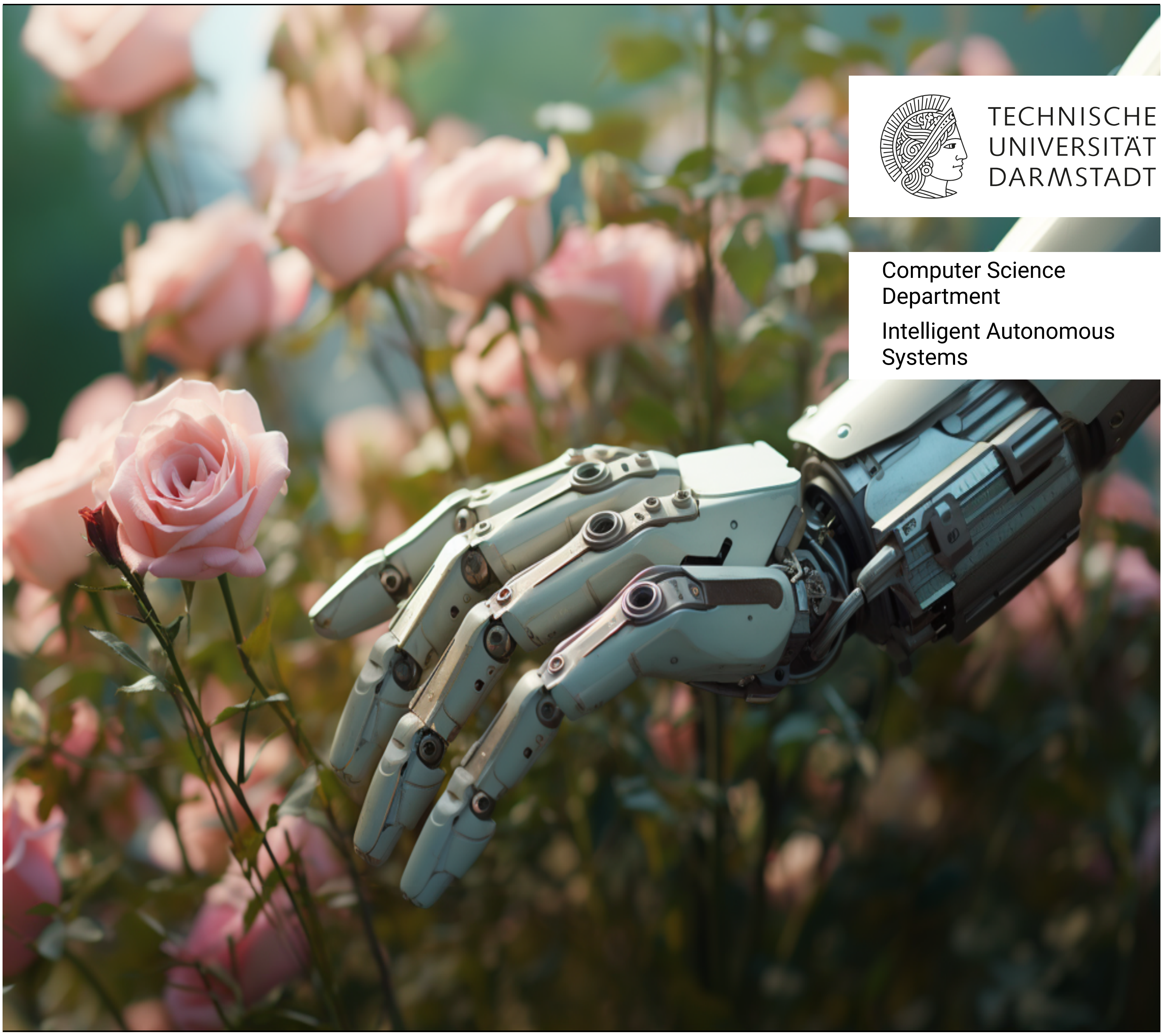
Deep Generative Models for Motion Planning and Control

Normalizing Flows, Energy-Based Models & Diffusion Models in Robotics
Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Julen Urain aus Deba, Spain
Tag der Einreichung: 31.10.2023, Tag der Prüfung: 18.12.2023

1. Gutachten: Prof. Jan Peters, Ph.D.
2. Gutachten: Prof. Katerina Fragkiadaki, Ph.D.
Darmstadt, Technische Universität Darmstadt



TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department

Intelligent Autonomous Systems

Deep Generative Models for Motion Planning and Control
Normalizing Flows, Energy-Based Models & Diffusion Models in Robotics

