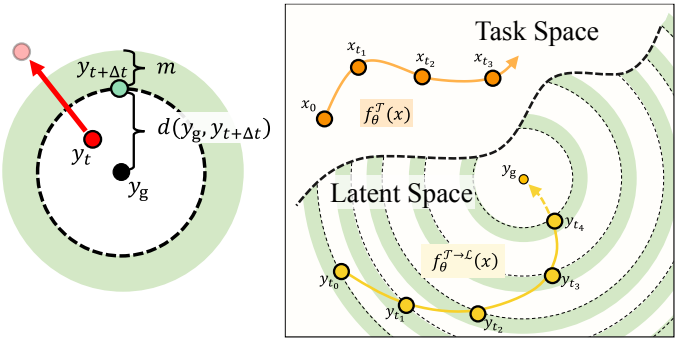


Figure 6.5.: Left: Illustration of the effect of optimizing ℓ_{stable}. The red arrow depicts y_t and $y_{t+\Delta t}$ being modified to fulfill (6.17). Right: Example of trajectories generated with $f_\theta^{\mathcal{T}}$ and $f_\theta^{\mathcal{T}\to\mathcal{L}}$ **post-training**. Each $y_{t+\Delta t}$ is closer to y_g than its predecessor y_t.

Given the absence of an analytical representation for Φ_θ^y, we approximate it using the forward Euler method, and optimize it using BPTT, in a similar manner to Sec. 6.4.1.

Importantly, optimizing this loss for all $t \in \mathbb{R}_{\geq 0}$ using BPTT is not feasible, as it would necessitate computing the loss over an infinite number of samples. Nevertheless, this limitation is not a significant concern when dealing with time-invariant dynamical systems. In such systems, any state y_t can be equivalently represented by an initial condition y_0, since both reside in the same state space $\mathcal{L}$. Consequently, when states are randomly sampled from $\mathcal{T}$, and thereby from $\mathcal{L}$, we are essentially sampling from the space of all possible y_t.

6.4.3. On the Stability Loss Metric

Intentionally, we have not specified which distance function or metric should be used for computing the stability loss, since it should be selected depending on the geometry employed to describe the robot's state space.

To elaborate, recall that the learned dynamical system can be expressed as $\dot{x}_t = \phi_\theta(y_t)$. It then follows that the output of our model is entirely determined by the latent state y_t. This implies that if two different states x_t^{a} and x_t^{b} map to the same latent state, that is, $y_t = \psi_\theta(x_t^{\text{a}}) = \psi_\theta(x_t^{\text{b}})$, the time derivative computed by the learned dynamical system for both states will be identical. Such behavior would manifest in certain states if ψ_θ were a non-bijective[4] function. It would, therefore, be potentially harmful for the learning process to enforce ψ_θ to be non-bijective, as this would constrain the family of solutions to which the system can converge, hindering the DNN's optimization process. Ideally, we would like ψ_θ

[4]More rigorously, the property being described is that of non-injective functions. However, for practical purposes, we can define the codomain of ψ_θ to be equal to its image, which is the property that an injective function must have to be bijective. Hence, in this specific context, we use these terms interchangeably.

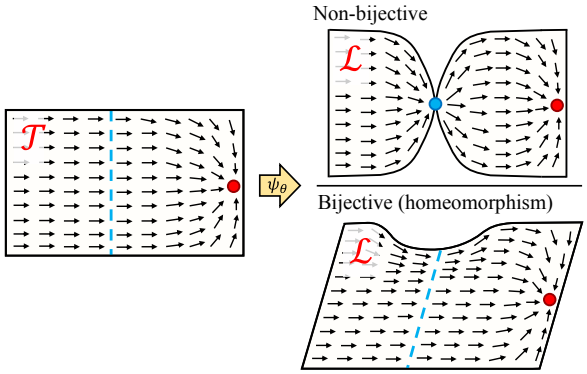


Figure 6.6.: Non-bijective vs bijective solution. In the non-bijective case, every point on the blue line in $\mathcal{T}$ maps to the blue point in $\mathcal{L}$. In the bijective case, the original line undergoes a transformation that both compresses and skews it, yet a one-to-one mapping is maintained.

to have the capacity to converge to a bijective function if required, where each state x_t maps to a unique latent state y_t. Consequently, for any state x_t, it would be possible to compute a value of $\dot{x}_t$ that is independent of those calculated for any other state. Fig. 6.6 presents an example of bijective and non-bijective mappings between dynamical systems where the surrogate stability conditions are enforced.

It turns out that the capacity of ψ_θ to converge to a bijective function is closely linked to the metric used in calculating ℓ_{stable}, which should be chosen based on the geometry of the robot's state space. In the following subsection, we explore this relationship in more depth.

Homeomorphisms and State Space Geometry

To study under which conditions ψ_θ can converge to a bijective function, let us introduce the concept of *topologically equivalent* manifolds. Two manifolds, e.g., $\mathcal{T}$ and $\mathcal{L}$, are topologically equivalent if a continuous and bijective mapping, known as an *homeomorphism*, exists between them [223]. In other words, two topologically equivalent manifolds can be *stretched*, *compressed*, or *twisted* into each other without *tearing* or *gluing* space. Since DNNs are continuous functions[5], a bijective function ψ_θ would also serve as a homeomorphism[6] between $\mathcal{T}$ and $\mathcal{L}$.

Moreover, since we have a notion of distance in both $\mathcal{T}$ and $\mathcal{L}$, it follows that these manifolds are metric spaces. A key property of metric spaces is that their topology is generated by their distance functions. Therefore, if two metric spaces are topologically equivalent, their metrics are termed as being *equivalent* [225]. It is important to clarify that this does not necessarily mean that the two

[5] We can assume this since every broadly used model is continuous [145, 224].

[6] DNNs are commonly continuously differentiable, so, sometimes in the literature the term *diffeomorphism* is employed instead, as diffeomorphisms are continuously differentiable homeomorphisms.

distance functions are identical, but rather that they induce metric spaces that are homeomorphic to each other.

In our context, this implies that the distance function employed in the stability loss (6.18) must induce a topology in $\mathcal{L}$ that is equivalent to that of $\mathcal{T}$. For example, if orientations are described using unit quaternions, the topology of $\mathcal{T}$ would be spherical. Then, the stability loss metric should generate a topology that is homeomorphic to the sphere. Otherwise, it would be infeasible for the DNN to establish a homeomorphism between $\mathcal{T}$ and $\mathcal{L}$.

In this work, Sec. 6.5, we use unit quaternions to represent orientations in our robot experiments. For a detailed discussion on metrics and pose control in this context, the reader is referred to Appendix C.2.

6.4.4. Boundary Conditions

Lastly, it is crucial to ensure that a dynamical system evolving in $\mathcal{T}$ always remains within this manifold. Two scenarios are relevant to this work: 1) ensuring $\mathcal{T}$ is positively invariant with respect to $f_\theta^{\mathcal{T}}$; and 2) considering the state's geometry when computing the evolution of the dynamical system.

Positively Invariant Sets

In PUMA, the stability of a motion is enforced by randomly sampling points from $\mathcal{T}$ and minimizing ℓ_{stable}. Thus, stability cannot be ensured in regions where this loss is not minimized, i.e., outside of $\mathcal{T}$. When boundaries are imposed on the robot's workspace, the learned dynamical system $f_\theta^{\mathcal{T}}$ can potentially evolve towards these boundaries, leaving $\mathcal{T}$. Hence, to ensure stability, a state evolving within $\mathcal{T}$ must not leave $\mathcal{T}$. In other words, $\mathcal{T}$ has to be a *positively invariant set* with respect to $f_\theta^{\mathcal{T}}$ [165, 221].

To address this, we design the dynamical system so that it cannot leave $\mathcal{T}$ by construction. This can be achieved by projecting any transitions that would leave $\mathcal{T}$ back onto its boundary. For example, in Euclidean state spaces, $\mathcal{T}$ can be represented as a hypercube; therefore, in this case, this projection is achieved by saturating/clipping the points that leave $\mathcal{T}$. Furthermore, we found that introducing an additional loss, denoted as ℓ_∂, can be beneficial in enforcing this condition through the optimization process of the DNN. As proposed in [165], this can be accomplished by using the scalar product between the dynamical system's velocity $v(x_t)$ (which equates to $f_\theta^{\mathcal{T}}$ for first-order systems) and the outward-pointing normal vector $n(x_t)$ at states within the boundary of $\mathcal{T}$. This product should be ensured to be equal to or less than zero. Then, to achieve this for DNNs, we introduce the following loss function

$$\ell_\partial = \max(0, n(x_t) \cdot v(x_t)). \tag{6.20}$$

Evolving in Non-Euclidean State Spaces

When modeling dynamical systems in non-Euclidean state spaces, it is crucial to ensure that states do not evolve outside the manifold representing them. For

instance, unit quaternions must remain within the unit sphere. However, if these states are evolved using Euclidean geometry tools, such as the forward Euler integration method, deviations from the manifold are likely to occur.

Non-Euclidean state spaces are commonly defined within a higher-dimensional Euclidean space and can sometimes be constructed by incorporating constraints into this space. Consider the unit quaternion as an example: its state space consists of every vector in $\mathbb{R}^4$ with a unit norm, forming the 3-sphere $\mathcal{S}^3 \subseteq \mathbb{R}^4$. Therefore, in such scenarios, by integrating an operation that enforces these constraints, such as normalization for $\mathcal{S}^3$, into the structure of the DNN representing $f_\theta^{\mathcal{T}}$, we ensure that transitions remain within the manifold. Moreover, in this way, the distortions that this operation introduces into the dynamical system's output are factored into the DNN's optimization process, ensuring they are accounted for.

Alternatively, when we have access to the *Riemannian metric* of a state space manifold, it is possible to use the *exponential* and *logarithmic maps* to do Euclidean calculus in the *tangent bundle* of the manifold, and then map the solution back on the manifold (the reader is referred to [226] for more details).

6.5. Experiments

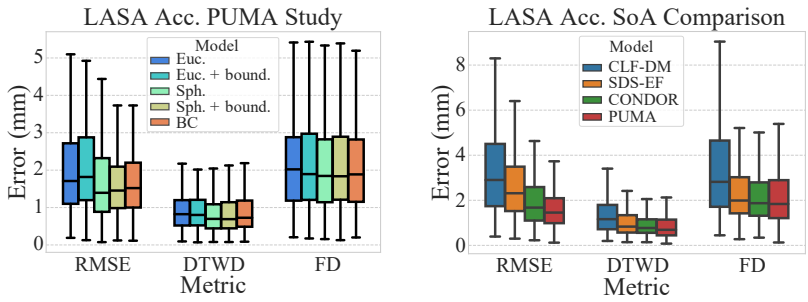
We validate our method with three datasets, each allowing us to study different aspects of it. For evaluation purposes, we use these datasets under the assumption of perfect tracking of the desired state derivatives, $\dot{x}_t^d$, provided by $f_\theta^{\mathcal{T}}$, without any involvement of robots in this process. Subsequently, we test our method in two real-world settings using two different robots. We have made our code implementation of PUMA publicly available at: `https://github.com/rperezdattari/Deep-Metric-IL-for-Stable-Motion-Primitives`. Details on the DNN's hyperparameters optimization process are presented in Appendix C.3.

6.5.1. Euclidean Datasets

Firstly, we evaluate our method using datasets of Euclidean motions. This enables us to study how different variations of PUMA perform for Euclidean motions and compare the performance of PUMA against state-of-the-art methods for stable Euclidean motion generation.

LASA

The LASA dataset [158] is composed of 30 human handwriting motions, each consisting of 7 demonstrations of desired trajectories under different initial conditions. These demonstrations are two-dimensional and designed to be modeled using first-order systems, i.e., output desired velocities as a function of their positions. To compare accuracy performance between different models, we employ the same metrics in every experiment: 1) Root Mean Squared Error (RMSE), Dynamic Time Warping Distance (DTWD) [194] and Fréchet Distance (FD) [195].



(a) Comparison of different variations of PUMA.

(b) Comparison of PUMA with other state-of-the-art (SoA) methods.

Figure 6.7.: Accuracy study in the LASA dataset.

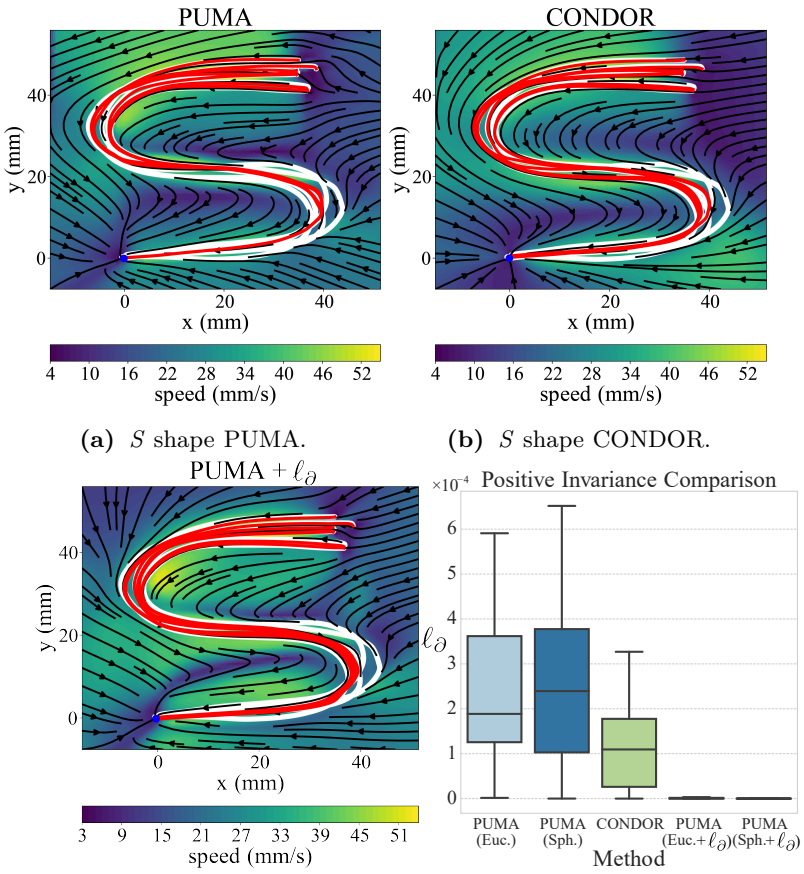
Table 6.1.: Percentage of unsuccessful trajectories over the LASA dataset ($L = 2500$, $P = 2500$, $\epsilon = 1$mm).

Behavioral Cloning	PUMA (Euc.)	PUMA (Euc. + ℓ_∂)	PUMA (Sph.)	PUMA (Sph. + ℓ_∂)
36.4653%	**0.0000%**	**0.0000%**	**0.0000%**	**0.0000%**

Accuracy Fig. 6.7a shows the accuracy of four variations of PUMA. Here, we compare the performance of different distance metrics in ℓ_{stable} for motions in Euclidean spaces, focusing on the Euclidean distance and the great-circle (spherical) distance. Furthermore, we also examine the influence of ℓ_∂ on the accuracy of the learned motions. We use Behavioral Cloning (BC) without a stability loss as an upper performance bound for comparison. Interestingly, each variation of PUMA achieves a similar performance; however, PUMA with a spherical metric shows slightly better performance than PUMA with an Euclidean metric. This suggests that the DNN has no difficulties in mapping the Euclidean space $\mathcal{T}$ into a spherical space $\mathcal{L}$. Lastly, we can also observe that the use of ℓ_∂ does not harm the accuracy performance of PUMA.

Stability The stability of the motions learned by PUMA hinges on the successful minimization of (6.8), which we need to test after the learning process concludes empirically. To do this, we integrate the dynamical system over L time steps, starting from P initial states, and observe whether the system converges to the goal. The larger the P, the more accurate our results. If L is sufficiently large, the system should reach the goal after L steps. By measuring the distance between the last state visited and the goal, and confirming that it falls below a pre-set threshold ϵ, we can evaluate if a trajectory is successful (i.e., it converges to the goal).

Table 6.1 provides the stability results of BC and different PUMA variations. The data shows that every variation of PUMA successfully enforces stability across the dataset, generating no unsuccessful trajectories. Compared to BC, which has



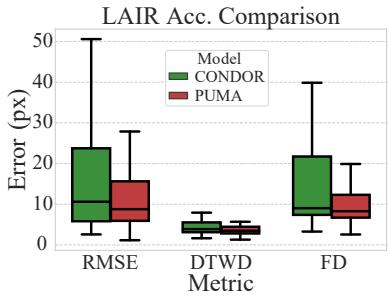
(a) S shape PUMA. **(b)** S shape CONDOR.

(c) S shape PUMA with ℓ_∂. **(d)** Positive invariance comparison.

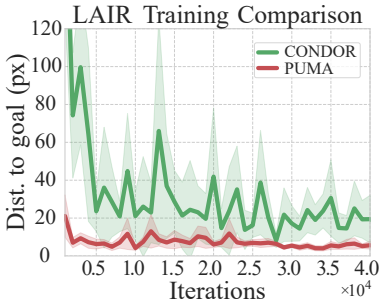
Figure 6.8.: Positive invariance evaluation. In figures (a)-(c) white curves represent demonstrations. Red curves represent learned motions when starting from the same initial conditions as the demonstrations. The arrows indicate the vector field of the learned dynamical system.

a 36.4653% rate of unsuccessful trajectories, the benefit of the proposed loss in enforcing stability becomes clear.

State-of-the-art comparison Fig. 6.7b presents an accuracy comparison of PUMA with other state-of-the-art methods, namely: 1) Control Lyapunov Function-based Dynamic Movements (CLF-DM) using Gaussian Mixture Regression (GMR) [164], 2) Stable Dynamical System learning using Euclideanizing Flows (SDS-EF) [159], and 3) CONDOR. The results for PUMA correspond to the best-performing variation in this dataset, i.e., spherical distance with ℓ_∂. We observe that PUMA achieves competitive results, demonstrating similar performance in DTWD and FD



(a) Accuracy comparsion.



(b) Training stability comparison measured as the average distance to the goal.

Figure 6.9.: LAIR dataset PUMA / CONDOR comparison.

to CONDOR and SDS-EF, and slightly superior performance under RMSE.

Boundary loss Fig. 6.8 demonstrates the effect of the boundary loss ℓ_∂ in PUMA. Figures 6.8a and 6.8b present an example of the qualitative performance of PUMA and CONDOR when no boundary loss is applied. In the top-left region of these images, PUMA exhibits non-smooth trajectories at the boundary, which abruptly change direction due to the applied saturation (see Sec. 6.4.4). Conversely, CONDOR learns a smoother trajectory in this region. This feature in PUMA vanishes when ℓ_∂ is applied, as depicted in Fig. 6.8c. Finally, Fig. 6.8d provides a quantitative evaluation of the boundary loss, confirming our qualitative observations.

In this work, this loss is **only relevant for Euclidean state spaces** because we do not introduce boundaries in the state space when controlling orientation in $\mathcal{S}^3$.

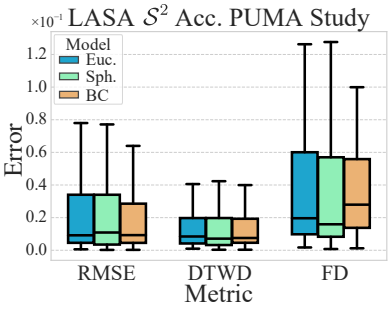
LAIR

Contrary to the LASA dataset, the LAIR dataset [156] is specifically designed to evaluate second-order motions, i.e., those that map current position and velocity to acceleration. This dataset comprises 10 human handwriting motions, with the state being 4-dimensional, encompassing a 2-dimensional position and velocity. The dataset's shapes contain multiple position intersections, intentionally designed to necessitate the use of at least second-order systems for their successful modeling.

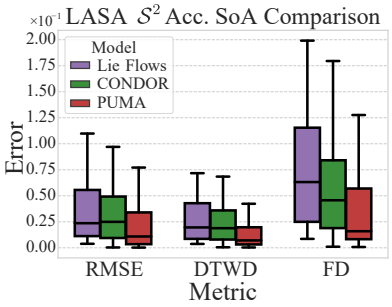
CONDOR / PUMA comparison The state-of-the-art methods outlined in Sec. 6.5.1, excluding CONDOR, do not address the challenge of learning stable second-order systems. This can be attributed to the greater difficulty inherent in learning such systems compared to first-order systems. As a result, we compare the performance of PUMA with that of CONDOR. Fig. 6.9a illustrates the comparative accuracy of both methods, with PUMA surpassing CONDOR on all

Table 6.2.: Percentage of unsuccessful trajectories over the LAIR dataset ($L = 2500$, $P = 2500$, $\epsilon = 10$px).

Behavioral Cloning	PUMA (Euc.)	PUMA (Euc. + ℓ_∂)	PUMA (Sph.)	PUMA (Euc.+ ℓ_∂)
14.1160%	**0.0000%**	**0.0000%**	**0.0000%**	**0.0000%**



(a) Comparison of different variations of PUMA.



(b) Comparison of PUMA with other state-of-the-art (SoA) methods.

Figure 6.10.: Accuracy study in LASA $\mathcal{S}^2$ dataset.

metrics. PUMA's greater learning flexibility allows it to converge to solutions beyond CONDOR's capabilities. Moreover, this enhanced flexibility, as depicted in Fig. 6.9b, endows PUMA with a more stable learning process. In practice, this makes a significant difference, as motions with unsuccessful trajectories must be discarded when considering the system's stability. Therefore, greater learning stability results in a larger set of motions without unsuccessful trajectories, providing more alternatives of successfully learned systems to choose from.

Stability Table 6.2 presents the results of stability analysis for PUMA on the LAIR dataset. Similar to the findings with the LASA dataset, every variation of PUMA achieves a 0% rate of unsuccessful trajectories. This validates PUMA's ability to learn stable second-order motions.