Accepted doctoral thesis by Julen Urain

Date of submission: 31.10.2023
Date of thesis defense: 18.12.2023

Darmstadt, Technische Universität Darmstadt

Nire aitaren oroimenean

Erklärungen laut Promotionsordnung

§ 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 31.10.2023

Julen Urain

Abstract

This thesis investigates the problem of robot motion generation, focusing primarily on **data-driven motion generation**. Traditionally, robot motion has been dictated by manually designed models. While effective for structured industrial environments where variability is minimal (e.g. palletizing robots handling similarly shaped boxes), these models fall short in more complex environments such as domestic spaces filled with diverse objects such as cutlery, bottles, and mugs.

Data-driven motion generation, often referred to as Learning from Demonstrations or Imitation Learning, is emerging as a promising solution for these complex environments. Using human expert demonstrations, the goal is to teach robots desired behaviors through the demonstrations. The power of this approach is evident in its transformative impact on fields such as Computer Vision and Natural Language Processing, where deep generative models have successfully produced images and text. In Robotics, however, current data-driven models still struggle to achieve the same breadth of generalization and robustness.

In this context, this thesis is inspired by the successful cases of both image and text generation. A critical insight underpins our investigation: An important factor of generalization in both image and text generation lies in the architectures chosen. While image generative models use architectures such as Convolutional Neural Networks to capture local geometric features in images, text generative models rely on structures such as Transformers to infer temporal features. With this in mind, the work presented in this thesis explores the following question: **What are the architectural elements that we should integrate into our generative models in order to correctly generate robot movements?**

In this direction, in this thesis, we propose three different works that explore the integration of different robotics-relevant properties (stability, geometry, and composability) in deep generative models.
(1) With ImitationFlows, we study the problem of integrating global stability into motion policies. We propose a novel architecture that exploits the expressiveness of Normalizing Flows with the guarantee of learning globally stable behaviors. We show that these models

can be used to represent stable motion behaviors in the robot's end-effector space (6D position and orientation); a useful space for many robotic tasks.
(2) With Composable Energy Policies, we study the problem of combining multiple motion policies to solve multi-objective problems. We explore the connections between multi-objective motion generation and Energy-Based Models and propose a novel model for combining energy-based policies represented in arbitrary spaces.
(3) With SE(3)-DiffusionFields, we explore the problem of learning useful cost functions for Motion Planning. We propose to adapt Diffusion Models to the Lie group SE(3), which allows us to design Diffusion Models in the robot's end-effector space. We show that we can use these models to represent grasp pose distributions and use them as cost functions in Motion Planning problems.
Each of the proposed methods has been evaluated in both simulated and real-world experiments to show their performance on real-world robotics problems, and we have open-sourced the codebase of the methods to encourage the community to build on our proposed solutions.

Overall, this thesis explores novel ways to apply deep generative models to robotics problems. We show the benefit of integrating robotics-relevant features such as geometry and composability with deep generative models, thereby benefiting from the expressiveness of deep generative models while improving generalization thanks to properly chosen inductive biases.

Zusammenfassung

In dieser Arbeit wird das Problem der Roboterbewegungserzeugung untersucht, wobei der Schwerpunkt auf der **datengesteuerten Bewegungserzeugung** liegt. Traditionell wurde die Roboterbewegung durch von Hand entworfene Modelle bestimmt. Während diese Modelle für strukturierte industrielle Umgebungen, in denen die Variabilität minimal ist (z. B. Palettierroboter, die ähnlich geformte Kartons handhaben), effektiv sind, greifen sie in komplexeren Umgebungen, wie z. B. in häuslichen Räumen, die mit verschiedenen Objekten wie Besteck, Flaschen und Tassen gefüllt sind, zu kurz.

Die datengesteuerte Bewegungserzeugung, die oft als Lernen aus Demonstrationen oder Nachahmungslernen bezeichnet wird, stellt eine vielversprechende Lösung für diese komplexen Umgebungen dar. Ziel ist es, Robotern durch Demonstrationen von menschlichen Experten das gewünschte Verhalten beizubringen. Die Stärke dieses Ansatzes zeigt sich in seinen transformativen Auswirkungen auf Bereiche wie Computer Vision und natürliche Sprachverarbeitung, wo tiefe generative Modelle erfolgreich Bilder und Texte erzeugt haben. In der Robotik haben die derzeitigen datengesteuerten Modelle jedoch immer noch Probleme, die gleiche Bandbreite an Generalisierung und Robustheit zu erreichen.