6.5.2. Non-Euclidean dataset: LASA $\mathcal{S}^2$

The LASA $\mathcal{S}^2$ dataset [204] comprises 24 motions, each with 3 demonstrations. Unlike the LASA dataset, the LASA $\mathcal{S}^2$ dataset represents these motions in spherical geometry, as indicated by its name, in $\mathcal{S}^2$. The sphere, where the motions evolve, is structured similarly to unit quaternions, though with one less dimension. Essentially, we have a 3-dimensional Euclidean space where vectors are constrained to have a unit norm. As a result, the state space is a 2-dimensional spherical manifold embedded in this 3-dimensional space. This setup allows us to examine the performance of PUMA in a manifold with similar attributes to those

Table 6.3.: Percentage of unsuccessful trajectories over the LASA $\mathcal{S}^2$ dataset ($L = 2500$, $P = 2500$, $\epsilon = 0.06$).

Behavioral Cloning	CONDOR	PUMA (Euc.)	PUMA (Sph.)
15.9217%	7.3133%	**0.0000%**	**0.0000%**

of unit quaternions. However, the more straightforward visualization of this setup facilitates a more intuitive analysis of PUMA's performance.

Accuracy

Fig. 6.10a presents the performance of two variations of PUMA, namely when using Euclidean and spherical distance functions. Moreover, BC is included, serving as a lower-bound reference. As explained in Sec. 6.5.1, the boundary loss ℓ_∂ does not apply in this case and is, therefore, not evaluated. We can observe that the accuracy performance of both variations of PUMA is very similar, and BC performs slightly better than both of them.

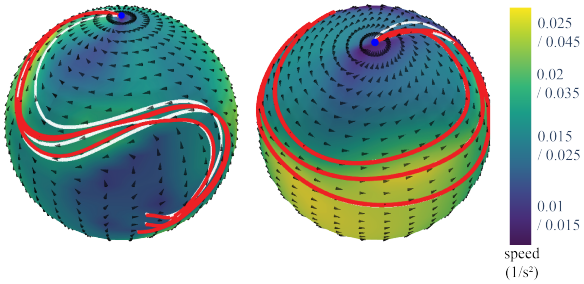
State-of-the-art Comparison Fig. 6.10b depicts a comparison between three methods: 1) CONDOR, 2) PUMA, and 3) *Lie Flows* [204], a method developed for learning stable motions in non-Euclidean manifolds using tools from Lie Theory. PUMA outperforms both CONDOR and Lie Flows in terms of accuracy. Moreover, in contrast to CONDOR, PUMA also successfully ensured stability in this dataset.

Stability

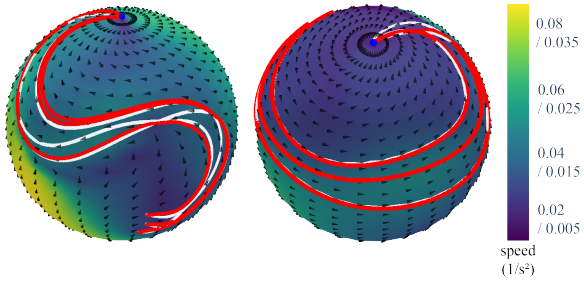
Table 6.3 depicts the stability results for the LASA $\mathcal{S}^2$ dataset. In this case, we also conducted a stability analysis of CONDOR, which, as discussed in Sec. 6.2, should not be capable of ensuring stability in non-Euclidean state spaces. This assertion is confirmed by the data in Table 6.3, which shows that CONDOR yields 7.3133% of unsuccessful trajectories. When compared to BC, which has a 15.9217% rate of unsuccessful trajectories, it can be inferred that CONDOR reduces the number of unsuccessful trajectories in non-Euclidean state spaces. However, its performance is still far from satisfactory. In contrast, both versions of PUMA achieve 0% of unsuccessful trajectories, marking a significant improvement. Moreover, this validates that the Euclidean distance is effective for enforcing stability in spheres, as it can induce spherical metrics, such as the chordal distance, in lower-dimensional manifolds (see Appendix C.2).

Qualitative Analysis

Fig. 6.11 illustrates motions learned with PUMA in $\mathcal{S}^2$ using both spherical and Euclidean distances. Within the observed region of the sphere, the vector field exhibits no spurious attractors. Furthermore, when initiated from the same conditions as the demonstrations, the trajectories in both cases show an accurate reproduction of the motions.



(a) Results using the great-circle distance.



(b) Results using the Euclidean (chordal) distance.

Figure 6.11.: Motions learned with PUMA in $\mathcal{S}^2$. White curves represent demonstrations. Red curves represent learned motions when starting from the same initial conditions as the demonstrations. The arrows indicate the vector field of the learned dynamical system.

6.5.3. Real-world Experiments

We validate our method in two real-world setups, using two different robots. Both robots have six degrees of freedom, with their end-effector pose represented in $\mathbb{R}^3 \times \mathcal{S}^3$ and are guided using PUMA. In these experiments, $f_\theta^{\mathcal{T}}$ is employed to generate velocity references in real-time, which are then sent to the low-level controllers responsible for tracking these references in the robots. Note that throughout the experiments, we operated under the assumption that the target states of the robots were already known.

Greenhouse

This experiment was conducted in a greenhouse, where a robot was trained to reach a black marker on a tomato plant (see Fig. 6.12). This marker, simulating a plant's peduncle, represented the target state the robot must reach to harvest the plant. Here, the marker allowed us to test the method without causing harm to the plant. This task presents significant challenges, as it requires the robot to perform precise position and orientation control while considering the plant's complex geometry, which is non-trivial to model. In a practical setting, a motion

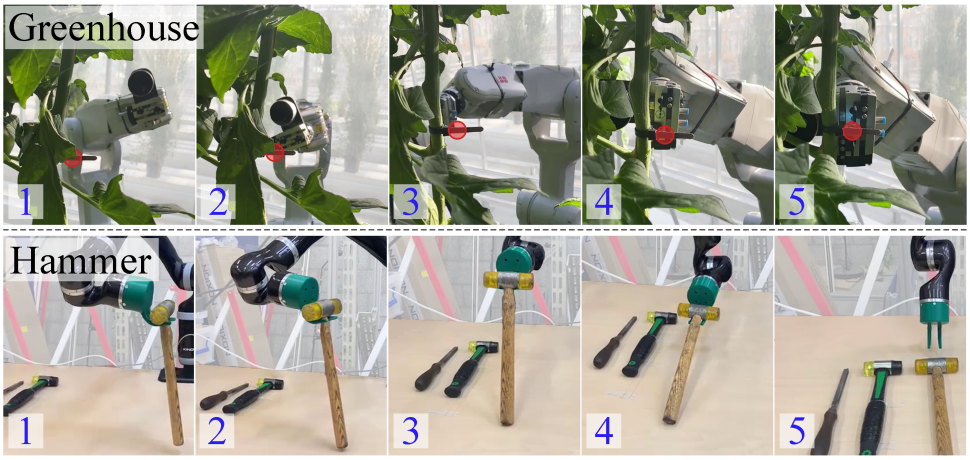


Figure 6.12.: Two sequences of frames, each depicting a robot performing a task: 1) operating in a greenhouse, and 2) placing a hammer. The red circle is used to highlight a target marker in the greenhouse experiment.

library should be developed to account for the variability across different plants and targets. This library could then be used to select or combine suitable motions to create an appropriate movement strategy considering the plant's characteristics. This experiment represents a preliminary step towards achieving this objective.

We utilized an *ABB IRB 1200* robot equipped with the *Externally Guided Motion*[7] module, allowing real-time joint velocity control. As the motion takes place in the end-effector space, the desired velocity at each time step was calculated using $f_\theta^{\mathcal{T}}$, and subsequently mapped to the joint space. This conversion was accomplished using the inverse kinematics module with joint limits from the *Robotics Toolbox* [227]. Similarly, we translated the robot's estimated joint state into the end-effector space using the forward kinematics module from the same toolbox. The same approach was taken to collect the demonstrations of this task directly in the end-effector space, where a space mouse was employed to teleoperate the robot.

Fig. 6.13 presents the demonstrations and simulations of motions conducted in this experiment. It can be observed that five demonstrations were provided. Note that these demonstrations do not have the same end time; only the final state needs to be consistent. Over a span of 70 seconds, all simulated trajectories visibly converge to the goal. For more details, the reader is referred to the attached video.

Hammer

The second experiment involves a robot accurately positioning a hammer next to other tools on a table. This task requires precise control of the robot's position and

[7]Code: https://github.com/ros-industrial/abb_robot_driver.

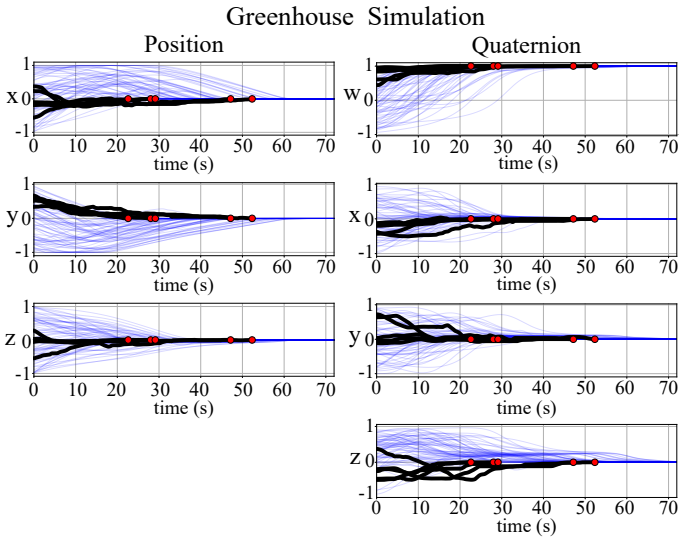


Figure 6.13.: Simulated trajectories, as a function of time, of the greenhouse experiment. Blue trajectories correspond to evaluations of the model under different initial conditions and the black trajectory corresponds to the demonstration. The red point is the goal.

orientation, as shown in Fig. 6.12. We are interested in controlling the hammer's movement, so we assume that the state of the hammer directly relates to the state of the robot's end effector. This assumption is necessary because the employed low-level controller acts on the robot's end-effector state, and we ensure stability in this state space. Given that the hammer is attached to the robot through a double hook, this assumption is valid as long as the hammer's head is securely held. This assumption breaks down towards the end of the motion when the robot places the hammer on the table. However, since the final states of both the hammer and the robot are similar, we observed that ensuring stability in the end-effector's motion effectively guides the hammer's motion towards its goal state, as can be seen in Fig. 6.12 and in the attached video.

We used the *Kinova Gen2 Ultra lightweight robot* arm with a double hook replacing its default gripper. The control commands obtained from $f_\theta^{\mathcal{T}}$ were directly sent to a Cartesian space controller from Kinova[8]. The demonstrations were collected through kinesthetic teaching, i.e., physically moving the robot along desired paths. The simulated performance in this task was similar to that in the greenhouse, given that the state space was identical in both cases. We refer the reader to the attached video for examples of the robot executing this task from various initial conditions.

[8]Kinova's controllers: https://github.com/Kinovarobotics/kinova-ros

6.6. Conclusions

We introduced a novel approach for learning stable robotic motions in both Euclidean and non-Euclidean state spaces. To achieve this, we introduced a new loss function based on the triplet loss from the deep metric learning literature. We validated this loss both theoretically and experimentally.

Our approach, PUMA, demonstrated state-of-the-art performance in every experiment, both in terms of accuracy and stability. It was validated using datasets where the dynamical system evolution was simulated, as well as real robotic platforms where it was employed to provide control commands to low-level controllers.

Compared to previous work, PUMA offers not only an improvement in addressing non-Euclidean state spaces but also increased flexibility by reducing restrictions in the latent space of the DNN, leading to generally better performance. More specifically, in previous work, the latent dynamical system is constrained to evolve along straight lines towards the goal. In contrast, PUMA allows the latent dynamical system to converge towards a broader range of stable dynamical systems, since its loss function only enforces the latent dynamical system to reduce the distance towards the goal. This feature expands the set of feasible solutions available to the DNN during optimization, thereby enhancing the model's adaptability.

Lastly, while this paper's findings are promising, there are also limitations that can be addressed in future works. First, further exploration of the scalability properties of PUMA is needed, as the largest DNN input employed in this paper had 7 dimensions. Second, we have so far focused on learning independent motion primitives for specific tasks. A relevant line of research would be integrating this model into a larger framework where multiple primitives are learned and combined together. Third, a topic that we did not address in this work is obstacle avoidance. Recent techniques, such as Geometric Fabrics [228], exploit dynamical systems to represent motions that achieve full-body obstacle avoidance, which can be integrated with dynamical systems learned in the end-effector space of the robot. Consequently, an exciting avenue for future work is exploring the combination of these methods with PUMA. Finally, another interesting research direction is studying the integration of these approaches with the low-level control of the robots. Currently, it is assumed that the transitions requested by PUMA can be tracked by the controllers of the robot. However, this is not always the case, and it would be beneficial to incorporate this information into the learning framework.

7

Conclusions and Future Work

7.1. Conclusions

Our research was driven by the significant potential of Imitation Learning (IL) to broaden robot deployment in real-world applications. This potential comes from IL's ability to enable robots to execute complex behaviors, while remaining adaptable and reactive. We recognize these capabilities as crucial elements currently missing in many real-world robotic applications. The importance of these features lies in the fact that most advanced robotic solutions are either too costly or impractical in numerous scenarios, leading to the predominance of simpler, non-reactive, systems. Therefore, by developing easily customizable robotic solutions, we can facilitate rapid adaptation to diverse environments, such as those encountered in the agro-food industry or household environments. Importantly, these solutions can be adapted by non-experts, eliminating the need for highly trained engineers to personalize systems for each individual scenario. This approach significantly reduces costs and expands feasibility, making advanced robotic systems accessible in contexts like medium-sized factories or homes, which would otherwise find them impractical.

As a result, for advancing towards robots that learn via imitation, we focused on two fundamental limitations of current IL approaches: **reliability** and **data efficiency**. These are critical obstacles that have previously limited the wider application of IL in practical robotics. Therefore, through this work, we have introduced multiple methodologies to address them, demonstrating IL's impressive potential in transferring complex human behaviors to robots.

To accomplish this, we highlighted that issues in IL related to reliability and data efficiency stem from a poor representation in the training data of the state distribution that a robot experiences during deployment. This translates into robots needing to make decisions in situations that have no similar counterparts in the data the robot used to learn its behavior, resulting in arbitrary actions and potential failure. One reason this scenario occurs is that a robot can make small mistakes that compound, thus drifting away from the support of the training data during the execution of a task, a phenomenon known as *covariate shift*. Another reason is the impractical approximation of this distribution due to factors

such as: 1) limited access to human data, leading to a *sparsely represented* state distribution, or 2) *long-tailed* learning problems. Long-tailed problems involve numerous unlikely situations; while not all are feasible to cover, some are bound to occur in practice. As a result, these sources of IL limitations provide interesting challenges that motivated the work introduced in this dissertation. We explored two key concepts for tackling these distributional challenges in IL: **online learning**, and **inductive bias**.

Firstly, we emphasized that covariate shift is not intrinsic to IL itself, but rather a consequence of the problem formulation used in *offline* IL. Thus, changing this formulation can effectively address the issue. In this context, *online* learning methods, also known as *Interactive Imitation Learning* (IIL), which involve real-time human-robot interaction, emerge as a promising approach, naturally mitigating covariate shift. However, despite its potential, IIL remains relatively underexplored in robotics. Consequently, our research addresses this gap by examining and contrasting various IIL methodologies, highlighting their strengths and limitations, and resolving ambiguities in critical areas such as on-policy and off-policy learning (Chapter 2). Furthermore, we introduced and studied a novel methodology for learning from human relative corrections, which integrates techniques from the field of state representation learning (Chapter 3). We observed that state representation learning is crucial when learning from relative corrections, especially when the success of a problem depends on properly incorporating temporal information. Our results indicated that, in such scenarios, this could be the decisive factor between solving a task and failing. Additionally, a secondary observation emerged from a simple simulated problem: the *inverted pendulum*. In this problem, we compared policies derived from multiple human teachers using two types of feedback: absolute corrections (i.e., demonstrations through interventions) and relative corrections (i.e., directional guidance). The rapid dynamics of this problem make providing demonstrations challenging; however, offering directional guidance is relatively simple. Our experiments showed that while only half of the human teachers could teach a successful policy using absolute corrections, all were successful with relative corrections.

Secondly, in Chapter 4, we investigated the incorporation of inductive bias within our learning frameworks. Recall that inductive bias introduces prior knowledge into the robot's policy, effectively narrowing down its potential solutions and guiding it toward desired behaviors. This not only makes frameworks more data efficient but also enables the establishment of guarantees, such as collision avoidance and goal convergence. In the context of IIL, we applied this concept to learning driving behaviors. More specifically, we incorporated a Model Predictive Controller (MPC) into an IIL framework. Consequently, the MPC, through well-established cost function terms, inherently provided capabilities to the robot, like collision avoidance and path tracking. Simultaneously, more complex behavioral aspects, such as determining the vehicle's optimal speed in situations containing traffic signals, other vehicles, or crossing pedestrians, were learned using interactive data. This method demonstrated the effective fusion of interactive machine learning and control methodologies, resulting in a flexible learning framework that

is both data-efficient and reliable. As expected, our results showed that our approach achieved better performance in terms of the number of collisions and deadlocks, compared to its optimization-based-only and data-driven-only variants. Specifically, our method resulted in 7.03 times fewer collisions and 1.31 times fewer deadlocks than its MPC-only variant, and 5.24 times fewer collisions and 1.38 times fewer deadlocks than its data-driven-only variant. Interestingly, this performance was achieved with only 2 hours of training time, a characteristic that distinctly differentiates it from other methods in the literature, which report requiring $100 - 200$ times more human time. We believe that a defining factor in this difference is the employed IL approach: unlike other works that used offline demonstrations, our method is based on collecting demonstrations online. This shows the potential of IIL methods to make problems, which may seem intractable in terms of data, manageable.

Lastly, in Chapters 5 and 6, we fully shifted our focus to inductive bias, moving beyond IIL. We continued utilizing control theory tools, primarily focusing on theoretical methods for analyzing the behavior of dynamical systems. Utilizing these methods, we designed novel loss functions for Deep Neural Networks (DNNs), aimed at incorporating inductive bias to enforce global asymptotic stability in motion primitives modeled as dynamical systems. This is pertinent in situations where the robot's motion needs to consistently converge to a specific goal state, such as in the acts of grabbing, swiping, or hanging. In these scenarios, by ensuring that the goal state is a globally asymptotically stable equilibrium of the dynamical system, convergence to this point is guaranteed, regardless of the robot's initial state. To achieve this, we introduced specific conditions in the DNNs' latent space to impose this stability. Utilizing deep metric learning tools, we then developed loss functions tailored to optimize the DNN to meet these conditions. Our research culminated in two methodologies: CONDOR (Chapter 5) and PUMA (Chapter 6). CONDOR, as our initial approach, exhibited exceptional performance and established the groundwork for further advancements. Building on these principles, PUMA introduced a more versatile loss function, not only offering better optimization properties but also extending stability enforcement to non-Euclidean state spaces. Such an extension is crucial in robotics, particularly for tasks like controlling a robot's orientation. Both introduced methods, depending on the evaluation metric used, demonstrated similar or superior accuracy compared to state-of-the-art approaches that enforce stability. This can be attributed to the fact that CONDOR and PUMA impose stability through the optimization process of the DNN, in contrast to other methods that achieve this by constraining the structure of their function approximators. Such constraints can impose excessive limitations on the range of solutions these models can reach, resulting in reduced learning capabilities and, consequently, suboptimal IL solutions. However, it is challenging to attribute these results solely to these constraints, as other factors, particularly hyperparameters, also influence these outcomes. Furthermore, methods constraining the structure of their learning models may also face scalability limitations, as encoding these restrictions in more complex models can be challenging. This is a key distinction with our approaches,

which are highly flexible in this respect, as they impose the stability behavior via optimization and require minimal (CONDOR) or no (PUMA) intervention in the DNN's architecture. Our results provide evidence in this direction. For example, CONDOR was easily extended to learn multiple stable motions within a single DNN and to interpolate between them. Furthermore, both CONDOR and PUMA successfully imposed stability in motions modeled as second-order systems, a feat not yet demonstrated by other DNN-based approaches. However, we believe there is still much to be explored in this regard, providing a foundation for exciting future research.

Throughout this dissertation, the developed methodologies are supported by robust theoretical principles, and have been rigorously tested using simulations, and, in most cases, real-world systems. These elements are indispensable in advancing the field of robotics. On one hand, given the expectation for robots to operate in environments where safety and performance guarantees are crucial, a deep understanding and predictability of these systems is essential. On the other hand, the true viability of our methodologies depends on their real-world applicability, necessitating empirical validation. To this end, we have tested our methodologies using multiple systems: a 3-DoF manipulator, an inverted pendulum, a 7-DoF KUKA iiwa manipulator, a 6-DoF Kinova Gen2 manipulator, and a 6-DoF IRB 1200 manipulator. These platforms have been fundamental in providing insights into real-world applications. It is important to note, however, that most current laboratory settings in robotics may not always capture the full complexities of real-world scenarios. This observation presents a significant challenge in the field, providing a valuable opportunity for further work. Such efforts are essential in advancing our robotic technologies toward practical realization.

7.2. Future Work

We have taken significant steps to address the challenges in IL. However, to fully realize the potential of IL, several areas require further exploration. Some of these challenges are directly related to the methodologies presented in this study, while others pertain to broader aspects of IL. We outline below a some of these critical challenges:

- **Human-robot interfaces:** A key challenge in IL that is often overlooked is the interface used for collecting human data. In some cases, the choice of interface can be the deciding factor between successfully learning high-performing policies and failing to learn anything at all. For example, kinesthetic teaching is a common method, involving physically manipulating the robot. However, this approach becomes increasingly challenging, and eventually infeasible, if the robot is too large or complex for the human to control due to multiple degrees of freedom. Similarly, teleoperation can be useful, but tasks demanding high precision may require more sophisticated devices. These devices should allow for easy control of both the robot's position and orientation simultaneously. Moreover, if certain areas of the workspace are inaccessible or occluded from the human operator's view,

adding onboard cameras to the robot becomes necessary. Moreover, as highlighted in [229], haptic feedback can also be essential for humans to provide accurate training data. Therefore, these examples illustrate how interfaces that are commonly used can easily become limited, requiring significant enhancements to effectively collect human data.

- **Human mistakes:** Another essential feature for IL to become practical is its robustness against human mistakes when providing data. This can be addressed either by preprocessing the data or by developing machine learning algorithms that naturally counteract this issue. For instance, if humans provide demonstrations with a Gaussian error centered on the correct demonstrations, many methods are already robust to these errors, as they assume a Gaussian policy and average out these mistakes. However, for instance, in situations involving outlier mistakes, this data would distort such policies. Hence, we need methodologies that consider these scenarios and can address them effectively.

- **Multimodal policies**: Multiple strategies often exist as viable policy solutions in the learning problem. Consequently, human data frequently presents a variety of them for identical scenarios, highlighting the importance of developing methods capable of handling these multimodalities. Efforts to tackle this issue have varied, employing different strategies. One approach involves forcing the learned policy to converge to a single solution [39, 230]. Another strategy filters the dataset to ensure only one modality is represented in it [153]. More recently, some research has introduced methodologies enabling the learning of multiple action options for the same state [231, 232]. While these approaches have shown promise in addressing this issue, they are relatively recent developments. Therefore, there remains considerable scope for further research to refine and enhance these methods.

- **Robot's model as inductive bias:** Frequently, we have access to the robot's kinematic and even dynamic behavior. Nevertheless, this information is rarely explicitly included in the policy optimization process. Including this valuable information would increase the data efficiency of our learning method. Moreover, it would also guide the optimization process towards learning behaviors that are achievable by the robot. This is particularly clear when modeling policies as dynamical systems. In these cases, policies generate desired state transitions which, if optimized naively, might be unfeasible for the robot. Therefore, our methods would greatly benefit from incorporating the robot's physical constraints into the learning process. Although there are some efforts in this direction, e.g., [233], there is still a considerable amount of work to be done.

- **Guarantees in high-dimensional spaces:** Lastly, a critical challenge that remains unaddressed is integrating guarantees into policies dealing with high-dimensional, and potentially multimodal, state spaces. This is motivated by the last decade's significant advancements in machine learning,

particularly with DNNs. These networks are noted for their ability to process high-dimensional data from multiple sources effectively. However, the majority of efforts to introduce guarantees into learned policies have focused on relatively low-dimensional state spaces. This approach is understandable, as guarantees often require a well-defined state space where analyzing the robot's behavior is manageable. The challenge escalates when considering inputs like raw pixels from images or point clouds, where analyzing the system's behavior in these unstructured spaces is daunting. Nevertheless, if our goal is to develop flexible robots that can interpret information from various sources and provide guarantees, we must tackle this issue. Without addressing it, we risk having two parallel streams of development, each making progress, but neither offering realistic solutions for the complex problems we face in the real world.

These challenges provide exciting avenues for future research, offering the potential to enhance the learning capabilities of our robots via IL, creating easily customizable, adaptable, and reliable systems.

Appendices

A

Additional Experimental Details (Chapter 3)

The metrics used for the comparisons in the experiments are the final policy's performance achieved and the speed of convergence, which is very relevant when dealing with real systems and human teachers. In every learning curve, a mean of the return obtained over 20 repetitions for every experiment is presented, along with the maximum and minimum values of these distributions. The performance was measured in the simulated environments with the objective functions included in OpenAI Gym. For the real pendulum, a similar function to the one implemented in OpenAI Gym was used, while for the task with the robot arm, the success rate was used.

For the experiments with simulated teachers, high-performance policies trained by expert teachers were used as oracles for providing the corrections. These policies have performances at the level of the state of the art (performance plotted in the figures). In Deep COACH, the corrections h are the sign of the relative change advised by the teacher. Therefore, to compute these binary corrections, the simulated teacher computes $h = \text{sign}(a_{teacher} - a_{agent})$ for each of the dimensions composing the action vector, wherein $a_{teacher}$ is the action computed by the policy of the teacher and a_{agent} is the action computed by the learning policy, as it has been implemented in [21, 96]. The frequency of the corrections given by the simulated human teacher is controlled with a probability that is reduced with time.

For the experiments in which actual human teachers participated in correcting policies, the corrections were provided through a keyboard. For each action dimension, two keys were designated, so the user could advise an increase or decrease, in each of the axes.

A.1. Neural Network Architecture

Fig. 3.3 in Chapter 3 shows the structure of the proposed NN architecture. The hyperparameters of each layer are shown in Tables A.1 and A.2.

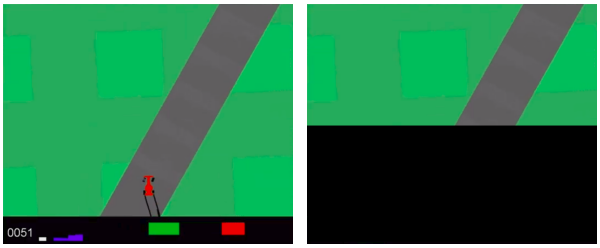
A

Table A.1.: Hyperparameters of convolutional and deconvolutional layers.

Layer	Activation	Filters	Filter size	Stride
C1	ReLU	16	3×3	2
C2	ReLU	8	3×3	2
C3	sigmoid	4	3×3	2
DC1	ReLU	8	3×3	2
DC2	ReLU	16	3×3	2
DC3	sigmoid	1	3×3	2

Table A.2.: Hyperparameters of fully-connected and recurrent layers.

Layer	Activation	N° neurons
FC1	tanh	256
FC2	tanh	256
FC3	tanh	1000
FC4	tanh	256
FC5	ReLU	1000
FC6	ReLU	1000
FC7	tanh	Task action dimension
R1	LSTM activations	$h^{LSTM} = 150$



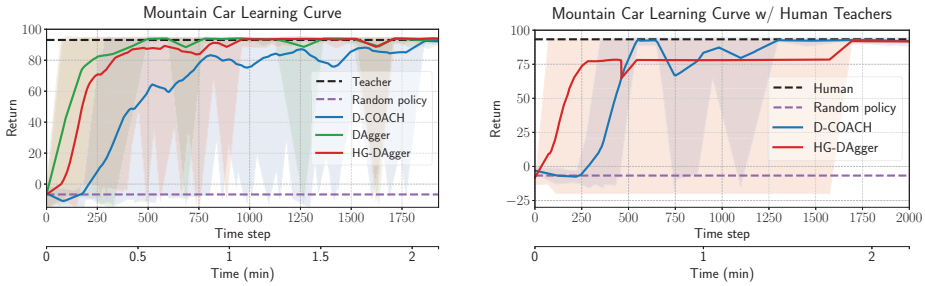
(a) Original frame **(b)** Occluded frame

Figure A.1.: Original frame of the Car Racing environment, on the left. The occluded frame used as observation in the network, on the right.

A.2. Ablation Study - Environment

All the results obtained in this study are shown in Chapter 3. The comparisons were carried out only with the Car Racing problem of OpenAI Gym. In this environment, the agent has three action dimensions which are: steering, acceleration, and brake.

As mentioned in Chapter 3, the bottom half of the frame is occluded in order to force the agent to make decisions based on past observations. An example of the occluded frame is shown in Fig. A.1, the current position of the car on the road is not observed by the learning policy; however, the entire frame is observed



(a) D-COACH and DAgger comparison in the Mountain Car problem using a simulated teacher.

(b) Mountain Car learning curve with human teachers

Figure A.2.: Comparative analysis in Mountain Car problem

by the policy used as a simulated teacher. Therefore, the corrections are based on appropriate actions with respect to the real state of the environment.

A.3. Simulated Tasks with Simulated Teachers

The Mountain Car and the Swing-Up Pendulum environments originally provide, at each time step, their low-dimensional explicit state. These are the position and velocity of the car in the x axis, and the angle and angular velocity of the pendulum, respectively. In order to obtain a high-dimensional observation (raw image) we have modified the source code, such that the environment returns an array with the rendered RGB frame.

Mountain Car

The action of the agent is the force applied in order to move the car. It is known that the optimal solution for this task is a bang-bang controller (using the extreme actions -1 or 1). Therefore, the correct actions given by the oracle are very different from the initial policy in the whole state space. This makes it easier to perform abrupt changes when updating the policy for DAgger-like agents than for Deep COACH because the latter performs smaller steps based on incremental and relative corrections (corrections are in the direction of the oracle's action, but not directly the actual action).

As it can be seen in Fig. A.2a, as expected, the learning convergence is faster for DAgger, followed by HG-DAgger, whereas Deep COACH is the slowest. DAgger converges faster than its modified version, HG-DAgger, because the former trains the policy with corrections provided by the oracle during every time step, which is most of the time not feasible when teaching with human users. All agents reach the oracle's performance at the end of the learning process.

A

A.4. Simulated Tasks with Human Teachers

The validation was executed with 8 participants between 20 and 30 years. In the experiments, the users observed the desired performance of the agent, received instructions on how to interact with each kind of agent, and had the chance to practice with the learning agent before recording results, in order to get used to the role of teacher. The users interact with the learning agent using a keyboard. Users correct the policies until they consider they cannot improve them anymore, or until a maximum duration of the training session of 500 seconds (8.33 minutes). The episodes of the tasks had a duration of 20 seconds, which for Mountain Car means the maximum duration if the goal is not reached, whereas for the pendulum this is constant. The duration of the time steps is 0.05 s.

Mountain Car

This task is very simple, since users understand properly how to teleoperate the agent, therefore, we could state that for this task, the teachers are always experts.

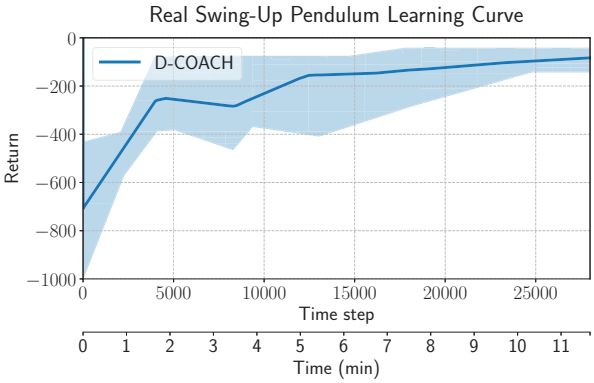
Results in Fig. A.2b are similar to the ones observed in Fig. A.2a in Sec. A.3, wherein at the beginning HG-DAgger improves way faster. However, in this case, HG-DAgger gets stuck and is outperformed by Deep COACH after 20 seconds. This happens as, despite the fact that the teachers are considered experts, they could sometimes provide mistaken or ambiguous corrections. These inconsistencies remain permanently in the database, so it is hard for the user to fix the generated error.

A.5. Validation on Physical Systems with Human Teachers

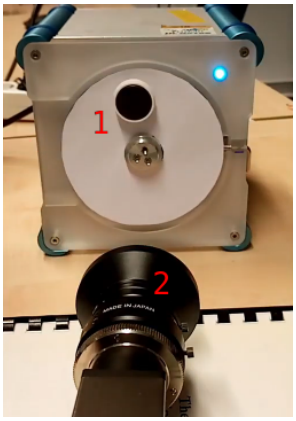
A.5.1. Real Swing-Up Pendulum

This system is similar to the one used previously from OpenAI Gym, although in this set-up the dynamics are even much faster, and the actions are voltages instead of torques. In this environment, a camera is set in front of the pendulum to obtain observations similar to the simulated environment, as it is shown in Fig. A.3b, wherein the camera is in the bottom, aligned with the pendulum. The observation was down-sampled to 32×32 pixels images, while for all the other tasks the down-sampling was to 64×64 pixels. An example of the actual observation of the camera is in Fig. A.4, where it is possible to see the input of the NN (down-sampled image), along with the prediction of the observation in $t + 1$, with a model that has been trained with the proposed architecture. In this example, it is interesting to observe that the pendulum was rotating clockwise, therefore the prediction shows the weight of the pendulum in a lower position.

Fig. A.3a shows the learning curve of 10 runs. On average, the users manage to teach a policy able to balance after 29 episodes, and they keep on improving the policy during the subsequent episodes. Preliminary tests showed that the sampling time needs to be reduced to 0.025 s to be able to control the system.

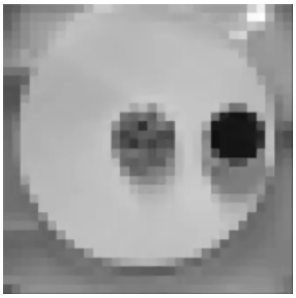


(a) Real Swing-Up Pendulum learning curve.

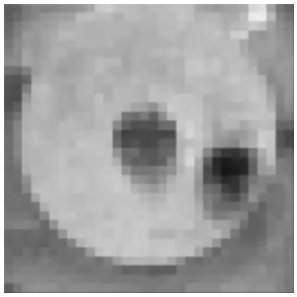


(b) Real Swing-Up Pendulum Setup. 1) Pendulum; 2) RGB camera.

Figure A.3.: Analysis and setup of Real Swing-Up Pendulum



(a) Network input

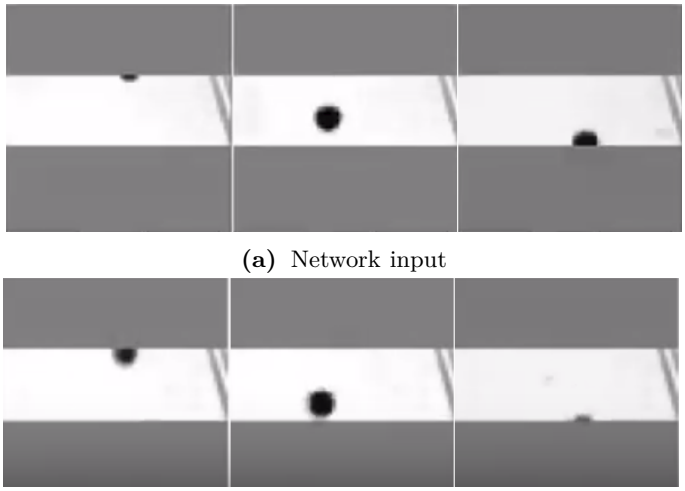


(b) Observation prediction

Figure A.4.: Downsampled observation of the camera, which is the input of the Network, on the left. Next observation prediction, on the right.

A.5.2. Orange selector with a robot arm

As mentioned in Chapter 3, this problem is composed of two sub-tasks, that are sequentially trained. In the experiments, at the beginning when learning the **orange selection**, the teachers corrected the policy only in order to make the robot move the end-effector over the horizontal plane. Therefore, the transition model along with a controller in charge of intercepting the oranges with the end-effector is learned in the first stage. Then, in order to learn the second sub-task (**pear rejection**) composing the problem, the already learned transition model is reused in the training process of a second controller that either moves the end-effector close to the belt or far when a pear is detected. These two controllers



(a) Network input

(b) Observation prediction

Figure A.5.: Examples of images in the input of the Network, on the top. Observation predicted for the next time step by the learned transition model, on the bottom.

work in parallel since their actions are over different joints.

In the associated video it is shown examples of the actual images that go to the input of the network, along with the prediction of the next observation. However, due to the velocity of the video, it is not easy to see that the predicted oranges are slightly shifted. In Fig. A.5 are shown examples of three different oranges crossing the field of view of the camera. The examples are input-output pairs of the transition model. The oranges moved by the belt are observed by the camera while crossing from the top to the bottom of the field of view. It could be seen that the predicted oranges are in a lower position with respect to the position observed in the input, i.e. the model learns to predict the movement of the belt.

B

Additional Proofs and Experimental Details (Chapter 5)

B.1. Approximating a Diffeomorphism

For achieving stable motions, CONDOR optimizes ℓ_{stable}. This loss is designed to enforce the conditions of Theorem 1 in the NN employed to represent the dynamical system $f_\theta^{\mathcal{T}}$. In this section, we show that as a consequence of this, ψ_θ approximates a diffeomorphism when $\mathcal{T}$ is a Euclidean space.

Definition 4 (Diffeomorphism)**.** A mapping between two manifolds $\psi_\theta : \mathcal{T} \to \mathcal{L}$ is called a diffeomorphism if it is differentiable and bijective.

In general, we can consider Neural Networks to be differentiable, since most of the employed activation functions are differentiable. However, there are some exceptions to this, as it is for the case of the ReLU activation [234]. In practice, these exceptions are non-differentiable at a small number of points and they have right and left derivatives, so they do not present many issues when computing their gradients. However, strictly speaking, for such cases, our method would make ψ_θ converge to a *homeomorphism* instead. Differently to a diffeomorphism, a homeomorphism only requires the mapping to be continuous, but not differentiable. Nevertheless, for simplicity, we will assume that ψ_θ is differentiable.

Then, we need to study if ψ_θ converges to a bijective function to conclude that it approximates a diffeomorphism.

Definition 5 (Bijective function)**.** A function is bijective if it is injective and surjective.

Definition 6 (Injective function)**.** A function is injective if every distinct element of its domain maps to a distinct element, i.e., ψ_θ is injective if $\psi_\theta(x_a) = \psi_\theta(x_b) \Rightarrow x_a = x_b, \forall x \in \mathcal{T}$.

Definition 7 (Surjective function)**.** A function is surjective if every element of the function's codomain is the image of at least one element of its domain, i.e., $\forall y \in \mathcal{L}, \exists x \in \mathcal{T}$ such that $y = \psi_\theta(x)$.

From these definitions, it is clear that the surjectivity of ψ_θ is straightforward to show, since it depends on how its codomain is defined. In this work, we define the codomain of ψ_θ as $\mathcal{L}$, which is the manifold resulting from the image of ψ_θ. In other words, the codomain $\mathcal{L}$ is a set that only contains the outputs of ψ_θ produced from $\mathcal{T}$. In such cases, a function is surjective, since its codomain and image are equal, which ensures that $\forall y \in \mathcal{L}, \exists x \in \mathcal{T}$ such that $y = \psi_\theta(x)$ (Definition 7).

Consequently, it only remains to prove that if the conditions of Theorem 1 are met, then ψ_θ is injective when $\mathcal{T}$ is a Euclidean space.

Proposition 2. *If the conditions of Theorem 1 are met and the domain of ψ_θ is $\mathcal{T}$; then, ψ_θ is injective.*

Proof. By contradiction, let us assume that these conditions are met and ψ_θ is not injective. Then, let us take two elements of $\mathcal{T}$, x_{a_0} and x_{b_0}, where $x_{a_0} \neq x_{b_0}$. If ψ_θ is not injective, there $\exists x_{a_0}$ and $\exists x_{b_0}$, such that $\psi_\theta(x_{a_0}) = \psi_\theta(x_{b_0})$. In such cases, from Condition 1) of Theorem 1, we know that as $t \to \infty$, the mappings of these elements will generate the same trajectory in $\mathcal{L}$ following the dynamical system $f^{\mathcal{L}}$, which converges to y_{g}.

Since the evolution of the variables x_{a_0} and x_{b_0} is completely defined by the evolution of their mappings in $\mathcal{L}$ (i.e., $\dot{x} = \phi(y)$), both variables will present the same time derivative in $\mathcal{T}$. Consequently, given that the trajectories obtained when starting from x_{a_0} and x_{b_0}, at time t, are described by $x_i(t) = x_{i_0} + \int_{\tau=0}^{t} f(x(\tau))d\tau$[1], where $i \in \{a, b\}$, their integral part will be the same $\forall t$. Then, $\forall t$ the distance between both trajectories is $d(x_{a_0}, x_{b_0}) = ||x_a(t) - x_b(t)|| = ||x_{a_0} - x_{b_0}||$, where $d(\cdot, \cdot)$ is a distance function.

Thus, as $t \to \infty$, x_a and x_b will converge to two different points $x_{a_{\text{g}}}$ and $x_{b_{\text{g}}}$, respectively. However, we also stated that their respective mappings converge to y_{g}, i.e., $\psi_\theta(x_{a_{\text{g}}}) = \psi_\theta(x_{b_{\text{g}}}) = y_{\text{g}}$. In this case, Condition 2) of Theorem 1 implies that $x_{a_{\text{g}}} = x_{b_{\text{g}}} = x_{\text{g}}$. This contradicts the fact that $x_{\text{g}}^a \neq x_{\text{g}}^b$. Consequently, ψ_θ is injective. □

Finally, from Proposition 2 we can conclude that if the conditions of Theorem 1 are enforced in $f_\theta^{\mathcal{T}}$; then, ψ_θ will approximate a diffeomorphism.

B.2. Stability of $f^{\mathcal{L}}$ with Adaptive Gains

In this section, we show that y_g is globally asymptotically stable in the system introduced in (5.9) when the adaptive gain $\alpha(y_t)$ is greater than zero. Note that the derivation introduced here is analogous to the one of the discrete-time case when the system is simulated using the forward Euler integration method. However, in the latter, the condition for global asymptotic stability is $0 < \alpha(y_t) < 2/\Delta t$.