In diesem Zusammenhang ist diese Arbeit von den erfolgreichen Fällen der Bild- und Texterzeugung inspiriert. Eine kritische Einsicht untermauert unsere Untersuchung: Ein wichtiger Faktor für die Generalisierung sowohl bei der Bild- als auch bei der Texterzeugung sind die gewählten Architekturen. Während bildgenerative Modelle Architekturen wie Convolutional Neural Networks verwenden, um lokale geometrische Merkmale in Bildern zu erfassen, verlassen sich textgenerative Modelle auf Strukturen wie Transformers, um zeitliche Merkmale abzuleiten. Vor diesem Hintergrund geht die Arbeit in dieser Dissertation der folgenden Frage nach: **Welches sind die architektonischen Elemente, die wir in unsere Modelle integrieren sollten, um Roboterbewegungen korrekt zu generieren?**

In dieser Richtung, in dieser Arbeit schlagen wir drei verschiedene Arbeiten, die die Integration von verschiedenen Robotik-relevanten Eigenschaften in der generativen Modelle

wie Stabilität, Geometrie und Composability zu erkunden.
(1) Mit ImitationFlows untersuchen wir das Problem der Integration von globaler Stabilität in Bewegungspolitiken. Wir schlagen eine neuartige Architektur vor, die die Expressivität von Normalizing Flows mit der Garantie des Lernens von global stabilem Verhalten ausnutzt. Wir zeigen, dass diese Modelle verwendet werden können, um stabiles Bewegungsverhalten im Endeffektor-Raum des Roboters darzustellen (6D-Position und -Orientierung); ein nützlicher Raum für mehrere Roboteraufgaben.
(2) Mit Composable Energy Policies untersuchen wir das Problem der Kombination mehrerer Bewegungspolitiken zur Lösung multikriterieller Probleme. Wir erforschen die Zusammenhänge zwischen multikriterieller Bewegungsgenerierung und energiebasierten Modellen und schlagen ein neuartiges Modell zur Kombination von energiebasierten Strategien in beliebigen Räumen vor.
(3) Mit SE(3)-DiffusionFields untersuchen wir das Problem des Lernens nützlicher Kostenfunktionen für die Bewegungsplanung. Wir schlagen vor, Diffusionsmodelle an die Lie-Gruppe SE(3) anzupassen, was uns erlaubt, Diffusionsmodelle im Endeffektorraum des Roboters zu entwerfen. Wir zeigen, dass wir diese Diffusionsmodelle verwenden können, um Greifposenverteilungen darzustellen und sie in Bewegungsplanungsproblemen als Kostenfunktion zu nutzen.
Jede der vorgeschlagenen Methoden wurde sowohl in simulierten als auch in realen Experimenten evaluiert, um ihre Leistung für reale Roboterprobleme zu demonstrieren, und wir haben die Codebasis der Methoden als Open Source zur Verfügung gestellt, um die Community zu ermutigen, auf unseren vorgeschlagenen Lösungen aufzubauen.

Insgesamt erforscht diese Arbeit neue Wege, um tiefe generative Modelle für Robotikprobleme zu nutzen. Wir zeigen den Nutzen der Integration von Robotik-relevanten Merkmalen wie Geometrie und Kompositionsfähigkeit mit tiefen generativen Modellen. Dabei profitieren wir von der Ausdruckskraft tiefer generativer Modelle und verbessern gleichzeitig die Generalisierung dank richtig gewählter induktiver Verzerrungen.

Acknowledgment

The past five years have been an incredible journey. The time spent pursuing my Ph.D. has helped me develop into a better researcher and person. The research presented in this dissertation is a testament to the invaluable contributions and unwavering support of countless individuals.

At this point, I would like to express my heartfelt gratitude to each and every one of them:

To my mentor; *Jan Peters*, for inspiring me to become a great roboticist, for advising me on how to do good research, for giving me the freedom to follow my own research ideas and for providing the perfect environment to develop them, and for providing valuable feedback and guidance for my future career.

To the three wise magi; *Davide*, *Carlo*, and *Georgia* who helped me shape my ideas into tangible works, taught me how to think through the problems, and provided me with invaluable feedback to improve the way I research and write.

To the knights of IAS, *Puze*, *Joe*, *Niklas*, *Joao*, *Pascal*, *Junning*, *Tuan*, *An*, *Kay*, *Samuele* and many other members of IAS who inspired me with their bright minds and hard work and made the whole journey fun.