To show global asymptotic stability, we introduce the Lyapunov candidate $V(y_t) = y_t^\top y_t$ and study if the condition $\dot{V}(y_t) < 0$ holds for all $y_t \in \mathbb{R}^n$. By introducing $A = \text{diag}(-\alpha(y_t))$ and, without loss of generality, setting $y_g = 0$, we

[1] In discrete time, the integral transforms into a summation.

write (5.9) as $\dot{y}_t = Ay_t$. Then,

$$\begin{aligned}\dot{V}(y_t) &= (Ay_t)^\top y_t + y_t^\top (Ay_t) \\ &= y_t^\top (A + A^\top) y_t \\ &= 2y_t^\top A y_t \quad \text{(since } A \text{ diagonal)}.\end{aligned} \tag{B.1}$$

Therefore, it follows that this function is negative when the eigenvalues of A are negative. Since A is a diagonal matrix, the eigenvalues correspond to its diagonal $-\alpha(y_t)$. Consequently, y_g is globally asymptotically stable in the system (5.9) when $\alpha(y_t) > 0$.

B.3. Neural Network Architecture

In this appendix, we provide details regarding the Neural Network's architecture. The criteria employed to design the architecture were to build a network: 1) large enough for it to be very flexible in terms of the motions that it can represent, and 2) with a reasonable size such that it can do inference in real time. Consequently, we observed that 3 feedforward fully connected layers, with 300 neurons each, for ψ_θ and ϕ_θ, i.e., 6 layers in total, were enough for obtaining accurate results and low inference times. The employed activation function was GELU [235] for every layer except for the last layer of the network, which had a linear activation, and for the last layer of α, which had a sigmoid function. In our case, the network inferred at $\mathbf{677 \pm 57}$ **Hz** (confidence interval with one standard deviation) using a laptop PC with an Intel i7-8750H (12) @ 4.100GHz CPU and an NVIDIA GeForce RTX 2070 Mobile GPU. PyTorch had the GPU enabled at inference time.

For the case of the adaptive gains $\alpha(y_t^{\mathcal{T}})$, two layers were employed instead. Note that these layers only affect the training time of the network, given that they are not required for inference.

Finally, layer normalization [236] was added after each layer of the network except for the last layers of ψ_θ, ϕ_θ and α. This type of normalization has shown to be beneficial for reducing training times and also for helping with vanishing gradients.

B.4. Hyperparameter Optimization

We introduce a hyperparameter optimization strategy for CONDOR's different variations on the LASA and LAIR datasets. We define an accuracy metric, $\mathcal{L}_{\text{acc}}$, calculated using the distance metrics from Sec. 5.5.1. We also evaluate the stability of the system by minimizing the diffeomorphism mismatch, i.e., the RMSE between $y_{1:\text{N}}^{\mathcal{L}}$ and $y_{1:\text{N}}^{\mathcal{T}}$, defining the stability term, $\mathcal{L}_{\text{stable}}$. Lastly, we account for the precision of the learned system's goal versus the real goal by measuring the average distance of all final trajectory points to the goal, creating the term $\mathcal{L}_{\text{goal}}$. Then, we define the following objective:

$$\mathcal{L}_{\text{hyper}} = \mathcal{L}_{\text{acc}} + \gamma_{\text{stable}}\mathcal{L}_{\text{stable}} + \gamma_{\text{goal}}\mathcal{L}_{\text{goal}}, \tag{B.2}$$

Table B.1.: Hyperparameter optimization results of CONDOR.

Hyperparameter	Opt.?	Hand-tuned value / initial opt. guess				Optimized value			
		CONDOR	CONDOR (fixed gains)	CONDOR (triplet)	CONDOR (1st order LAIR)	CONDOR***	CONDOR (fixed gains)	CONDOR (triplet)	CONDOR (1st order LAIR)
CONDOR									
Max adap. latent gain ($\alpha_{\max}$)	✓	1e-2	8e-3*	1e-2	9.997e-2	9.997e-2	2.470e-3*	3.970e-2	0.174
Stability loss weight (λ)	✓	1	1	1	9.300e-2	9.300e-2	3.481	0.280	2.633
Window size imitation (H^i)	✓	14	14	14	14	14	14	14	1
Window size stability (H^s)	✓	4	4	2	4	1	1	2	8
Contrastive margin (m)	✓	1e-2	1e-2	1e-4**	3.334e-2	3.334e-2	3.215e-3	1.977e-4**	1.557e-2
Batch size imitation (B^i)	✗	250	250	250	250	-	-	-	-
Batch size stability (B^s)	✗	250	250	250	250	-	-	-	-
Neural Network									
Optimizer	✗	AdamW	AdamW	AdamW	AdamW	-	-	-	-
Number of iterations	✗	40000	40000	40000	40000	-	-	-	-
Learning rate	✓	1e-4	1e-4	1e-4	1e-4	4.855e-4	4.295e-4	8.057e-4	5.553e-5
Weight decay	✗	1e-4	1e-4	1e-4	1e-4	-	-	-	-
Activation function	✗	GELU	GELU	GELU	GELU	-	-	-	-
Num. layers (ψ_θ, ϕ_θ, α)	✗	(3, 3, 3)	(3, 3, -)	(3, 3, 3)	(3, 3, 3)	-	-	-	-
Neurons/hidden layer	✗	300	300	300	300	-	-	-	-
Layer normalization	✗	yes	yes	yes	yes	-	-	-	-

* Corresponds to the optimized fixed gain.
** Corresponds to the triplet loss margin.
*** Also applies for CONDOR (2nd order).

where γ_{stable} and γ_{goal} are weighting factors. After initial tests, we settled on $\gamma_{\text{stable}} = 0.48$ and $\gamma_{\text{goal}} = 3.5$.

In practical applications, hyperparameter tuning has limitations like time consumption and susceptibility to the curse of dimensionality. To mitigate this, we focused on five strategies: reducing the objective function's overhead, limiting the evaluation set, employing Bayesian optimization, pruning, and selecting a subset of hyperparameters.

Reduced objective function's overhead

The objective function minimized in the hyperparameter optimization process is periodically computed throughout each learning process. Consequently, if this function is expensive to compute, it will make the optimization process slower. More specifically, we observe that the computation of the accuracy using the DTWD and FD metrics adds considerable overhead to the computation time of the objective function. Furthermore, we also observe that the values of the RMSE, DTWD, and FD are highly correlated. Therefore, since computing the RMSE is much faster than computing the other metrics, the hyperparameter optimization loss that accounts for accuracy only consists of the RMSE, i.e., ℓ_{IL} with $H^i = n$ and $t' = 0$.

Reduced evaluation set

Optimizing hyperparameters for the LASA/LAIR dataset using different motions simultaneously is challenging since the objective computed from different motions is not comparable. Instead, we focused on optimizing using a single, *difficult* motion, assuming robust hyperparameters for it would perform well overall. We selected the *heee* motion from the LASA dataset for first-order motions and the *capricorn* motion from the LAIR dataset for second-order motions. These motions,

with complex features like large curvatures or sharp edges, represented challenging test cases.

Bayesian optimization

During optimization, every evaluation of a different set of hyperparameters is costly. Therefore, instead of randomly selecting the hyperparameters to evaluate at each run or following a grid search approach, we select the most promising set given the ones evaluated so far. To achieve this, we employ the Tree Parzen Estimator (TPE) [237], which builds a probability model of the objective function and uses it to select the next set of hyperparameters based on how promising they are. We use the implementation available in the Optuna API [238].

Pruning

Throughout the optimization process, it is possible to detect inauspicious runs after a few evaluations. Hence, these trials can be *pruned* before the training process ends, freeing the computational resources for a new run to be executed. We also incorporate this feature in the optimization process using the pruning method available in the Optuna API.

Select a subset of hyperparameters

Finally, before starting the optimization process, by interacting with CONDOR, we identified a subset of the hyperparameters that showed to have the largest influence over its results. Therefore, to reduce the dimensionality of the search problem, only this subset of hyperparameters is optimized. The rest are manually tuned based on our interactions with the framework.

B.4.1. Results

Table B.1 details the results of the hyperparameters optimization process, including the optimized parameters, and their pre- and post-optimization values. It is divided into two sections: hyperparameters specific to CONDOR, and those general to DNNs. Note that most optimized hyperparameters pertain to the CONDOR method. For the LAIR dataset, we used LASA's optimized hyperparameters as a starting point, resulting in no improved set found for the second-order CONDOR method. Hyperparameters used in BC are excluded as those applicable were identical to those of CONDOR.

Note that the hyperparameter $\alpha_{\max} \in (0, 1]$ has not been introduced yet. This hyperparameter limits the maximum value of the adaptive gain α in $f^{\mathcal{L}}$ (see Sec. 5.4.5). Hence, if in this work we define $\Delta t = 1$ for α in $f^{\mathcal{L}}$; then, even though its maximum allowed value is 1, we limit it even further using $\alpha_{\max}$. This process improves CONDOR's performance, as observed in preliminary experiments. Hence, we define $\alpha = \alpha_{\max}\bar{\alpha}(y_t^{\mathcal{T}})$, with $\bar{\alpha}$ as the DNN output using the sigmoid activation function, ensuring $\alpha \in (0, \alpha_{\max})$.

B.5. Real-World Experiments: Low-Level Control

Regarding the low-level control strategy employed in this work, we focus on fully actuated rigid body dynamics systems, which evolve according to the following equation of motion:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + D(q)\dot{q} = u, \tag{B.3}$$

where q is the joint angle vector, $M(q)$ is the inertia matrix, $C(q,\dot{q})$ is the Coriolis/centripetal vector, $G(q)$ is the gravity vector, $D(q)$ is the viscous friction matrix and u is the actuation torque vector [239].

The objective of the low-level controller is to, at every time step, map the desired state x^d and state derivative $\dot{x}^d$ provided by CONDOR to u, such that the system is driven towards the desired state. In our experiments, we learn motions in task space and in joint space. Hence, we employ slightly different strategies for each case.

The control frequency of the low-level controller is **500 Hz** in every experiment.

B.5.1. Joint space control

In this work, independently of the task that CONDOR controls, every motion reference is eventually mapped to joint space. Hence, this subsection explains our approach to track this reference in joint space $(q^d, \dot{q}^d)$. We achieve this by means of a proportional-derivative (PD) controller with gravity compensation, i.e.,

$$u = \alpha(q^d - q) + \beta(\dot{q}^d - \dot{q}) + G(q), \tag{B.4}$$

where α and β are gain matrices. The higher the gains of this controller, the smaller the tracking error [240, 241]. Moreover, an interesting property of this approach is that as q^d approaches q_g, where q_g corresponds to the mapping from x_g to the configuration space of the robot, CONDOR makes $\dot{x}^d$, and in consequence $\dot{q}^d$, tend to 0. Then, (B.4) behaves similarly to

$$u = \alpha(q^d - q) - \beta\dot{q} + G(q). \tag{B.5}$$

This control law ensures global asymptotic stability at the equilibrium q^d for any choice of α and β as long as these are positive definite matrices [239].

B.5.2. Task space control: Online

To control the robot when references $\dot{x}^d$ are given online (see Fig. 5.5a) in task space, we use the real-time Inverse Kinematics (IK) library TRACK-IK [242]. To do so, we integrate the velocity reference using the forward Euler integrator to obtain x^d, and we map this position to joint space using this library to obtain q^d. We apply exponential smoothing to these results to alleviate vibrations and stuttering issues. Finally, we compute the desired velocity $\dot{q}^d$ using the forward difference of q, i.e., $\dot{q}^d = (q^d - q)/\Delta t$, where Δt is the time step length of CONDOR. Then, at every time step, the values $q^d, \dot{q}^d$ are provided to the controller described in Appendix B.5.1.

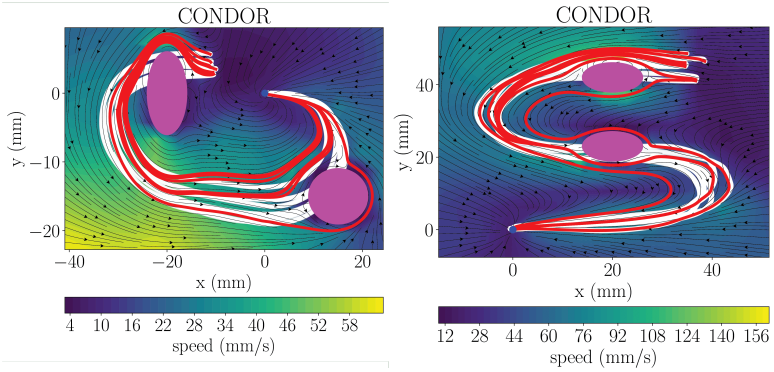


Figure B.1.: Obstacle avoidance in the LASA dataset.

B.5.3. Task space control: Offline

In the offline case (see Fig. 5.5b), a trajectory in task space $(x_0^d, x_1^d, ..., x_N^d)$ is first computed with CONDOR. This trajectory is fed to the low-level controller to execute it offline. To achieve this, firstly, we map the trajectory to joint space using the Levenberg-Marquadt IK solver of the Robotics Toolbox [243]. In this case, we employ this solver instead of TRACK-IK, because it is robust around singularities and can avoid problems like stuttering [244]. This is important for obtaining very smooth solutions in scenarios where this is critical, such as in writing tasks. Note that the solver does not run in real time; however, this is not problematic, since the IK solutions are computed offline.

Afterward, the resulting joint space reference trajectory is approximated with a spline [245] that evolves as a function of time. Finally, when the motion starts, the time is incremented by Δt at each time step and used to query the reference value that is sent to the controller described in Appendix B.5.1.

B.6. Obstacle Avoidance

Obstacle avoidance for motions modeled as dynamical systems is a problem that has been addressed in the literature [196, 198, 246]. Any of these methods can be combined with our proposed framework. In this work, we implemented the method presented in [196] in PyTorch, and combined it with CONDOR. We compute a modulation matrix $M(x)$ that, when multiplied with the learned dynamical system $f(x)$, modifies the motion such that a new dynamical system $\bar{f}(x) = M(x)f(x)$ is obtained. $\bar{f}(x)$ avoids obstacles while maintaining the stability properties of $f(x)$. For more details please refer to [196] (obstacle avoidance of multiple convex obstacles).

Fig. B.1 shows motions from the LASA dataset where we test this approach. We observe that the motions generated with CONDOR remain stable after applying the modulation matrix. Furthermore, the obstacles are successfully avoided, showing that, as expected, the dynamical system motion formulation of CONDOR can be

effectively combined with methods designed to work with dynamical systems.

Finally, this method was tested with 3D obstacles in a real 7DoF robot manipulator when controlling its end effector position. These results are provided in the attached video.

C

Additional Proofs and Method Details (Chapter 6)

C.1. Upper Bound β

In this section, we introduce and prove the proposition used in Sec. 6.4.2 to demonstrate that the surrogate stability conditions ensure asymptotic stability in the dynamical system $f_\theta^{\mathcal{T}}(x_t)$.

Proposition 3 (Existence of class-$\mathcal{KL}$ function)**.** *Consider a dynamical system $\dot{y}_t = f(y_t)$ with $f : \mathcal{L} \subset \mathbb{R}^n \to \mathbb{R}^n$ continuously differentiable, and $\Phi_\theta^y(t, y_0) : \mathbb{R}_{\geq 0} \times \mathcal{L} \to \mathcal{L}$ as its evolution function. For a distance function d_t in $\mathcal{L}$, we define its evolution, for a given t and y_0, as $\delta : \mathcal{L} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, with $\delta(y_0, t) = ||y_g - \Phi_\theta^y(t, y_0)||$.*

Then, consider the function $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ defined as

$$\beta(d_0, t) = z_0 + \int_0^t \dot{z}_s ds, \tag{C.1}$$

with derivative

$$\dot{z}_t = \alpha\big(z_t - \delta^{max}(d_0, t)\big), \tag{C.2}$$

where $\alpha \in \mathbb{R}_{<0}$ and

$$\delta^{max}(d_0, t) = \max_{y_0 \in \mathcal{Y}_0} \left(\max_{s \in [t, t+\Delta t]} \delta(y_0, s) \right), \tag{C.3}$$

with $\mathcal{Y}_0(d_0) = \{y_0 \in \mathcal{L} : ||y_g - y_0|| = d_0\}$ and $\Delta t \in \mathbb{R}_{>0}$. The initial condition z_0 is set as

$$z_0 = \delta_0^{max} + d_0, \tag{C.4}$$

where $\delta_0^{max} = \delta^{max}(d_0, 0)$.

Then, under the conditions of Theorem 5, i.e., $\forall t \in \mathbb{R}_{\geq 0}$,

1. $d_t = d_{t+\Delta t}$, *for $y_0 = y_g$,*

2. $d_t > d_{t+\Delta t}, \quad \forall y_0 \in \mathcal{L} \setminus \{y_g\},$

there exists a class-$\mathcal{KL}$ function β such that $\forall y_t \in \mathcal{L}$ and $\forall t \in \mathbb{R}_{\geq 0}$,

$$||y_g - y_t|| \leq \beta(d_0, t), \tag{C.5}$$

which can also be expressed as

$$\delta(y_0, t) \leq \beta(d_0, t). \tag{C.6}$$

Proof. To prove that a function β is an upper bound of δ and is class-$\mathcal{KL}$, we need the following conditions to hold $\forall y_t \in \mathcal{L}$ and $\forall t \in \mathbb{R}_{\geq 0}$:

i) $\beta(0, t) = 0$,

ii) β decreases with t,

iii) $\delta \leq \beta$,

iv) β is continuous with respect to d_0 and t,

v) $\beta \to 0$ as $t \to \infty$,

vi) β strictly increases with d_0.

Therefore, we proceed to prove all of these conditions. A summary of this proof is depicted in Fig. C.1.

$\beta(0, t) = 0$

To demonstrate that this condition is valid, we introduce Lemma 1. For this lemma to be applicable, a specific condition must be met. To validate this condition, we refer to Lemma 2 and Corollary 1, both of which will be introduced subsequently. We also present Corollary 2, which will prove beneficial in subsequent discussions.

Lemma 1 ($\beta(0, t) = 0$ constant)**.** *$\beta(0, t) = 0$ if $\delta^{max}(0, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$.*

Proof. If $\delta^{\max}(0, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$, for $d_0 = 0$, from equations (C.4) and (C.2), we can observe that $z_0 = 0$ and $\dot{z}_t = 0$, $\forall t \in \mathbb{R}_{\geq 0}$. Replacing this in (C.1), we conclude that $\beta(0, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$. □

Lemma 2 ($\delta = 0$ constant)**.** *$\delta(y_g, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$, if:*

1. *$d_t = d_{t+\Delta t}, \quad$ for $y_{t+\Delta t} = y_g$ (Theorem 5, 1),*
2. *$d_t > d_{t+\Delta t}, \quad \forall y_t \in \mathcal{L} \setminus \{y_g\}$ (Theorem 5, 2).*

Proof. Let us introduce t_g, which is the earliest $t \in \mathbb{R}_{\geq 0}$ where $y_t = y_g$. Then, condition 1 gives rise to two scenarios for every state visited after t_g:

i) *$\delta(y_0, t)$ is constant:* A constant function satisfies $d_t = d_{t+\Delta t}$.

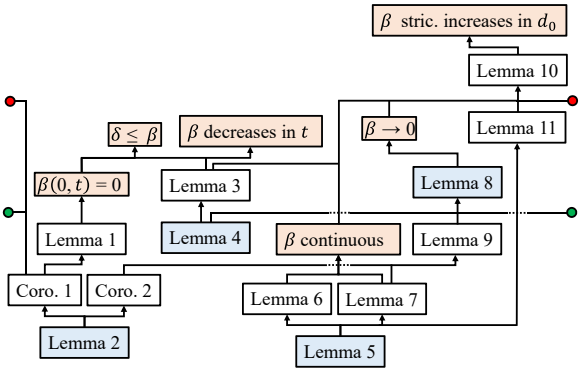


Figure C.1.: Summary of lemmas and corollaries related to the proof of Proposition 3. Bisque-colored boxes indicate the conditions necessary for the proposition to be true, while blue boxes represent lemmas relying directly on its assumptions and requirements.

ii) $\delta(y_0, t)$ *is periodic:* A set of functions could meet the condition $d_t = d_{t+\Delta t}$ by varying over time but always returning to the same value after Δt seconds. However, in this case, the only possibility is that, for all $t > t_g$, we have $\delta(y_0, t) = \delta(y_0, t + \Delta t)$, i.e., a periodic function. This is because the time derivative of δ is solely defined by y_t, since $\partial\delta/\partial t = \partial\delta/\partial y_t \cdot \dot{y}_t$, and both of these variables only depend on y_t. Therefore, $\partial\delta/\partial t$ at y_g, and at any state visited afterwards, given the state, must always be the same. This implies that δ must evolve over time to the same states as those to which it evolved Δt seconds ago, since for every state, the derivative is the same as it was Δt seconds before.

However, the periodic case for $\delta(y_0, t)$ contradicts condition 2, which requires that δ strictly decreases after Δt seconds, and in the periodic scenario, it returns to the same value after Δt seconds. Hence, the only remaining scenario is that δ is constant after t_g, with a value of 0 (by definition). Lastly, by noting that $y_0 = y_g$ implies that $t_g = 0$, we get that $\delta(y_g, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$. □

Corollary 1 ($\delta^{\max}(0,t) = 0$)**.** $\delta^{max}(0,t) = 0$ *if* $\delta(y_g, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$.

Proof. If $\delta(y_g, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$, and noting $d_0 = 0$ implies that every $y_0 \in \mathcal{Y}_0$ in (C.3) is equal to y_g, from (C.3), it follows that $\delta^{\max}(0,t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$. □

Corollary 2 ($\min(\delta^{\max})$)**.** $\min(\delta^{max}) = 0$ *if* $\delta^{max}(0,t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$.

Proof. Recall that δ is positive definite and that $\delta^{\max}$ takes the value of some δ. Then, $\delta^{\max}(0,t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$, indicates that this is the minimum value $\delta^{\max}$ can achieve. Hence, $\min(\delta^{\max}) = 0$. □

Then, provided that the conditions of Proposition 3 are met, the conditions of Lemma 2 are fulfilled. Therefore, $\delta(y_g, t) = 0$ for every $t \in \mathbb{R}_{\geq 0}$, and, hence, $\delta^{\max}(0, t) = 0$ (from Corollary 1). Consequently, we can employ Lemma 1 to demonstrate that $\beta(0, t) = 0$ for all $t \in \mathbb{R}_{\geq 0}$.

β decreases with t, and $\delta \leq \beta$

We introduce Lemma 3, which presents one condition that, if satisfied, implies that $\beta > \delta$ and β strictly decreases over time, $\forall d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$. After proving this lemma, we introduce Lemma 4, which is then used to show that this condition is satisfied in our case. Lastly, we combine this with the fact that $\beta(0, t) = 0$ to conclude that β decreases with t (i.e., is a non-increasing function), and $\delta \leq \beta$, $\forall d_0 \in \mathbb{R}_{\geq 0}$.

Lemma 3 ($\beta > \delta$ and strictly decreasing)**.** *$\beta(d_0, t) > \delta(y_0, t)$ and $\beta(d_0, t)$ strictly decreases with respect to t, $\forall d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$, $\forall t \in \mathbb{R}_{\geq 0}$, if:*

1. *δ^{max} decreases with respect to t, $\forall d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$.*

Proof. Although $\delta^{\max}$ evolves as a function of time, we can infer from (C.2) that, locally, β behaves as a linear first-order dynamical system, with the origin at $\delta^{\max}$. Given that $\alpha < 0$, it follows that β will converge towards $\delta^{\max}$ over time.

Moreover, (C.4) indicates that $z_0 > \delta_0^{\max}$ for every $d_0 > 0$. Given the properties of linear first-order systems, β will strictly decrease towards $\delta_0^{\max}$, without ever reaching or overshooting this value. Combined with condition 1, which indicates that $\delta^{\max}$ decreases with respect to t for all $d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$, we can conclude that β will always be greater than $\delta^{\max}$ and will continue to strictly decrease towards this value as a function of time. Therefore, $\beta > \delta$ and β strictly decreases with respect to t, $\forall d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$.

□

Lemma 4 ($\delta^{\max}$ decreases over time)**.** *$\delta^{max}(d_0, t)$ decreases with respect to t, $\forall d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$, $\forall t \in \mathbb{R}_{\geq 0}$, if:*

1. *$d_t > d_{t+\Delta t}$, $\forall t \in \mathbb{R}_{\geq 0}$, $\forall y_0 \in \mathcal{L} \setminus \{y_g\}$, (Theorem 5, 2).*

Proof. Let us define

$$\delta^{\max}_{t+\Delta t}(y_0, t) = \max_{s \in [t, t+\Delta t]} \delta(y_0, s), \tag{C.7}$$

which allows us to write $\delta^{\max} = \max_{y_0 \in \mathcal{Y}_0} \left(\delta^{\max}_{t+\Delta t}\right)$. We will first prove that $\delta^{\max}_{t+\Delta t}$ decreases with t.

To do so, observe that condition 1 can be rewritten as $\delta(y_0, t + \Delta t) < \delta(y_0, t)$. Given that this condition holds true for every s in the interval $[t, t + \Delta t]$, none of the values of $\delta(y_0, s)$ computed in this interval can exceed the maximum of this interval, i.e., $\delta^{\max}_{t+\Delta t}(y_0, t)$, after Δt seconds. If there were such a value $\delta(y_0, s)$ that is greater than $\delta^{\max}_{t+\Delta t}(y_0, t)$ after Δt seconds, it would imply that $\delta(y_0, s)$ had increased, for some s, after Δt, contradicting condition 1. Therefore, $\delta^{\max}_{t+\Delta t}$ can

only decrease as time progresses. Since this holds for all instances of t, we conclude that $\delta^{\max}_{t+\Delta t}$ decreases with respect to t, $\forall y_0 \in \mathcal{L} \setminus \{y_g\}$.

Now, the function $\delta^{\max}$ computes the maximum over a set of functions $\delta^{\max}_{t+\Delta t}$ with different initial conditions $y_0 \in \mathcal{Y}_0(d_0)$. However, for any given d_0, $\forall y_0 \in \mathcal{L} \setminus \{y_g\}$, each one of these functions decreases with respect to time. Then, for any given interval of time, we have two possible scenarios:

i) $\delta^{\max}$ is equal to one function $\delta^{\max}_{t+\Delta t}$ (the current maximum). In this case, $\delta^{\max}$ decreases with time, since $\delta^{\max}_{t+\Delta t}$ decreases with time.

ii) The function that is currently the maximum over the set of functions $\delta^{\max}_{t+\Delta t}$ reaches a point in time t' where it changes. At the point where the change occurs, the function will be lower than or equal to its previous values for $t < t'$, since the previous maximum is decreasing. Furthermore, it will be greater than or equal to the values for $t > t'$, since the new maximum also decreases.

Therefore, we can conclude that $\forall d_0$ where $y_0 \in \mathcal{Y}_0(d_0)$ is not y_g, $\delta^{\max}$ decreases with respect to time. Moreover, by noting that $\forall y_0 \in \mathcal{Y}_0(d_0)$, $d_0 \neq 0$ implies that $y_0 \neq y_g$, it follows that $\delta^{\max}$ decreases with respect to time, $\forall d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$, $\forall t \in \mathbb{R}_{\geq 0}$. □

As a consequence, the assumptions and conditions outlined in Proposition 3 enable us to apply Lemma 4 to demonstrate that the conditions of Lemma 3 are satisfied. This, in turn, leads us to the conclusion that $\beta > \delta$ and $\beta(d_0, t)$ strictly decreases with respect to t, $\forall d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$, $\forall t \in \mathbb{R}_{\geq 0}$. Lastly, before we showed that, for $d_0 = 0$, $\beta(0, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$. Therefore, in general, we have that $\beta \geq \delta$ and β decreases with respect to t, $\forall d_0 \in \mathbb{R}_{\geq 0}$, $\forall t \in \mathbb{R}_{\geq 0}$.

Continuity of β

We require β to be continuous with respect to t, for each fixed d_0, and with respect to d_0, for each fixed t. To achieve this, we will introduce lemmas 5, 6 and 7.

Lemma 5. *[Continuity of δ] If $\dot{y}_t = f(y_t)$ is continuously differentiable, then $\delta(y_0, t)$ is continuous with respect to y_0 and t.*

Proof. Given that $\dot{y}_t = f(y_t)$ is continuously differentiable, it follows that $\Phi^y_\theta(t, y_0)$ is continuously differentiable with respect to both t and y_0 [247]. Moreover, since δ is a metric on $\mathcal{L}$ defined by $\delta = ||y_g - \Phi^y_\theta(t, y_0)||$, it is continuous with respect to $\Phi^y_\theta(t, y_0)$ [248]. Since $\Phi^y_\theta(t, y_0)$ is continuously differentiable and thus continuous, and the composition of two continuous functions is continuous, it follows that $\delta(y_0, t)$ is continuous with respect to both y_0 and t. □

Lemma 6 ($\delta^{\max}$ continuous with respect to t)**.** *δ^{max} is a continuous function with respect to t, if δ is continuous with respect to t.*

Proof. We will first prove this for $\delta^{\max}_{t+\Delta t}$ (as defined in (C.7)). Given that $\delta^{\max}_{t+\Delta t}$ is the maximum value of $\delta(y_0, s)$ over the interval $[t, t+\Delta t]$, any change in $\delta^{\max}_{t+\Delta t}$ must be due to a change in δ for some s in the boundaries of $[t, t+\Delta t]$. Then, since δ changes continuously, $\delta^{\max}_{t+\Delta t}$ can only change continuously as well. Thus, $\delta^{\max}_{t+\Delta t}$ is continuous with respect to t.

However, we need to show that $\delta^{\max} = \max_{y_0 \in \mathcal{Y}_0} \left(\delta^{\max}_{t+\Delta t}\right)$ is continuous with respect to t. This follows from the fact that $\delta^{\max}$ computes the point-wise maximum along t over a set of continuous functions $\delta^{\max}_{t+\Delta t}$, which results in a continuous function [249]. □

C

Lemma 7 ($\delta^{\max}$ continuous with respect to d_0)**.** *δ^{max} is continuous with respect to d_0, if δ is continuous with respect to y_0.*

Proof. Given the continuity of the function $\delta(y_0, t)$ with respect to y_0, the function $\delta^{\max}_{t+\Delta t}$ computes the point-wise maximum along y_0 over a set of continuous functions, specifically $\{\delta(y_0, s) : s \in [t, t+\Delta t]\}$. Consequently, $\delta^{\max}_{t+\Delta t}$ is also continuous in y_0 [249].

Building on this, we seek to demonstrate that $\delta^{\max} = \max_{y_0 \in \mathcal{Y}_0} \left(\delta^{\max}_{t+\Delta t}\right)$ is continuous with respect to d_0. To accomplish this, we will use *the maximum theorem* [250]. This theorem states that if $\mathcal{Y}_0(d_0)$ is *continuous* with respect to d_0 and *compact-valued*[1], and $\delta^{\max}_{t+\Delta t}(y_0, t)$ is continuous in y_0, then $\delta^{\max}(d_0, t)$ is continuous in d_0.

$\mathcal{Y}_0(d_0)$ is continuous because the relationship between each y_0 and d_0, i.e., the metric on $\mathcal{L}$, is continuous [248]. $\mathcal{Y}_0(d_0)$ is compact-valued if, for each d_0, $\mathcal{Y}_0$ constitutes a compact set. Given that $\mathcal{Y}_0 \subset \mathbb{R}^n$, the *Heine-Borel theorem* [249] indicates that $\mathcal{Y}_0$ is compact if it is both closed and bounded. Every $\mathcal{Y}_0$ is closed as it fulfills an algebraic equation, and can be contained within any ball of radius larger than d_0, making it bounded. Consequently, $\mathcal{Y}_0(d_0)$ is compact-valued.

Therefore, since $\delta^{\max}_{t+\Delta t}(y_0, t)$ is continuous with respect to y_0, and $\mathcal{Y}_0(d_0)$ is both continuous and compact-valued, the maximum theorem states that $\delta^{\max}(d_0, t)$ is continuous with respect to d_0. □

Now, we can proceed to conclude about the continuity of β for both t and d_0.

1. *Continuity in t*: For any given d_0, since β evolves as a linear first-order system (as per (C.1)), its solution will exist provided that $\delta^{\max}$ is continuous with respect to t [251]. Then, under the assumptions of Proposition 3, Lemma 6 confirms that $\delta^{\max}$ is indeed continuous, provided that δ is continuous. From Lemma 5 we infer that δ is continuous with respect to t. Hence, β is well-defined for any given d_0, indicating that it is differentiable with respect to t, and, therefore, continuous.

2. *Continuity in d_0*: For any given t, from (C.1), we know that β will be continuous with respect to d_0 as long as both of its terms, z_0 and the intregral of $\dot{z}_t$, are continuous. The continuity of both terms depends on the continuity of

[1] The concepts of *compact-valued* and *continuous* refer to those employed in the literature of set-valued functions/correspondences. For further details, we refer the reader to [250].

$\delta^{\max}$. Then, since Lemma 7 indicates that $\delta^{\max}$ is continuous with respect to d_0 provided that δ is continuous with respect to y_0, and Lemma 5 confirms that this is indeed the case (given the assumptions of Proposition 3), we conclude that β is continuous with respect to d_0.

$\beta \to 0$ as $t \to \infty$

To prove that this condition holds, we will utilize Lemma 8. The application of this lemma necessitates the use of Lemma 4 and Corollary 1, which are already introduced previously, and Lemma 9, which is introduced afterwards.

Lemma 8 (β converges to zero)**.** *$\beta \to 0$ as $t \to \infty$ if:*

1. $\delta^{max} \in [0, \max(\delta)]$,
2. δ^{max} *decreases with respect to* t, $\forall d_0 \in \mathbb{R}_{\geq 0}$, $\forall t \in \mathbb{R}_{\geq 0}$,
3. $d_t < d_{t-\Delta t}$, $\forall y_0 \in \mathcal{L} \setminus \{y_g\}$, $\forall t \in \mathbb{R}_{\geq 0}$, *(Theorem 5, 2).*
4. δ^{max} *surjective in* d_0,
5. $\delta^{max}(0, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$.

Proof. According to (C.2), β approaches $\delta^{\max}$ as $t \to \infty$. Thus, demonstrating that $\delta^{\max} \to 0$ as $t \to \infty$ would imply that $\beta \to 0$ as $t \to \infty$.

Note that $\delta^{\max} \in [0, \max(\delta)]$ (condition 1) and $\delta^{\max}$ decreases for all t (condition 2). This indicates that as t goes to infinity, $\delta^{\max}$ must approach a limit $a \in [0, \max(\delta)]$. We will proceed to show that this limit can only be zero.

Let us define $y_0^* \in \mathcal{Y}_0$ and $t^* \in \mathbb{R}_{\geq 0}$ as the variables in (C.3) where the maxima are achieved for a given d_0 and t. Thus, $\delta^{\max} = \delta(y_0^*, t^*)$. From condition 3, we know that $\delta(y_0^*, t^* + \Delta t) < \delta(y_0^*, t^*)$, $\forall y_0^* \in \mathcal{L} \setminus \{y_g\}$, $\forall t^* \in \mathbb{R}_{\geq 0}$. This leads to two scenarios:

i) y_0^* *remains constant:* If y_0^* does not change in the interval $[t, t^* + \Delta t]$, then $\delta(y_0^*, t^*)$, and therefore $\delta^{\max}$, must strictly decrease at some point within this interval. Otherwise, $\delta(y_0^*, t^* + \Delta t) < \delta(y_0^*, t^*)$ would not hold true.

ii) y_0^* *changes:* If y_0^* changes during the interval $[t, t^* + \Delta t]$, given that $\delta^{\max}$ decreases with t (condition 2), this change can only occur if $\delta(y_0^*, t^*)$, and hence $\delta^{\max}$, strictly decreases at some point within the interval.

In either case, $\delta^{\max}$ strictly decreases at some point in the interval $[t, t^* + \Delta t]$, $\forall d_0 \in \mathbb{R}_{\geq 0} \setminus \{0\}$, $\forall t \in \mathbb{R}_{\geq 0}$.

Taking this into account, if the limit was some value $a \neq 0$, a contradiction would arise. If $a \neq 0$, we would always have a $d_0 \neq 0$ such that $\delta^{\max}(d_0, 0) = a$ (conditions 4 and 5). This implies that $\delta^{\max}$ must become lower than a before $t^* + \Delta t$, and, since it is decreasing (condition 2), it will remain lower as time progresses. Therefore, the only feasible value for the limit is $a = 0$, which confirms $\delta^{\max} \to 0$ as $t \to \infty$. Consequently, $\beta \to 0$ as $t \to \infty$. □

Lemma 9 ($\delta^{\max}$ with d_0)**.** *$\delta^{max} \in [0, \max(\delta)]$, and δ^{max} surjective with respect to d_0, if:*

1. $\min(\delta^{max}) = 0$,
2. *δ^{max} continuous with respect to d_0.*

Proof. Given that $\delta^{\max}$ computes a maximum over values of δ, we get $\max(\delta^{\max}) = \max(\delta)$. Then, we know that $\min(\delta^{\max}) = 0$ (condition 1), and that $\delta^{\max}$ is continuous with respect to d_0 (condition 2). Therefore, as a consequence of the *intermediate value theorem*, $\delta^{\max}$ can take any value between $\min(\delta^{\max})$ and $\max(\delta^{\max})$, i.e., $\delta^{\max} \in [0, \max(\delta)]$. Moreover, each value of $\delta^{\max} \in [0, \max(\delta)]$ must have at least one corresponding value of d_0; hence, in this interval, $\delta^{\max}(d_0, t)$ is surjective with respect to d_0. □

To summarize, given the conditions and assumptions of Proposition 3, Lemmas 4, 9 and Corollary 1, indicate that the conditions of Lemma 8 are fulfilled, implying that $\beta \to 0$ as $t \to \infty$.

β strictly increases with d_0

To prove this, we will introduce Lemma 10, which needs a condition supported by Lemma 11, introduced afterwards.

Lemma 10 (β strictly increases with d_0)**.** *β strictly increases with d_0 if:*

1. *z_0 strictly increases with d_0,*
2. *$\delta^{max}(0, t) = 0$, $\forall t \in \mathbb{R}_{\geq 0}$,*
3. *β strictly decreases with t, $\forall d_0 \in \mathbb{R}_{\geq 0}$,*
4. *β continuous with t,*
5. *$\beta \to 0$ as $t \to \infty$.*

Proof. For a fixed t, the β strictly increases in d_0 if for any two numbers d_0^{a} and d_0^{b} (where $d_0^{\mathrm{a}} < d_0^{\mathrm{b}}$) within its domain, the condition $d_0^{\mathrm{a}} < d_0^{\mathrm{b}}$ implies $\beta^{\mathrm{a}}(d_0^{\mathrm{a}}, t) < \beta^{\mathrm{b}}(d_0^{\mathrm{b}}, t)$.

By condition 1, z_0 strictly increases with d_0, so we have $z_0^{\mathrm{a}}(d_0^{\mathrm{a}}) < z_0^{\mathrm{b}}(d_0^{\mathrm{b}})$. Moreover, since (any) z_0 is non-negative (see (C.4)), it follows that $z_0^{\mathrm{b}} > z_0^{\mathrm{a}} \geq 0$. Given condition 2, this indicates that $d_0^{\mathrm{b}} > 0$.

Recalling (C.1), we have

$$\beta^{\mathrm{b}} = z_0^{\mathrm{b}} + \int_0^t \dot{z}_s ds. \tag{C.8}$$

Since $d_0^{\mathrm{b}} > 0$, it follows from condition 3 that $\beta(d_0^{\mathrm{b}}, t)$ strictly decreases with respect to t (i.e., $\dot{z}_t < 0$). Furthermore, β is continuous with t and approaches zero as time

goes to infinity (conditions 4 and 5). This, combined with $z_0^{\mathrm{b}} > z_0^{\mathrm{a}}$, implies that there must exist a time $t^{\mathrm{a}} > 0$ such that

$$\beta^{\mathrm{b}} = \underbrace{z_0^{\mathrm{b}} + \int_0^{t^{\mathrm{a}}} \dot{z}_s ds}_{z_0^{\mathrm{a}}} + \int_{t^{\mathrm{a}}}^{t} \dot{z}_s ds. \tag{C.9}$$

Since $\dot{z}_t < 0$, starting from the same initial condition, a larger integration interval results in a smaller value of β. Therefore, we deduce that $\beta^{\mathrm{b}} > \beta^{\mathrm{a}}$, as β^{a} follows the same format as equation (C.9) (when replacing the terms equivalent to z_0^{a}, with z_0^{a}) but with an integral that starts at 0 instead of t^{a}, and $0 < t^{\mathrm{a}}$. Thus, under the given conditions, β strictly increases with respect to d_0. □

C

Lemma 11 (z_0 strictly increases with d_0)**.** *z_0 strictly increases with d_0 if:*

1. *$\delta(y_0, t)$ continuous with respect to t.*

Proof. For z_0 to strictly increase with d_0, for any two numbers d_0^{a} and d_0^{b} within its domain, the inequality $d_0^{\mathrm{a}} < d_0^{\mathrm{b}}$ must imply $z_0^{\mathrm{a}}(d_0^{\mathrm{a}}) < z_0^{\mathrm{b}}(d_0^{\mathrm{b}})$. From (C.4), we have $z_0 = \delta_0^{\max} + d_0$. Given that d_0 strictly increases with itself, it suffices to analyze the behavior of $\delta_0^{\max}$.

Recall that $\delta_0^{\max} = \max_{y_0 \in \mathcal{Y}_0} \left(\delta_{t+\Delta t}^{\max}(y_0, 0)\right)$ and that $\delta_{t+\Delta t}^{\max}(y_0, 0)$ computes the maximum over the set $\{\delta(y_0, s) : s \in [0, \Delta t]\}$. We will refer to this set as $\mathcal{W}(y_0)$. Importantly, $\mathcal{W}(y_0)$ contains $\delta(y_0, 0) = d_0$ for all y_0. Then, for any d_0^{a} and d_0^{b} satisfying $d_0^{\mathrm{a}} < d_0^{\mathrm{b}}$, the behavior of $\delta_0^{\max}$ can be studied in three different scenarios. Let $y_0^{\mathrm{a}} \in \mathcal{Y}_0(d_0^{\mathrm{a}})$ be the state that maximizes $\delta_{t+\Delta t}^{\max}$ for d_0^{a}, then:

i) $\boldsymbol{d_0^{\mathrm{a}} = \delta_{t+\Delta t}^{\max}(y_0^{\mathrm{a}}, 0)}$: This scenario occurs when d_0^{a} is the maximum of $\mathcal{W}(y_0^{\mathrm{a}})$. Then, if $y_0^{\mathrm{b}} \in \mathcal{Y}_0(d_0^{\mathrm{b}})$ is the state that maximizes $\delta_{t+\Delta t}^{\max}$ for d_0^{b}, and, hence, $d_0^{\mathrm{b}} \in \mathcal{W}(y_0^{\mathrm{b}})$, the maximum of $\mathcal{W}(y_0^{\mathrm{b}})$ cannot be lower than d_0^{b}. Consequently, since $d_0^{\mathrm{a}} < d_0^{\mathrm{b}}$, we get $\delta_{t+\Delta t}^{\max}(y_0^{\mathrm{a}}, 0) < \delta_{t+\Delta t}^{\max}(y_0^{\mathrm{b}}, 0)$. As both y_0^{a} and y_0^{b} are those that maximize $\delta_{t+\Delta t}^{\max}$ over y_0, for d_0^{a} and d_0^{b}, respectively, we conclude that $\delta_0^{\max}(d_0^{\mathrm{a}}) < \delta_0^{\max}(d_0^{\mathrm{b}})$.

ii) $\boldsymbol{d_0^{\mathrm{a}} < \delta_{t+\Delta t}^{\max}(y_0^{\mathrm{a}}, 0) \leq d_0^{\mathrm{b}}}$: This scenario occurs when the maximum of $\mathcal{W}(y_0^{\mathrm{a}})$ is not d_0^{a}, and d_0^{b} is greater than or equal to this maximum. Following a similar reasoning to the above, we can conclude that $\delta_0^{\max}(d_0^{\mathrm{a}}) \leq \delta_0^{\max}(d_0^{\mathrm{b}})$.

iii) $\boldsymbol{d_0^{\mathrm{a}} < d_0^{\mathrm{b}} < \delta_{t+\Delta t}^{\max}(y_0^{\mathrm{a}}, 0)}$: This scenario occurs when the maximum of $\mathcal{W}(y_0^{\mathrm{a}})$ is not d_0^{a}, and d_0^{b} is lower than this maximum. Provided that δ is continuous with t (condition 1), in the set $\mathcal{W}(y_0^{\mathrm{a}})$, since, for $t = 0$, $\delta = d_0^{\mathrm{a}}$, and for some $t^* > 0$, $\delta = \delta_{t+\Delta t}^{\max}(y_0^{\mathrm{a}}, 0)$, then we know there must exist a t^{b}, with $0 < t^{\mathrm{b}} < t^* \leq \Delta t$, where $\delta = d_0^{\mathrm{b}}$.