To my external collaborators, *Anqi*, *Bala*, *Karl*, *Sasha*, and *Michelle*, who taught me a lot, showed me new ways of thinking, and guided me through my internship and collaborations.

To my friends *Rafa*, *Laura*, *Juancho*, *Dani*, *Manu* and *Barbara*, who made the German winters warmer and the summers brighter.

To my wife *Margot*, who has been the biggest emotional support on this journey. For understanding me, for waiting for me, for supporting me, and for every single moment we shared.

To my family and all my friends in the Basque Country, who have reminded me that there is a warm place where I can always return to recharge my batteries.

Contents

1. Introduction

”In the robot’s maze, where paths were once concealed,
With data’s dance, the true trajectory’s revealed.”
GPT-4

A perennial challenge in Robotics is: *How do we program the precise motion a robot must follow to accomplish a specific task?* This pivotal question gives birth to the concept of **Robot Motion Generation**[1]. At its essence, Robot Motion Generation endeavors to generate a feasible sequence of actions that enables a robot to navigate the complexities of its environment and execute tasks with precision and efficiency. Traditional approaches [105, 182, 117, 119, 188] often involve a modular engineering methodology, with each module designed to address a particular facet of the task. Consider the illustrative example on Figure 1.1 of an industrial robot, tasked with transferring boxes from one pallet to another. This seemingly straightforward task unfolds into a myriad of considerations that the engineer must take into consideration: adherence to joint limits, collision avoidance, ensuring secure grasping of boxes, and the generation of fluid, uninterrupted trajectories, to name a few.

Despite the success of automation in our industrial warehouses, this success is conditioned on a highly structured and limited environment. The boxes the robot manipulates always have the same shape, the scene remains fixed and the task the robot should solve is always the same. Now, let us consider the task of tidying up the kitchen on Figure 1.1. The scene presents a highly unstructured situation in which the objects to interact are highly diverse; from mugs and bottles to dishes. Additionally, the robot should be able to adapt to possible changes in the scene due to a person entering the kitchen and we might require different motion behaviors to interact with different objects. Despite both tasks can be represented

[1] We use the term Robot Motion Generation to encapsulate all types of algorithms and models that generate motion for robots, from myopic Motion Control models [105, 182] to Motion Planning algorithms [117, 188].

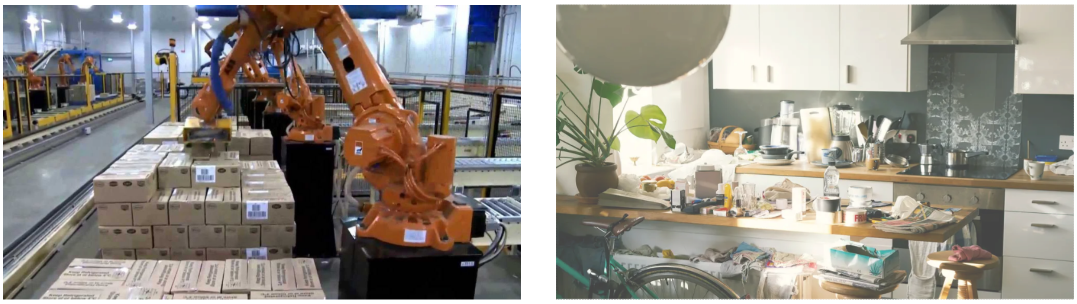


Figure 1.1.: (Left) A robotics box palletizing system. (Right) A cluttered kitchen to tidy up. Despite both tasks can be represented as a sequentiation of pick-and-place operations, the second remains unsolved.[2]

as a series of consecutive pick-and-place operations, while the first is solved daily, the second remains unsolved. We consider this is due to two major bottlenecks in traditional approaches:
(I) The high variability in the objects to interact would require the engineer to model a desirable behavior for each object and if the number of objects is too high, this is an intractable approach.
(II) Despite different behaviors could be modeled for different objects, some desirable behaviors might be hard to formalize in code while providing a human demonstration of the desired behavior is easier and more intuitive.

A promising research direction to tackle this problem is to use data-driven models rather than engineered models to program robot behaviors. **Learning from demonstrations** [208, 14, 6], the paradigm of learning robot behaviors by imitating expert demonstrations has started to show some impressive results in both locomotion [147] and autonomous driving [264] moving the robots out of the research labs. Nevertheless, learning-based methods for robotics manipulation remain confined to the research labs. Due to the high variability and dimensionality of robotics manipulation (See Figure 1.1), current approaches have not been able to showcase robust generalization. This state of affairs might be directly connected to two possible reasons:
(I) We are training our models on too small datasets, unable to capture the whole complexity of the task to be solved.
(II) We are using models that are not properly tailored for robotics data, leading to poor

[2] A robotics box palletizing system, image from ABB Robotics - Palletizing Cartons. A cluttered kitchen, image from Justin Lambert/Getty Images.

generalization beyond the data regime.