Now, observe that $t^* \in [t^{\mathrm{b}}, t^{\mathrm{b}} + \Delta t]$. Therefore, $\delta_{t+\Delta t}^{\max}(y_0^{\mathrm{a}}, t^{\mathrm{b}})$, which computes the maximum over $\{\delta(y_0^{\mathrm{a}}, s) : s \in [t^{\mathrm{b}}, t^{\mathrm{b}} + \Delta t]\}$, cannot be lower than $\delta_{t+\Delta t}^{\max}(y_0^{\mathrm{a}}, 0)$. Moreover, we can select y_t^{a} at time t^{b}, i.e., $y_{t^{\mathrm{b}}}^{\mathrm{a}}$, as a new

initial condition such that $\delta^{\max}_{t+\Delta t}(y^{\mathrm{a}}_{t^{\mathrm{b}}}, 0) = \delta^{\max}_{t+\Delta t}(y^{\mathrm{a}}_0, t^{\mathrm{b}})$. Since $\delta(y^{\mathrm{a}}_{t^{\mathrm{b}}}, 0) = d^{\mathrm{b}}_0$, we get that $y^{\mathrm{a}}_{t^{\mathrm{b}}} \in \mathcal{Y}_0(d^{\mathrm{b}}_0)$. Hence, even though $y^{\mathrm{a}}_{t^{\mathrm{b}}}$ might not maximize $\delta^{\max}_{t+\Delta t}$ for $\mathcal{Y}_0(d^{\mathrm{b}}_0)$, we know that the maximum, achived at y^{b}_0, must be at least greater than or equal to $\delta^{\max}_{t+\Delta t}(y^{\mathrm{a}}_{t^{\mathrm{b}}}, 0)$.

Consequently, we have that $\delta^{\max}_{t+\Delta t}(y^{\mathrm{b}}_0, 0) \geq \delta^{\max}_{t+\Delta t}(y^{\mathrm{a}}_{t^{\mathrm{b}}}, 0)$, and $\delta^{\max}_{t+\Delta t}(y^{\mathrm{a}}_{t^{\mathrm{b}}}, 0) = \delta^{\max}_{t+\Delta t}(y^{\mathrm{a}}_0, t^{\mathrm{b}}) \geq \delta^{\max}_{t+\Delta t}(y^{\mathrm{a}}_0, 0)$. Hence, $\delta^{\max}_{t+\Delta t}(y^{\mathrm{a}}_0, 0) \leq \delta^{\max}_{t+\Delta t}(y^{\mathrm{b}}_0, 0)$, and, therefore, $\delta^{\max}_0(d^{\mathrm{a}}_0) \leq \delta^{\max}_0(d^{\mathrm{b}}_0)$.

C

In summary, for any d^{a}_0 and d^{b}_0 with $d^{\mathrm{a}}_0 < d^{\mathrm{b}}_0$, it is always true that $\delta^{\max}_0(d^{\mathrm{a}}_0) \leq \delta^{\max}_0(d^{\mathrm{b}}_0)$, meaning $\delta^{\max}_0$ is non-decreasing with respect to d_0. As the sum of a non-decreasing function ($\delta^{\max}_0$) and a strictly increasing function (d_0) results in a strictly increasing function, we conclude that z_0 strictly increases with d_0. □

From Lemma 10, we can conclude that β strictly increases with respect to d_0, since assuming the conditions of Proposition 3, and that we have proven that β is continuous in t, Lemmas 11, 3, and 8 indicate that the conditions of Lemma 10 are fulfilled.

To summarize, we have demonstrated that all the requirements for $\beta(d_0, t)$ to be a class-$\mathcal{KL}$ function, and to serve as an upper bound for $\delta(y_0, t)$, are met for every $y_t \in \mathcal{L}$ and for all $t \in \mathbb{R}_{\geq 0}$. With this, our proof of Proposition 3 is complete. □

C.2. Learning on Spherical Manifolds

In this work, we employ unit quaternions to control orientation; therefore, we consider two distance functions that are relevant when learning spherical geometries: 1) great-circle distance, and 2) chordal distance.

- **Great-circle distance:** This is the distance along a *great circle*. A great circle is the largest circle that can be drawn on any given sphere, defining the shortest distance between two points. In the context of unit quaternions, which have a unitary norm, this distance is equivalent to the central angle α subtended by two points on the sphere. Hence, we can define it as

$$d_{\mathrm{g.c.}} = \alpha. \tag{C.10}$$

- **Chordal distance:** This is a distance that can be computed when an n-sphere is embedded in a higher-dimensional Euclidean space, i.e., $\mathcal{S}^n \subset \mathbb{R}^{n+1}$. Then, by computing the Euclidean distance in $\mathbb{R}^{n+1}$ between two points in $\mathcal{S}^n$, we *induce* a distance in $\mathcal{S}^n$, corresponding to the chordal distance [252–254]. As the name suggests, this distance is the length of the chord connecting two points in an n-sphere. Hence, it is defined as

$$d_{\mathrm{chord}} = 2r \sin(\alpha/2), \tag{C.11}$$

where r is the radius of the n-sphere.

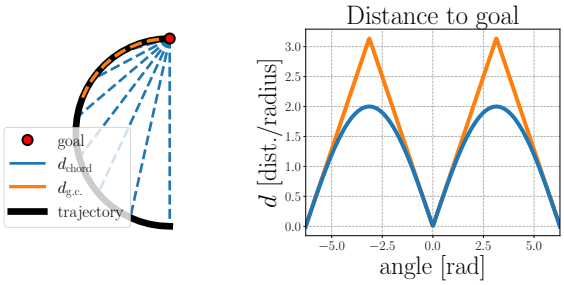


Figure C.2.: Left: A spherical trajectory towards a goal at the north pole. The chordal and great-circle distances at specific points are indicated as d_{chord} and $d_{\text{g.c.}}$, respectively. Right: Distances as a function of the central angle between points.

These distances are depicted in Fig. C.2. Since both define the same topology $\mathcal{S}^n$ they are considered to be equivalent, and can be utilized in PUMA to enforce stability when states are represented as unit quaternions. The great circle distance is especially suited to spherical spaces as it defines the *shortest path*, or the *geodesic*, between two points. Conversely, the chordal distance is straightforward to apply because it naturally arises when calculating the Euclidean distance at the output of ψ_θ. This is due to $\mathcal{L} \subset \mathbb{R}^m$, where $m > n$ represents the output size of ψ_θ.

C.2.1. Comments on local stability

Considering a one-dimensional sphere, it is notable that at $\alpha = \pi$ (with respect to the goal), both the great-circle distance and the chordal distance can decrease in two possible ways: by evolving either to the right or to the left (see Fig. C.2). Consequently, to ensure these distances decrease in the region around $\alpha = \pi$, one of these options should be selected. However, this would require the dynamical system to instantly change its direction at this point, rendering $f_\theta^{\mathcal{T}}$ discontinuous. Yet, in Theorem 5, we assume this function is continuous, as continuity is a prerequisite for the class-$\mathcal{KL}$ upper bound β to also be continuous, which is necessary to prove stability.

Therefore, by extending this idea to higher-dimensional spheres, this implies that when employing these metrics, we can only enforce local asymptotic stability for $\mathcal{S}^n \setminus \{p\}$, where p is the point at $\alpha = \pi$. Moreover, due to the continuity of $f_\theta^{\mathcal{T}}$, this point must correspond to a zero, and, therefore, represents an unstable equilibrium. This property is not a flaw in the great-circle distance or the chordal distance. Rather, it is a result of the topology of $\mathcal{S}^n$, which does not allow for the existence of a single stable equilibrium. This is a consequence of the Poincaré-Hopf theorem [255].

C.2.2. Pose Control

Until now, our discussion has centered on the applicability of our method for unit quaternions. However, practical control of a robot's pose requires simultaneous control of both its position (Euclidean) and orientation (non-Euclidean). This can be achieved by using a *product metric*, i.e., a metric resulting from the Cartesian product of spaces. In this case, we are looking at the product $\mathbb{R}^3 \times \mathcal{S}^3$.

A simple product metric is the sum of the metrics from each space in the product [225], e.g., $d_{\mathbb{R}^n} + d_{\mathcal{S}^n}$ for $\mathbb{R}^n \times \mathcal{S}^n$, where $d_{\mathbb{R}^n}$ is a distance in $\mathbb{R}^n$ and $d_{\mathcal{S}^n}$ is a distance in $\mathcal{S}^n$. However, it is not straightforward to do this in $\mathcal{L}$ without modifying the DNN structure. This is because the latent states y_t are an entangled representation of the robot states x_t, making it impossible to simply add together the distances of the Euclidean and non-Euclidean parts in the latent space.

Interestingly, when the product metric is computed using manifolds of identical topology, such as $\mathbb{R} \times \mathbb{R}$, the resulting metric is *equivalent* to the one obtained by directly computing the metric in the higher-dimensional space ($\mathbb{R}^2$ in this example) [225]. Hence, for such scenarios, there is no need to explicitly disentangle the states in $\mathcal{L}$; instead, we can compute the metric directly in the complete latent space. Our case, nevertheless, is more complex since the topologies of $\mathcal{S}^3$ and $\mathbb{R}^3$ are different. Fortunately, we found two ways of easily overcoming this limitation.

Pose in Euclidean space

We have observed that an Euclidean metric, the chordal distance, can generate spherical metrics in lower-dimensional manifolds. This becomes particularly relevant when a Euclidean metric is employed in $\mathbb{R}^m$, where m exceeds the dimensionality of our state space (6 in our case). In this scenario, the metric is equivalent to $d_{\mathbb{R}^3} + d_{\mathbb{R}^{m'}}$ for $m' > 3$ and $3 + m' = m$. Given that $\mathcal{S}^3$ can be induced inside $\mathbb{R}^{m'}$, this suggests that $\mathbb{R}^3 \times \mathcal{S}^3$ can be induced in $\mathbb{R}^m$ when using the Euclidean distance in this space.

Pose in spherical space

Finally, we also note that the great-circle distance can be employed to achieve the same objective. Suppose we compute this distance in $\mathcal{S}^6$. Then, it would be equivalent to $d_{\mathcal{S}^3} + d_{\mathcal{S}^3}$. Interestingly, a diffeomorphism can be found between $\mathbb{R}^n$ and a subset of $\mathcal{S}^n$, e.g., the *stereographic projection* [256]. This implies that it is feasible to use the metric $d_{\mathcal{S}^3}$ in $\mathcal{L}$, and find a valid representation for $\mathbb{R}^3$ within a subspace of $\mathcal{S}^3$. Consequently, the product space $\mathbb{R}^3 \times \mathcal{S}^3$ can be represented within a subset of $\mathcal{S}^6$.

C.3. Hyperparameter Optimization

To optimize the hyperparameters of the different variations of PUMA employed in the LASA and LAIR datasets, we utilized the Tree Parzen Estimator [237]. This optimization method builds a probability model that facilitates the selection of

Table C.1.: Hyperparameter optimization results of the different variations of PUMA.

Hyperparameter	Opt.?	Hand-tuned value / initial opt. guess				Optimized value			
		Euc.	Sph.	Euc. (2nd order)	Sph. (2nd order)	Euc.	Sph.	Euc. (2nd order)	Sph. (2nd order)
CONDOR									
Stability loss margin (m)	✓	1.250e-8	1.250e-4	1.250e-4	1.250e-4	5.921e-3	3.012e-05	2.424e-08	2.919e-7
Triplet imitation loss weight (λ)	✓	1	1	1	1	1.315e-1	3.496	1.022e-1	4.473e-1
Window size imitation ($\mathcal{H}^i$)	✓	14	14	14	14	13	13	14	14
Window size stability ($\mathcal{H}^s$)	✓	4	1	1	1	11	13	11	11
Batch size imitation ($\mathcal{B}^i$)	✗	250	250	250	250	-	-	-	-
Batch size stability ($\mathcal{B}^s$)	✗	250	250	250	250	-	-	-	-
Neural Network									
Optimizer	✗	Adam	Adam	Adam	Adam	-	-	-	-
Number of iterations	✗	40000	40000	40000	40000	-	-	-	-
Learning rate	✓	1e-4	1e-4	1e-4	1e-4	9.784e-5	8.574e-4	1.670e-4	1.245e-4
Activation function	✗	GELU	GELU	GELU	GELU	-	-	-	-
Num. layers (ψ_θ, ϕ_θ)	✗	(3, 3)	(3, 3)	(3, 3)	(3, 3)	-	-	-	-
Neurons/hidden layer	✗	300	300	300	300	-	-	-	-
Layer normalization	✗	yes	yes	yes	yes	-	-	-	-

the most promising set of hyperparameters in each optimization round. For the implementation of the Tree Parzen Estimator, we used the Optuna API [238].

To evaluate the selected sets of hyperparameters, we employ a loss function $\mathcal{L}_{\text{hyper}}$ composed of two components. The first component, $\mathcal{L}_{\text{acc}}$, assesses the accuracy of the trained model. This is done by computing the RMSE between the demonstrations and the trajectories simulated by the learned model, in a manner similar to the approach used in the experiments section. The second component, $\mathcal{L}_{\text{goal}}$, quantifies the average distance between the final points of trajectories, simulated using the learned model, and the target goal. Thus, we have:

$$\mathcal{L}_{\text{hyper}} = \mathcal{L}_{\text{acc}} + \gamma \mathcal{L}_{\text{goal}} \tag{C.12}$$

where γ is a weighting factor. After initial tests, we settled on $\gamma = 3.5$.

To make the computationally intensive process of optimizing hyperparameters more feasible, we employed six strategies: 1) reducing the overhead of the objective function, 2) limiting the size of the evaluation set, 3) utilizing Bayesian optimization, 4) applying pruning techniques, 5) selecting a subset of hyperparameters for optimization, and 6) employing an optimization range. For details on the first five strategies, the reader is referred to [156]. The sixth strategy consists of restricting the hyperparameter search to a predefined range.

Table C.1 presents the results of this optimization process. We can observe the hyperparameters that were optimized, their initial values (chosen based on preliminary tests), and their final optimized values. The ranges for hyperparameter optimization are as follows: 1) $m : [1e-9, 1e-1]$, 2) $\lambda : [1e-1, 10]$, 3) $\mathcal{H}^i : [1, 14]$, 4) $\mathcal{H}^s : [1, 14]$, and 5) learning rate: $[1e-5, 1e-3]$. Note that the variations using the boundary loss are not included in the table, since in those cases the same hyperparameters were employed and the boundary loss was added with a weight of 0.001. The same holds for the behavioral cloning case. Lastly, regarding Sec. 6.4.4 it is important to note that in every experiment involving spherical state spaces, the dynamical system was kept within the manifold by normalizing the forward Euler integration output.

Bibliography

[1] C. Cheung, S. Baker, T. Maxwell, and B. Plackett. "Growth in AI and robotics research accelerates". In: *Nature* (2022).

[2] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. "An algorithmic perspective on imitation learning". In: *Foundations and Trends® in Robotics* 7.1-2 (2018), pp. 1–179.

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[4] S. Chernova and A. L. Thomaz. *Robot learning from human teachers*. Vol. 8. 3. Morgan & Claypool Publishers, 2014, pp. 1–121.

[5] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard. "Recent advances in robot learning from demonstration". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (2020), pp. 297–330.

[6] M. Bain and C. Sammut. "A Framework for Behavioural Cloning." In: *Machine Intelligence 15*. 1995, pp. 103–129.

[7] M. Sugiyama. *Introduction to statistical machine learning*. Morgan Kaufmann, 2015.

[8] S. Ross, G. Gordon, and D. Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 627–635.

[9] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[10] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 5628–5635.

[11] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots. "Imitation learning for agile autonomous driving". In: *The International Journal of Robotics Research* 39.2-3 (2020), pp. 286–302.

[12] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning". In: *Proceedings of the 5th Conference on Robot Learning*. Ed. by A. Faust, D. Hsu, and G. Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2022, pp. 991–1002.

[13] S. Chernova and M. Veloso. “Interactive policy learning through confidence-based autonomy”. In: *Journal of Artificial Intelligence Research* 34 (2009), pp. 1–25.

[14] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza. “Power to the people: The role of humans in interactive machine learning”. In: *AI Magazine* 35.4 (2014), pp. 105–120.

[15] C. Celemin and J. Ruiz-del-Solar. “An interactive framework for learning continuous actions policies based on corrective feedback”. In: *Journal of Intelligent & Robotic Systems* 95.1 (2019), pp. 77–97.

[16] J. Spencer, S. Choudhury, M. Barnes, M. Schmittle, M. Chiang, P. Ramadge, and S. Srinivasa. “Expert intervention learning”. In: *Autonomous Robots* 46.1 (2022), pp. 99–113.

[17] L. Alzubaidi, J. Bai, A. Al-Sabaawi, J. Santamarıéa, A. Albahri, B. S. N. Al-dabbagh, M. A. Fadhel, M. Manoufali, J. Zhang, A. H. Al-Timemy, *et al.* “A survey on deep learning tools dealing with data scarcity: definitions, challenges, solutions, tips, and applications”. In: *Journal of Big Data* 10.1 (2023), p. 46.

[18] Y. Zhang, B. Kang, B. Hooi, S. Yan, and J. Feng. “Deep long-tailed learning: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).

[19] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. “Relational inductive biases, deep learning, and graph networks”. In: *arXiv* (2018).

[20] N. Figueroa and A. Billard. “Locally active globally stable dynamical systems: Theory, learning, and experiments”. In: *The International Journal of Robotics Research* (2022), p. 02783649211030952.

[21] R. Pérez-Dattari, C. Celemin, J. Ruiz-del-Solar, and J. Kober. “Continuous Control for High-Dimensional State Spaces: An Interactive Learning Approach”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 7611–7617.

[22] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. “CARLA: An open urban driving simulator”. In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.

[23] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.* “On the opportunities and risks of foundation models”. In: *arXiv preprint arXiv:2108.07258* (2021).

[24] OpenAI. *GPT-4 Technical Report*. 2023.

[25] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.* "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[26] R. Firoozi, J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman, *et al.* "Foundation models in robotics: Applications, challenges, and the future". In: *arXiv preprint arXiv:2312.07843* (2023).

[27] C. Celemin*, R. Pérez-Dattari*, E. Chisari*, G. Franzese*, L. de Souza Rosa, R. Prakash, Z. Ajanović, M. Ferraz, A. Valada, and J. Kober. "Interactive Imitation Learning in Robotics: A Survey". In: *Foundations and Trends® in Robotics* 10.1–2 (2022).

[28] R. Bellman. "A Markovian decision process". In: *Journal of Mathematics and Mechanics* (1957), pp. 679–684.

[29] S. J. Russell and P. Norvig. *Artificial Intelligence : A Modern Approach*. Malaysia and Pearson Education Limited, 2016.

[30] R. A. Howard. "Dynamic programming and markov processes". In: (1960).

[31] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1 (1998), pp. 99–134. ISSN: 0004-3702.

[32] S. Ross and D. Bagnell. "Efficient reductions for imitation learning". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 661–668.

[33] N. Wilde, E. Biyik, D. Sadigh, and S. L. Smith. "Learning Reward Functions from Scale Feedback". In: *Proceedings of the 5th Conference on Robot Learning*. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2022, pp. 353–362.

[34] A. Wilson, A. Fern, and P. Tadepalli. "A Bayesian Approach for Policy Learning from Trajectory Preference Queries". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1133–1141.

[35] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[36] M. K. Ho, M. L. Littman, F. Cushman, and J. L. Austerweil. "Teaching with rewards and punishments: Reinforcement or communication?" In: *CogSci*. 2015.

[37] F. Liese and I. Vajda. “On divergences and informations in statistics and information theory”. In: *IEEE Transactions on Information Theory* 52.10 (2006), pp. 4394–4412.

[38] S. K. S. Ghasemipour, R. Zemel, and S. Gu. “A divergence minimization perspective on imitation learning methods”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 1259–1277.

[39] L. Ke, S. Choudhury, M. Barnes, W. Sun, G. Lee, and S. Srinivasa. “Imitation learning as f-divergence minimization”. In: *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*. Springer. 2021, pp. 313–329.

[40] C. M. Bishop. “Pattern recognition”. In: *Machine learning* 128.9 (2006).

[41] Ç. Meriçli, M. Veloso, and H. L. Akin. “Complementary humanoid behavior shaping using corrective demonstration”. In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. 2010, pp. 334–339.

[42] Ç. Meriçli, M. Veloso, and H. L. Akın. “Task Refinement for Autonomous Robots Using Complementary Corrective Human Feedback”. In: *International Journal of Advanced Robotic Systems* 8.2 (2011), p. 16.

[43] C. Mericli. “Multi-Resolution Model Plus Correction Paradigm for Task and Skill Refinement on Autonomous Robots”. PhD thesis. Citeseer, 2011.

[44] J. Zhang and K. Cho. *Query-Efficient Imitation Learning for End-to-End Autonomous Driving*. 2016.

[45] K. Menda, K. Driggs-Campbell, and M. J. Kochenderfer. “EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019, pp. 5041–5048.

[46] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. Kochenderfer. “HG-DAgger: Interactive Imitation Learning with Human Experts”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8077–8083.

[47] R. Hoque, A. Balakrishna, C. Putterman, M. Luo, D. S. Brown, D. Seita, B. Thananjeyan, E. Novoseller, and K. Goldberg. “LazyDAgger: Reducing Context Switching in Interactive Imitation Learning”. In: *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. 2021, pp. 502–509.

[48] R. Hoque, A. Balakrishna, E. Novoseller, A. Wilcox, D. S. Brown, and K. Goldberg. “ThriftyDAgger: Budget-Aware Novelty and Risk Gating for Interactive Imitation Learning”. In: *Proceedings of the 5th Conference on Robot Learning*. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2022, pp. 598–608.

[49] M. Laskey, S. Staszak, W. Y.-S. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg. "SHIV: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 462–469.

[50] M. Laskey, C. Chuck, J. Lee, J. Mahler, S. Krishnan, K. Jamieson, A. Dragan, and K. Goldberg. "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 358–365.

[51] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese. *Human-in-the-Loop Imitation Learning using Remote Teleoperation*. 2020.

[52] E. Chisari, T. Welschehold, J. Boedecker, W. Burgard, and A. Valada. "Correct me if i am wrong: Interactive learning for robotic manipulation". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 3695–3702.

[53] J. Spencer, S. Choudhury, M. Barnes, M. Schmittle, M. Chiang, P. Ramadge, and S. Srinivasa. "Learning from interventions". In: *Robotics: Science and Systems (RSS)*. 2020.

[54] J. Luo, O. Sushkov, R. Pevceviciute, W. Lian, C. Su, M. Vecerik, N. Ye, S. Schaal, and J. Scholz. "Robust Multi-Modal Policies for Industrial Assembly via Reinforcement Learning and Demonstrations: A Large-Scale Study". In: *Robotics: Science and Systems XVII, 2021*. 2021.

[55] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards". In: *arXiv preprint arXiv:1707.08817* (2017).

[56] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich. "Efficiently Combining Human Demonstrations and Interventions for Safe Training of Autonomous Systems in Real-Time". In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. ISBN: 978-1-57735-809-1.

[57] G. Kahn, P. Abbeel, and S. Levine. "LaND: Learning to Navigate From Disengagements". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 1872–1879.

[58] J. DelPreto, J. I. Lipton, L. Sanneman, A. J. Fay, C. Fourie, C. Choi, and D. Rus. "Helping robots learn: a human-robot master-apprentice model using demonstrations via virtual reality teleoperation". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 10226–10233.

[59] B. D. Argall, B. Browning, and M. Veloso. “Learning robot motion control with demonstration and advice-operators”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2008, pp. 399–404.

[60] B. D. Argall. “Learning mobile robot motion control from demonstration and corrective feedback”. PhD thesis. Carnegie Mellon University, 2009.

[61] B. D. Argall, B. Browning, and M. M. Veloso. “Teacher feedback to scaffold and refine demonstrated motion primitives on a mobile robot”. In: *Robotics and Autonomous Systems* 59.3 (2011), pp. 243–255. ISSN: 0921-8890.

[62] Ç. Meriçli and M. Veloso. “Improving Biped Walk Stability Using Real-Time Corrective Human Feedback”. In: *RoboCup 2010: Robot Soccer World Cup XIV*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 194–205. ISBN: 978-3-642-20217-9.

[63] M. P. Deisenroth, G. Neumann, J. Peters, *et al.* “A survey on policy search for robotics”. In: *Foundations and Trends® in Robotics* 2.1–2 (2013), pp. 1–142.

[64] R. Perez, C. Celemin, J. Ruiz-del-Solar, and J. Kober. “Interactive Learning with Corrective Feedback for Policies based on Deep Neural Networks”. In: *International Symposium on Experimental Robotics*. Springer. 2018.

[65] R. Pérez-Dattari, C. Celemin, G. Franzese, J. Ruiz-del-Solar, and J. Kober. “Interactive learning of temporal features for control: Shaping policies and state representations from human feedback”. In: *IEEE Robotics & Automation Magazine* 27.2 (2020), pp. 46–54.

[66] C. Celemin, G. Maeda, J. Ruiz-del-Solar, J. Peters, and J. Kober. “Reinforcement learning of motor skills using policy search and human corrective advice”. In: *The International Journal of Robotics Research* 38.14 (2019), pp. 1560–1580.

[67] S. Jauhri, C. Celemin, and J. Kober. “Interactive Imitation Learning in State-Space”. In: *Proceedings of the 2020 Conference on Robot Learning*. Vol. 155. Proceedings of Machine Learning Research. PMLR, 2021, pp. 682–692.

[68] F. Torabi, G. Warnell, and P. Stone. “Behavioral cloning from observation”. In: *arXiv preprint arXiv:1805.01954* (2018).

[69] B. D. Argall, E. L. Sauser, and A. G. Billard. “Tactile Guidance for Policy Adaptation”. In: *Foundations and Trends® in Robotics* 1.2 (2011), pp. 79–133. ISSN: 1935-8253.

[70] M. Ewerton, G. Maeda, G. Kollegger, J. Wiemeyer, and J. Peters. “Incremental imitation learning of context-dependent motor skills”. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. 2016, pp. 351–358.

[71] G. Canal, G. Alenyà, and C. Torras. “Personalization Framework for Adaptive Robotic Feeding Assistance”. In: *Social Robotics*. Cham: Springer International Publishing, 2016, pp. 22–31. ISBN: 978-3-319-47437-3.

[72] A. Jain, B. Wojcik, T. Joachims, and A. Saxena. “Learning Trajectory Preferences for Manipulators via Iterative Improvement”. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 575–583.

[73] A. Jain, S. Sharma, T. Joachims, and A. Saxena. “Learning preferences for manipulation tasks from online coactive feedback”. In: *The International Journal of Robotics Research* 34.10 (2015), pp. 1296–1313.

[74] A. Bajcsy, D. P. Losey, M. K. O’Malley, and A. D. Dragan. “Learning robot objectives from physical human interaction”. In: *Conference on Robot Learning*. PMLR. 2017, pp. 217–226.

[75] A. Bajcsy, D. P. Losey, M. K. O’Malley, and A. D. Dragan. “Learning from physical human corrections, one feature at a time”. In: *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*. 2018, pp. 141–149.

[76] D. P. Losey, A. Bajcsy, M. K. O’Malley, and A. D. Dragan. “Physical interaction as communication: Learning robot objectives online from human corrections”. In: *The International Journal of Robotics Research* 41.1 (2022), pp. 20–44.

[77] A. L. Thomaz, G. Hoffman, and C. Breazeal. “Reinforcement learning with human teachers: Understanding how people want to teach robots”. In: *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*. IEEE. 2006, pp. 352–357.

[78] R. Toris, H. B. Suay, and S. Chernova. “A practical comparison of three robot learning from demonstration algorithms”. In: *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE. 2012, pp. 261–262.

[79] P. Koppol, H. Admoni, and R. Simmons. “Interaction considerations in learning from humans”. In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2021.

[80] S. Levine, A. Kumar, G. Tucker, and J. Fu. “Offline reinforcement learning: Tutorial, review, and perspectives on open problems”. In: *arXiv preprint arXiv:2005.01643* (2020).

[81] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. “Dart: Noise injection for robust imitation learning”. In: *Conference on robot learning*. PMLR. 2017, pp. 143–156.

[82] D. Arumugam, J. K. Lee, S. Saskin, and M. L. Littman. *Deep Reinforcement Learning from Policy-Dependent Human Feedback*. 2019.

[83] A. Balakrishna, B. Thananjeyan, J. Lee, F. Li, A. Zahed, J. E. Gonzalez, and K. Goldberg. “On-policy robot imitation learning from a converging supervisor”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 24–41.

[84] G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. Citeseer, 1994.

[85] C. J. C. H. Watkins. "Learning from delayed rewards". In: (1989).

[86] A. Mahmood. "Incremental off-policy reinforcement learning algorithms". In: (2017).

[87] D. Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[88] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo method*. Vol. 10. John Wiley & Sons, 2016.

[89] J. Hammersley and D. Handscomb. "Monte carlo methods, methuen & co". In: *Ltd., London* 40 (1964).

[90] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor, and M. L. Littman. "Interactive learning from policy-dependent human feedback". In: *34th International Conference on Machine Learning - Volume 70*. JMLR. org. 2017, pp. 2285–2294.

[91] W. B. Knox and P. Stone. "Tamer: Training an agent manually via evaluative reinforcement". In: *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*. IEEE. 2008, pp. 292–297.

[92] R. Zhang, F. Torabi, L. Guan, D. H. Ballard, and P. Stone. "Leveraging human guidance for deep reinforcement learning tasks". In: *arXiv preprint arXiv:1909.09906* (2019).

[93] W. B. Knox and P. Stone. "Interactively shaping agents via human reinforcement: The TAMER framework". In: *Fifth International Conference on Knowledge Capture*. ACM. 2009, pp. 9–16.

[94] T. Degris, M. White, and R. Sutton. "Off-Policy Actor-Critic". In: *International Conference on Machine Learning*. 2012.

[95] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. "Deep reinforcement learning from human preferences". In: *Advances in Neural Information Processing Systems*. 2017, pp. 4299–4307.

[96] C. Celemin and J. Ruiz-del-Solar. "An Interactive Framework for Learning Continuous Actions Policies Based on Corrective Feedback". In: *Journal of Intelligent & Robotic Systems* (2018), pp. 1–21.

[97] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.* "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[98] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[99] W. Böhmer, J. T. Springenberg, J. Boedecker, M. Riedmiller, and K. Obermayer. “Autonomous learning of state representations for control: An emerging field aims to autonomously learn state representations for reinforcement learning agents from their real-world sensor observations”. In: *KI-Künstliche Intelligenz* 29.4 (2015), pp. 353–362.

[100] T. Lesort, N. Dıéaz-Rodrıéguez, J.-F. Goudou, and D. Filliat. “State representation learning for control: An overview”. In: *Neural Networks* 108 (2018), pp. 379–392.

[101] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.* “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), p. 529.

[102] M. Hausknecht and P. Stone. “Deep Recurrent Q-learning for Partially Observable MDPs”. In: *2015 AAAI Fall Symposium Series*. 2015.

[103] G. Lample and D. S. Chaplot. “Playing FPS games with deep reinforcement learning”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[104] D. Ha and J. Schmidhuber. “Recurrent world models facilitate policy evolution”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 2450–2462.

[105] A. Zhang, H. Satija, and J. Pineau. “Decoupling dynamics and reward for transfer learning”. In: *arXiv preprint arXiv:1804.10689* (2018).

[106] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. “OpenAI gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[107] I. Mason, S. Starke, H. Zhang, H. Bilen, and T. Komura. “Few-shot Learning of Homogeneous Human Locomotion Styles”. In: *Computer Graphics Forum*. Vol. 37. 7. 2018, pp. 143–153.

[108] H. Zhang, S. Starke, T. Komura, and J. Saito. “Mode-adaptive neural networks for quadruped motion control”. In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–11.

[109] R. Pérez-Dattari*, B. Brito*, O. de Groot, J. Kober, and J. Alonso-Mora. “Visually-guided motion planning for autonomous driving from interactive demonstrations”. In: *Engineering Applications of Artificial Intelligence* 116 (2022).

[110] P. Veličković and C. Blundell. “Neural Algorithmic Reasoning”. In: *Patterns* 2.7 (2021), p. 100273.

[111] I. A. Zamfirache, R.-E. Precup, R.-C. Roman, and E. M. Petriu. “Reinforcement Learning-based control using Q-learning and gravitational search algorithm with experimental validation on a nonlinear servo system”. In: *Information Sciences* 583 (2022), pp. 99–120.

[112] L. Ferranti, B. Brito, E. Pool, Y. Zheng, R. M. Ensing, R. Happee, B. Shyrokau, J. Kooij, J. Alonso-Mora, and D. M. Gavrila. "SafeVRU: A Research Platform for the Interaction of Self-Driving Vehicles with Vulnerable Road Users". In: *2019 IEEE Intelligent Vehicles Symposium* (2019).

[113] A. Waytz, J. Heafner, and N. Epley. "The mind in the machine: Anthropomorphism increases trust in an autonomous vehicle". In: *Journal of Experimental Social Psychology* 52 (2014), pp. 113–117.

[114] S. Kolekar, J. de Winter, and D. Abbink. "Human-like driving behaviour emerges from a risk-based driver model". In: *Nature communications* 11.1 (2020), pp. 1–13.

[115] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[116] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. "A survey of motion planning and control techniques for self-driving urban vehicles". In: *IEEE Transactions on intelligent vehicles* 1.1 (2016), pp. 33–55.

[117] C. R. Baker and J. M. Dolan. "Traffic interaction in the urban challenge: Putting Boss on its best behavior". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS* (2008), pp. 1752–1758.

[118] C. Hubmann, M. Becker, D. Althoff, D. Lenz, and C. Stiller. "Decision making for autonomous driving considering interaction and uncertain prediction of surrounding vehicles". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 1671–1678.

[119] B. Zhou, W. Schwarting, D. Rus, and J. Alonso-Mora. "Joint Multi-Policy Behavior Estimation and Receding-Horizon Trajectory Planning for Automated Urban Driving". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 2388–2394.

[120] P. Cai, Y. Luo, D. Hsu, and W. S. Lee. "HyP-DESPOT: A hybrid parallel algorithm for online planning under uncertainty". In: *The International Journal of Robotics Research* 40.2-3 (2021), pp. 558–573.

[121] B. Brito, B. Floor, L. Ferranti, and J. Alonso-Mora. "Model predictive contouring control for collision avoidance in unstructured dynamic environments". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4459–4466.

[122] M. Everett, Y. F. Chen, and J. P. How. "Collision Avoidance in Pedestrian-Rich Environments With Deep Reinforcement Learning". In: *IEEE Access* 9 (2021), pp. 10357–10377.

[123] T. Fan, P. Long, W. Liu, and J. Pan. "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios." In: *The International Journal of Robotics Research* 39.7 (2020), pp. 856–892.

[124] W. Schwarting, T. Seyde, I. Gilitschenski, L. Liebenwein, R. Sander, S. Karaman, and D. Rus. “Deep latent competition: Learning to race using visual control policies in latent space”. In: *Proceedings of the 2020 Conference on Robot Learning*. 2021, pp. 1855–1870.

[125] J. Kulhánek, E. Derner, and R. Babuška. “Visual Navigation in Real-World Indoor Environments Using End-to-End Deep Reinforcement Learning”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4345–4352.

[126] Y. F. Chen, M. Everett, M. Liu, and J. P. How. “Socially aware motion planning with deep reinforcement learning”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 1343–1350.

[127] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel. “Adversarial attacks on neural network policies”. In: *arXiv preprint arXiv:1702.02284* (2017).

[128] Z. Qiao, J. Schneider, and J. M. Dolan. “Behavior Planning at Urban Intersections through Hierarchical Reinforcement Learning”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 2667–2673.

[129] Y. Song and D. Scaramuzza. “Learning High-Level Policies for Model Predictive Control”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 7629–7636.

[130] V. Tolani, S. Bansal, A. Faust, and C. Tomlin. “Visual navigation among humans with optimal control as a supervisor”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2288–2295.

[131] J. Huang, S. Xie, J. Sun, Q. Ma, C. Liu, D. Lin, and B. Zhou. “Learning a decision module by imitating driver’s control behaviors”. In: *Conference on Robot Learning*. PMLR. 2021, pp. 1–10.

[132] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.* “End to end learning for self-driving cars”. In: *arXiv preprint arXiv:1604.07316* (2016).

[133] W. Zhan, L. Sun, D. Wang, H. Shi, A. Clausse, M. Naumann, J. Kummerle, H. Konigshof, C. Stiller, A. de La Fortelle, *et al.* “Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps”. In: *arXiv preprint arXiv:1910.03088* (2019).

[134] A. Prakash, A. Behl, E. Ohn-Bar, K. Chitta, and A. Geiger. “Exploring data aggregation in policy learning for vision-based urban autonomous driving”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11763–11773.

[135] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. “Exploring the limitations of behavior cloning for autonomous driving”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 9329–9338.

[136] J. Ho and S. Ermon. “Generative Adversarial Imitation Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016. (Visited on 02/21/2022).

[137] N. R. Waytowich, V. G. Goecks, and V. J. Lawhern. “Cycle-of-Learning for Autonomous Systems from Human Interaction”. In: (2018).

[138] T. Ablett, F. Marić, and J. Kelly. “Fighting Failures with FIRE: Failure Identification to Reduce Expert Burden in Intervention-Based Learning”. In: *arXiv preprint arXiv:2007.00245* (2020).

[139] J. Abramson, A. Ahuja, I. Barr, A. Brussee, F. Carnevale, M. Cassin, R. Chhaparia, S. Clark, B. Damoc, A. Dudzik, *et al.* “Imitating interactive intelligence”. In: *arXiv preprint arXiv:2012.05672* (2020).

[140] B. Berg, B. Brito, J. Alonso-Mora, and M. Alirezaei. “Curvature Aware Motion Planning with Closed-Loop Rapidly-exploring Random Trees”. In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. 2021.

[141] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.* “Rectifier nonlinearities improve neural network acoustic models”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*. Vol. 30. 1. 2013.

[142] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari. “FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs”. In: *International Journal of Control* 93.1 (2020), pp. 13–29.

[143] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[144] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.* “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[145] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[146] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos. “Image segmentation using deep learning: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[147] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu. “A survey on deep transfer learning”. In: *International conference on artificial neural networks*. Springer. 2018, pp. 270–279.

[148] V. Badrinarayanan, A. Kendall, and R. Cipolla. “SegNet: A deep convolutional encoder-decoder architecture for image segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[149] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. “DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[150] K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[151] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[152] S. Chernova and M. Veloso. “Learning equivalent action choices from demonstration”. In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 1216–1221.

[153] P. Valletta, R. Pérez-Dattari, and J. Kober. “Imitation Learning with Inconsistent Demonstrations through Uncertainty-based Data Manipulation”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021.

[154] M. Vitelli, Y. Chang, Y. Ye, M. Wołczyk, B. Osiński, M. Niendorf, H. Grimmett, Q. Huang, A. Jain, and P. Ondruska. “SafetyNet: Safe planning for real-world self-driving vehicles using machine-learned policies”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 897–904.

[155] B. Brito, H. Zhu, W. Pan, and J. Alonso-Mora. “Social-VRNN: One-Shot Multi-modal Trajectory Prediction for Interacting Pedestrians”. In: *Conference on Robot Learning* (2020).

[156] R. Pérez-Dattari and J. Kober. “Stable Motion Primitives via Imitation and Contrastive Learning”. In: *IEEE Transactions on Robotics* (2023).

[157] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. “Dynamical movement primitives: learning attractor models for motor behaviors”. In: *Neural computation* 25.2 (2013), pp. 328–373.

[158] S. M. Khansari-Zadeh and A. Billard. “Learning stable nonlinear dynamical systems with gaussian mixture models”. In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957.

[159] M. A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, and N. Ratliff. "Euclideanizing Flows: Diffeomorphic Reduction for Learning Stable Dynamical Systems". In: *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. Vol. 120. Proceedings of Machine Learning Research. PMLR, 2020, pp. 630–639.

[160] J. Kober, M. Glisson, and M. Mistry. "Playing catch and juggling with a humanoid robot". In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE. 2012, pp. 875–881.

[161] V. Sindhwani, S. Tu, and M. Khansari. "Learning contracting vector fields for stable imitation learning". In: *arXiv preprint arXiv:1804.04878* (2018).

[162] N. Perrin and P. Schlehuber-Caissier. "Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems". In: *Systems & Control Letters* 96 (2016), pp. 51–59.

[163] J. Urain, M. Ginesi, D. Tateo, and J. Peters. "ImitationFlow: Learning Deep Stable Stochastic Dynamic Systems by Normalizing Flows". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 5231–5237.

[164] S. M. Khansari-Zadeh and A. Billard. "Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions". In: *Robotics and Autonomous Systems* 62.6 (2014), pp. 752–765.

[165] A. Lemme, K. Neumann, R. F. Reinhart, and J. J. Steil. "Neural learning of vector fields for encoding stable dynamical systems". In: *Neurocomputing* 141 (2014), pp. 3–14.

[166] M. Awad and R. Khanna. *Efficient learning machines: Theories, concepts, and applications for engineers and system designers*. Springer nature, 2015.

[167] S. Chopra, R. Hadsell, and Y. LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 539–546.

[168] M. Kaya and H. Ş. Bilge. "Deep metric learning: A survey". In: *Symmetry* 11.9 (2019), p. 1066.

[169] G. Li, Z. Jin, M. Volpp, F. Otto, R. Lioutikov, and G. Neumann. "ProDMPs: A Unified Perspective on Dynamic and Probabilistic Movement Primitives". In: *arXiv preprint arXiv:2210.01531* (2022).