The importance of properly tackling these two problems has been recently shown essential in both Natural Language Processing (NLP) and Computer Vision (CV). Recent advancements in **Deep Generative Models**, such as image generation [185, 204, 203, 181] or text generation [101, 20, 267] were achieved by developing models that can learn the underlying distribution of vast amounts of data and properly generalize beyond the demonstrations. The success of deep generative models in image and text generation relies on two critical components. On the one side, highly expressive generative models (Energy Based Models (EBM) [42], Normalizing Flows (NFlow) [198, 27], Diffusion Models [227, 80]) leverage the information on massive amounts of data. Equally important, carefully designed model architectures (U-Net [200], Transformers [248]) encode the unique geometric or temporal features of the data and improve the generalization of the generative models beyond the dataset. Convolutional Neural Networks extract local visual features in images while Transformers find temporal correlations in text sentences or videos.

This thesis is built inspired by these success cases. What are the lessons that we, as roboticists, could learn from the success in other areas of AI? How should we build our Learning from Demonstration models for robot motion generation? As previously stated, an integral part of proper generalization on deep generative models is the choice of the model. Thus, our research is focused on searching those models and algorithmic principles that allow a proper deployment of deep generative models for robotics manipulation. Which are those inherent computational and mathematical principles that roboticists have exploited in Classical Robotics? How can we distill these principles in data-driven deep generative models to enhance generalization? In essence, the research question that drives this thesis is:

How can one integrate existing knowledge and data-driven generative modeling methods to learn motion generation models, benefiting from the goods of each field?

In this thesis, we have identified *two essential principles* for robot motion generation; **geometry of robot skills**, and **skill composability**. The importance of geometry in modeling robot skills cannot be understated. For example, when modeling a grasping policy, the task is defined in the robot's end-effector space, necessitating its modeling on the Lie group SE(3). If we learn this policy directly in the robot's configuration space, it might lead to bad generalization under novel contexts. On the other hand, complex robot tasks are compositional in nature. A simple task, such as pouring water on some flowers, requires solving multiple tasks concurrently: avoiding collisions with the environment, considering the robot's joint limits, and reaching to the flowers to water them. Thus, we should take it into consideration, instead of learning monolithic policies.

1.1. Contributions

This thesis examines one of the most straightforward strategies for data-driven motion generation. The abstract procedure consists of two interleaved steps:

1. Given a set of data demonstrations of a desirable behavior, fit a density model/generative model that covers the underlying distribution of the data.
2. Exploit the learned model to generate robot motion rather as a control policy or integrated into a motion planning problem.

This high-level strategy has been widely explored in the robot learning literature and it is popularly known as Learning from Demonstration [209, 193] or Imitation Learning [208, 163]. Depending on the type of algorithm and model that is learned, Learning from Demonstration problems are classified into different fields. Behavioral Cloning [180, 52] or Motion Primitives [207, 166] propose learning control policies. These policies are deployed as high-frequency closed-loop controllers and are myopic with respect to future events. Inverse Optimal Control [96, 50] and Inverse Reinforcement Learning [273, 56] fields instead propose learning cost functions for motion planning. Motion planning methods are computationally more expensive than the policies leading to slower control frequencies. In favor, these methods look ahead to the future, finding the motion that best maximizes future outcomes. Despite being different approaches, all of them can be described as a special case of the above high-level strategy.

The purpose of this thesis is to explore the state of the art in these fields, bridging explicit connections with the field of deep generative models and suggesting methods to move the field forward:

- In Chapter 2, the foundations of data-driven robot motion generation are presented. We explore the different deep generative models that have been used in fields such as Behavioral Cloning, Motion Primitives, or Inverse Optimal Control and present them in a common framework that allows us to understand their relation to deep generative models. This perspective allows us to find out how novel approaches in deep generative models such as Diffusion Models could be applied to Robot Motion Generation problems.