[170] H. Ben Amor, G. Neumann, S. Kamthe, O. Kroemer, and J. Peters. "Interaction primitives for human-robot cooperation tasks". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 2831–2837.

[171] A. Pervez, Y. Mao, and D. Lee. "Learning deep movement primitives using convolutional neural networks". In: *2017 IEEE-RAS 17th international conference on humanoid robotics (Humanoids)*. IEEE. 2017, pp. 191–197.

[172] B. Ridge, A. Gams, J. Morimoto, A. Ude, *et al.* "Training of deep neural networks for the generation of dynamic movement primitives". In: *Neural Networks* 127 (2020), pp. 121–131.

[173] S. Bahl, M. Mukadam, A. Gupta, and D. Pathak. "Neural dynamic policies for end-to-end sensorimotor learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5058–5069.

[174] S. Calinon, A. Pistillo, and D. G. Caldwell. "Encoding the time and space constraints of a task in explicit-duration hidden Markov model". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 3413–3418.

[175] J. Duan, Y. Ou, J. Hu, Z. Wang, S. Jin, and C. Xu. "Fast and stable learning of dynamical systems based on extreme learning machine". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.6 (2017), pp. 1175–1185.

[176] N. Figueroa and A. Billard. "A Physically-Consistent Bayesian Non-Parametric Mixture Model for Dynamical System Learning." In: *CoRL*. 2018, pp. 927–946.

[177] H. C. Ravichandar, I. Salehi, and A. P. Dani. "Learning Partially Contracting Dynamical Systems from Demonstrations." In: *CoRL*. 2017, pp. 369–378.

[178] K. Neumann and J. J. Steil. "Learning robot motions with stable dynamical systems under diffeomorphic transformations". In: *Robotics and Autonomous Systems* 70 (2015), pp. 1–15.

[179] H. Ravichandar and A. Dani. "Learning contracting nonlinear dynamics from human demonstration for robot motion planning". In: *Dynamic Systems and Control Conference*. Vol. 57250. American Society of Mechanical Engineers. 2015, V002T27A008.

[180] C. Blocher, M. Saveriano, and D. Lee. "Learning stable dynamical systems using contraction theory". In: *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE. 2017, pp. 124–129.

[181] F. Schroff, D. Kalenichenko, and J. Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[182] H. K. Khalil. *Nonlinear control*. Vol. 406. Pearson New York, 2015.

[183] J. K. Hunter. "Introduction to dynamical systems". In: *UCDavis Mathematics MAT A* 207 (2011), p. 2011.

[184] C. M. Legaard, T. Schranz, G. Schweiger, J. Drgoňa, B. Falay, C. Gomes, A. Iosifidis, M. Abkar, and P. G. Larsen. "Constructing neural network-based models for simulating dynamical systems". In: *arXiv preprint arXiv:2111.01495* (2021).

[185] N. Lambert, K. Pister, and R. Calandra. "Investigating Compounding Prediction Errors in Learned Dynamics Models". In: *arXiv preprint arXiv:2203.09637* (2022).

[186] G. Bontempi, S. Ben Taieb, and Y.-A. L. Borgne. "Machine learning strategies for time series forecasting". In: *European business intelligence summer school*. Springer. 2012, pp. 62–77.

[187] A. Venkatraman, M. Hebert, and J. A. Bagnell. "Improving multi-step prediction of learned time series models". In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015.

[188] P. J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[189] J. Langford, R. Salakhutdinov, and T. Zhang. "Learning nonlinear dynamic models". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 593–600.

[190] A. Graves. "Generating sequences with recurrent neural networks". In: *arXiv preprint arXiv:1308.0850* (2013).

[191] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[192] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[193] A. Veit, M. J. Wilber, and S. Belongie. "Residual networks behave like ensembles of relatively shallow networks". In: *Advances in neural information processing systems* 29 (2016).

[194] M. Müller. "Dynamic time warping". In: *Information retrieval for music and motion* (2007), pp. 69–84.

[195] T. Eiter and H. Mannila. *Computing discrete Fréchet distance*. Tech. rep. CD-TR 94/64. Technische Universitat Wien, 1994.

[196] S. M. Khansari-Zadeh and A. Billard. "A dynamical system approach to realtime obstacle avoidance". In: *Autonomous Robots* 32.4 (May 2012), pp. 433–454. ISSN: 1573-7527.

[197] M. Saveriano and D. Lee. "Distance based dynamical system modulation for reactive avoidance of moving obstacles". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 5618–5623.

[198] L. Huber, A. Billard, and J.-J. Slotine. "Avoidance of Convex and Concave Obstacles With Convergence Ensured Through Contraction". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1462–1469.

[199] L. Huber, J.-J. Slotine, and A. Billard. "Avoiding Dense and Dynamic Obstacles in Enclosed Spaces: Application to Moving in Crowds". In: *IEEE Transactions on Robotics* (2022).

[200] L. Van der Maaten and G. Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[201] A. Ude, B. Nemec, T. Petrić, and J. Morimoto. “Orientation in Cartesian space dynamic movement primitives”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 2997–3004.

[202] R. Pérez-Dattari, C. Della Santina, and J. Kober. “Deep Metric Imitation Learning for Stable Motion Primitives.” In: *Advanced Intelligent Systems* (2024).

[203] J. Zhang, H. B. Mohammadi, and L. Rozo. “Learning Riemannian Stable Dynamical Systems via Diffeomorphisms”. In: *6th Annual Conference on Robot Learning*. 2022.

[204] J. Urain, D. Tateo, and J. Peters. “Learning stable vector fields on lie groups”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 12569–12576.

[205] M. Saveriano, F. J. Abu-Dakka, and V. Kyrki. “Learning stable robotic skills on Riemannian manifolds”. In: *Robotics and Autonomous Systems* (2023), p. 104510.

[206] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. “Probabilistic movement primitives”. In: *Advances in neural information processing systems*. 2013, pp. 2616–2624.

[207] M. Y. Seker, M. Imre, J. H. Piater, and E. Ugur. “Conditional Neural Movement Primitives.” In: *Robotics: Science and Systems*. Vol. 10. 2019.

[208] G. Li, Z. Jin, M. Volpp, F. Otto, R. Lioutikov, and G. Neumann. “ProDMP: A Unified Perspective on Dynamic and Probabilistic Movement Primitives”. In: *IEEE Robotics and Automation Letters* 8.4 (2023), pp. 2325–2332.

[209] W. Zhi, T. Lai, L. Ott, and F. Ramos. “Diffeomorphic Transforms for Generalised Imitation Learning.” In: *L4DC*. 2022, pp. 508–519.

[210] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal. “Online movement adaptation based on previous sensor experiences”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 365–371.

[211] L. Koutras and Z. Doulgeri. “A correct formulation for the orientation dynamic movement primitives for robot control in the cartesian space”. In: *Conference on robot learning*. PMLR. 2020, pp. 293–302.

[212] M. Lang and S. Hirche. “Computationally efficient rigid-body gaussian process for motion dynamics”. In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1601–1608.

[213] M. Arduengo, A. Colomé, J. Lobo-Prat, L. Sentis, and C. Torras. “Gaussian-process-based robot learning from demonstration”. In: *Journal of Ambient Intelligence and Humanized Computing* (2023), pp. 1–14.

[214] J. Silvério, L. Rozo, S. Calinon, and D. G. Caldwell. “Learning bimanual end-effector poses from demonstrations using task-parameterized dynamical systems”. In: *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2015, pp. 464–470.

[215] I. Havoutis and S. Calinon. “Learning from demonstration for semi-autonomous teleoperation”. In: *Autonomous Robots* 43 (2019), pp. 713–726.

[216] M. J. Zeestraten, I. Havoutis, J. Silvério, S. Calinon, and D. G. Caldwell. “An approach for imitation learning on Riemannian manifolds”. In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1240–1247.

[217] S. Kim, R. Haschke, and H. Ritter. “Gaussian mixture model for 3-dof orientations”. In: *Robotics and Autonomous Systems* 87 (2017), pp. 28–37.

[218] Y. Huang, F. J. Abu-Dakka, J. Silvério, and D. G. Caldwell. “Toward orientation learning and adaptation in cartesian space”. In: *IEEE Transactions on Robotics* 37.1 (2020), pp. 82–98.

[219] F. J. Abu-Dakka, Y. Huang, J. Silvério, and V. Kyrki. “A probabilistic framework for learning geometry-based robot manipulation skills”. In: *Robotics and Autonomous Systems* 141 (2021), p. 103761.

[220] H. C. Ravichandar and A. Dani. “Learning position and orientation dynamics from demonstrations via contraction analysis”. In: *Autonomous Robots* 43.4 (2019), pp. 897–912.

[221] H. K. Khalil. *Nonlinear systems; 3rd ed.* Prentice Hall, 2002.

[222] C. M. Kellett. “A compendium of comparison function results”. In: *Mathematics of Control, Signals, and Systems* 26 (2014), pp. 339–374.

[223] T. Needham. *Visual Differential Geometry and Forms: A Mathematical Drama in Five Acts*. Princeton University Press, 2021.

[224] A. Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O’Reilly Media, Inc.", 2022.

[225] E. Deza, M. M. Deza, M. M. Deza, and E. Deza. *Encyclopedia of distances*. Springer, 2009.

[226] S. Sommer, T. Fletcher, and X. Pennec. “Introduction to differential and Riemannian geometry”. In: *Riemannian Geometric Statistics in Medical Image Analysis*. Elsevier, 2020, pp. 3–37.

[227] P. Corke and J. Haviland. “Not your grandmother’s toolbox–the Robotics Toolbox reinvented for Python”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 11357–11363.

[228] K. Van Wyk, M. Xie, A. Li, M. A. Rana, B. Babich, B. Peele, Q. Wan, I. Akinola, B. Sundaralingam, D. Fox, *et al.* “Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior”. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 3202–3209.

[229] C. Cuan, A. Okamura, and M. Khansari. “Leveraging Haptic Feedback to Improve Data Quality and Quantity for Deep Imitation Learning Models”. In: *arXiv preprint arXiv:2211.03020* (2022).

[230] P. Becker, O. Arenz, and G. Neumann. “Expected Information Maximization: Using the I-Projection for Mixture Density Estimation”. In: *International Conference on Learning Representations*. 2020.

[231] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. “Implicit behavioral cloning”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 158–168.

[232] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2023.

[233] M. Lutter and J. Peters. “Combining physics and deep learning to learn continuous-time dynamics models”. In: *The International Journal of Robotics Research* 42.3 (2023), pp. 83–107.

[234] V. Nair and G. E. Hinton. “Rectified linear units improve restricted Boltzmann machines”. In: *Icml*. 2010.

[235] D. Hendrycks and K. Gimpel. “Gaussian error linear units (GELUs)”. In: *arXiv preprint arXiv:1606.08415* (2016).

[236] J. L. Ba, J. R. Kiros, and G. E. Hinton. “Layer normalization”. In: *arXiv preprint arXiv:1607.06450* (2016).

[237] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. “Algorithms for hyper-parameter optimization”. In: *Advances in neural information processing systems* 24 (2011).

[238] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. “Optuna: A next-generation hyperparameter optimization framework”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[239] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. “Robotics: Modelling, Planning and Control”. In: Springer, 2009. Chap. Force Control, pp. 363–405.

[240] S. Kawamura, F. Miyazaki, and S. Arimoto. “Is a local linear PD feedback control law effective for trajectory tracking of robot motion?” In: *Proceedings. 1988 IEEE International Conference on Robotics and Automation*. IEEE. 1988, pp. 1335–1340.

[241] C. Della Santina, C. Duriez, and D. Rus. “Model based control of soft robots: A survey of the state of the art and open challenges”. In: *arXiv preprint arXiv:2110.01358* (2021).

[242] P. Beeson and B. Ames. “TRAC-IK: An open-source library for improved solving of generic inverse kinematics”. In: *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2015, pp. 928–935.

[243] P. Corke and J. Haviland. “Not your grandmother’s toolbox–the Robotics Toolbox reinvented for Python”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 11357–11363.

[244] I. B. Love. “Levenberg-Marquardt Algorithm in Robotic Controls”. PhD thesis. Ph. D. dissertation, University of Washington, 2020.

[245] P. Dierckx. *Curve and surface fitting with splines*. Oxford University Press, 1995.

[246] M. Saveriano and D. Lee. “Point cloud based dynamical system modulation for reactive avoidance of convex and concave obstacles”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 5380–5387.

[247] V. I. Arnold. “Ordinary differential equations”. In: 3rd ed. Springer Science & Business Media, 1992, pp. 97–98.

[248] J. R. Munkres. “Topology”. In: 2nd ed. Pearson Education, 2000, pp. 119–126.

[249] C. C. Pugh. “Real mathematical analysis”. In: 2nd ed. Undergraduate Texts in Mathematics. Springer, 2015, pp. 81, 97.

[250] K. C. Border. “Fixed point theorems with applications to economics and game theory”. In: Cambridge university press, 1985, pp. 53–66.

[251] R. K. Nagle, E. B. Saff, and A. D. Snider. “Fundamentals of differential equations”. In: 9th ed. Pearson, 2018, p. 53.

[252] J. M. Lee. *Riemannian manifolds: an introduction to curvature*. Vol. 176. Springer Science & Business Media, 2006.

[253] C. Berg. *Complex Analysis*. Matematisk Afdeling, Københavns Universitet, 2008.

[254] J. Jeong, M. Jun, and M. G. Genton. “Spherical process models for global spatial statistics”. In: *Statistical science: a review journal of the Institute of Mathematical Statistics* 32.4 (2017), p. 501.

[255] V. Guillemin and A. Pollack. *Differential topology*. Vol. 370. American Mathematical Soc., 2010.

[256] J. Oprea. *Differential geometry and its applications*. MAA, 2007.

Acknowledgements

Pursuing a Ph.D. has been a transformative experience, both academically and personally. It was my first time living on my own and far from home. The challenges included not only conducting high-level research but also learning to navigate a country I knew little about, building a new social network, adapting to working in a second language, and more. The global pandemic, which lasted for years, made this journey even more challenging. Looking back, I will not claim that my Ph.D. process was easy, because it was not. Nevertheless, I am happy to say that I managed to conduct research I am proud of, which I hope will further advance our knowledge in robotics. It was a beautiful experience that significantly helped me grow both as a person and a researcher. This would not have been possible without the support of my family, friends, and supervisors, for which I am forever grateful.

I would like to express my gratitude to all the people who positively influenced this experience, with whom I have created memories I will cherish for the rest of my life.

Foremost, I would like to thank my daily supervisor and co-promotor, Jens, for his invaluable guidance and support throughout my Ph.D. journey. Your insights and expertise have been instrumental in shaping my mindset and approach to research. You provided me with the academic freedom to find my research path, but you were always ready to quickly and lucidly assess the state of my work, efficiently identifying key points to address or improve—a quality I highly admire. I hope to someday reach that level of scientific clarity and maturity.

In addition, I would like to thank my co-promotor, Robert, for his priceless advice and feedback, and for sharing his passion for research, always having the computer ready to show some interesting paper.

I would also like to express my gratitude to Georgia, Hamidreza, Gert, Martijn, and Hans for kindly agreeing to be part of my Ph.D. committee, taking the time to review my dissertation and providing detailed feedback.

My heartfelt thanks go to my colleagues at the Department of Cognitive Robotics. I would like to thank Hai, Padmaja, Mert, Osama, Andrés, and Ajith for their kindness and for always being ready to share a laugh. I fondly remember you all from my first years at TU Delft. I also extend my gratitude to Bas, Jelle, Linda, Jihong, Ravi, Leandro, Cong, and Zlatan. Our group meetings, discussions, and trips will always hold a special place in my heart. Special thanks to Jelle for helping with the translation of the thesis's summary to Dutch. Additionally, I would like to thank Maxi (Kronmüller), Max (Spahn), Max (Lodel), Corrado, and Oscar for their smiles and for always sparking insightful conversations.

Thanks to Carlos, who has been one of my mentors since the days I took my first steps in robotics, and who, together with the *chili con carne* people, Tomás (Pippia),

Momo, Abhimanyu, Maolong, and Mattia, welcomed me with open arms during my first months in the Netherlands. Thanks also to Bruno for also welcoming me when I arrived in the Netherlands and—later, together with Luisa—always making sure to build a beautiful group of friends, frequently organizing great dinners at his place. Finally, I want to thank Giovanni and Álvaro, my Ph.D. brothers, with whom I shared almost the entire Ph.D. together. We grew together through this experience, navigating moments of research uncertainty typical of the early years, enduring the pandemic, and the stress of finishing the Ph.D. dissertation. Although there were times we probably drove each other crazy, the experiences we shared were unique, and I will always look back on them with a smile. Thanks also to Natalia and Bea, who joined us in this journey, sometimes having to tolerate *very interesting* robotics conversations on a Friday evening.

Moreover, thanks to the colleagues who joined along the way: Tasos, Dennis, Elia, Nils, Clarence, Gustavo, Anna, Ashwin, Julian, Andreu, Ebrahim, Yujie, Anton, Diego, Domenico, and Tomáš, for the enjoyable conversations we shared during lunchtime or in the hallways of the faculty. To Zhaoting, Saray, and Maxi (Stolzle), thank you for the fruitful and interesting collaborations we have had. To Mariano, thank you for making the days at the office more enjoyable. To Luzia, for the pleasant moments we have shared since those dinners Bruno organized years ago. And to Lasse, who was a key member of these dinners, thank you for nurturing and sustaining our group's friendships and camaraderie with thoughtful gestures, including organizing farewells and hackathons.

Thanks to Manuel for always being up for sharing a beer, having a barbecue, or engaging in a quick chess match; to Lorenzo, for your kindness, openness, and the delicious pasta dishes you would commonly invite us to share together with a *Primitivo*; to Italo, for hosting us at your place countless times, even though sometimes you had to leave early the next day; and to Tomás (Coleman) for the numerous house invitations too, and for driving me around during our trip to Japan, where we could experience the local *craic* firsthand.

I am also grateful to Luka, Micah, and Nicky for sharing their insights and knowledge on the KUKA robot, which was crucial in building the controllers used in many of my experiments. Additionally, I extend my gratitude to Javier and Cosimo for their openness to insightful conversations and their collaborations on some of the chapters of this dissertation.

Special thanks to the support staff, Hanneke, Karin, Noortje, Ellen, and Loulou, for always quickly assisting us with any problems we encountered and for organizing incredible department events. I also extend my gratitude to André for his invaluable technical support.

Furthermore, I would like to thank the FlexCRAFT team, including Akshay, David, Xin, Rekha, Robbert, Rick, Thomas, Ad, Peter, Jordy, Bastiaan, Boi, Jimmy, Gert, and Eldert, for their collaboration and workshop meetings, which were always motivating and enjoyable. Working with such a dedicated group has been an inspiring experience.

If there is one person who felt this experience almost as deeply as I did, and to whom I will be forever grateful, it is Elena. You were there when I worked late nights,

on weekends, and sometimes even on holidays. You took care of me, ensuring that I ate more than just bread during the most stressful weeks, and looked after me every time I got sick, even taking an express flight from Portugal if necessary. I cannot thank you enough for your love and for how important you have been throughout these years, and it is difficult to imagine achieving the outcome of this dissertation without your support. I also want to thank your family for welcoming me every time we visited your beautiful home in Rome, sharing many pleasant dinners and walks through the city.

Lastly, I would like to thank my family for their constant love and support. Despite the distance, I felt their encouragement and strength every day. In person, you ensured this was the case with warm welcomes whenever I visited home. Whether it was a barbecue being prepared or a nice breakfast after a long journey, you always ensured we had great days to catch up after being apart for so long. To my parents, thank you for always being there and providing me with the tools to follow my passion. Whether it was teaching me discipline or physics many years ago, or letting me take over the study room with papers and the whiteboard in recent years, your support has been invaluable. Thanks also to my sisters, who, despite our busy lives, always made time to share family moments whenever I visited. Those Sunday lunches, with the whole family gathered once more, giving the impression that time stood still and we were all living together again, are moments I cherish every day of my life.

Rodrigo Pérez Dattari
Delft, July 2024

List of Publications

Journal papers

R. Pérez-Dattari, C. Della Santina, and J. Kober. "Deep Metric Imitation Learning for Stable Motion Primitives." In: *Advanced Intelligent Systems* (2024) — **Chapter 6**

R. Pérez-Dattari and J. Kober. "Stable Motion Primitives via Imitation and Contrastive Learning". In: *IEEE Transactions on Robotics* (2023) — **Chapter 5**

C. Celemin*, R. Pérez-Dattari*, E. Chisari*, G. Franzese*, L. de Souza Rosa, R. Prakash, Z. Ajanović, M. Ferraz, A. Valada, and J. Kober. "Interactive Imitation Learning in Robotics: A Survey". In: *Foundations and Trends® in Robotics* 10.1–2 (2022) — **Chapter 2**

R. Pérez-Dattari*, B. Brito*, O. de Groot, J. Kober, and J. Alonso-Mora. "Visually-guided motion planning for autonomous driving from interactive demonstrations". In: *Engineering Applications of Artificial Intelligence* 116 (2022) — **Chapter 4**

R. Pérez-Dattari, C. Celemin, G. Franzese, J. Ruiz-del-Solar, and J. Kober. "Interactive learning of temporal features for control: Shaping policies and state representations from human feedback". In: *IEEE Robotics & Automation Magazine* 27.2 (2020), pp. 46–54 — **Chapter 3**

Conference papers

E. Drijver, R. Pérez-Dattari, J. Kober, C. Della Santina, and Z. Ajanović. "Robotic Packaging Optimization with Reinforcement Learning". In: *IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2023

P. Valletta, R. Pérez-Dattari, and J. Kober. "Imitation Learning with Inconsistent Demonstrations through Uncertainty-based Data Manipulation". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021

* Authors contributed equally.