Then, in three separate chapters, we explore three possible directions to integrate robotics-relevant features (Geometry, Composability, and Stability) with deep generative models:

- In Chapter 3, **we explore the problem of integrating Stability guarantees into our policies.** Despite some previous works [103, 102, 158] in the field of Motion Primitives have explored the problem of designing motion generation models that have stability guarantees; most of these models are limited in expressivity. With ImitationFlows (iFlows) [242, 239], we introduce a model to represent globally stable policies with NFlow. This combination allows exploiting the expressivity of a deep generative model such as NFlow with a robotics-related desirable property, Stability. Given multiple robot tasks are usually defined in the end-effector space, we also adapt this policy to the Lie Group SE(3), to represent globally stable policies in the end-effector space.

- A limitation of policies is that usually are learned to satisfy a single objective. Nevertheless, in robotic tasks, it is common to require motion behaviors that satisfy multiple objectives jointly (avoid collisions, take in mind joint limits and velocity limits, reach a target ...). In Chapter 4, **we study the problem of composing multiple policies to solve multi-objective robot tasks.** The problem of multi-objective motion generation has been explored with methods that range from motion control approaches (Riemannian Motion Policies (RMP) [191, 127]) to motion planning ones (Model Predictive Control (MPC) [255, 13, 82]). With Composable Energy Policies (CEP) [240, 241], we draw connections between EBM [78, 121] and multi-objective motion generation methods. EBM allow the motion generation in a modular approach, with each EBM representing the desired behavior for each objective. We explore the benefits and limitations of composing multiple EBM to generate robot motion and relate it with RMP and MPC methods.

- Despite learning EBM is the most common approach to learning cost functions in multi-objective motion optimization problems, they have several problems. The learned cost functions are usually non-smooth leading to problems when running iterative optimization algorithms. In Chapter 5, **we dive into the problem of learning useful cost functions for motion planning.** With SE(3)-DiffusionFields (SE(3)-DiF) [244], we explore the application of Diffusion Models in motion planning problems. In the context of Inverse Optimal Control (IOC) and Inverse Reinforcement Learning (IRL), EBM are usually learned to represent cost functions. In this chapter, we present the limitations of learning EBM and show the benefits of Diffusion Models as cost functions. To make the Diffusion Models valid to represent cost functions in the end-effector space (to learn for example grasp pose costs), we adapt the training

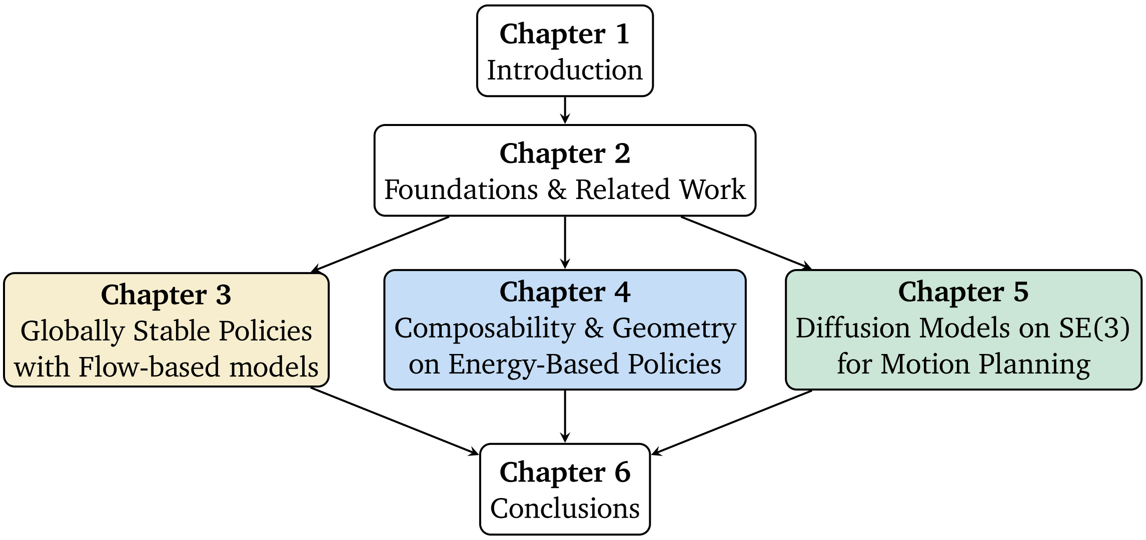


Figure 1.2.: Structure of the thesis. Chapters 3 to 5 are self-contained and introduce a novel method to exploit deep generative models for Motion Control and Motion Planning. Chapter 3: ImitationFlows, Chapter 4: Composable Energy Policies and Chapter 5: SE(3)-DiffusionFields .

and sampling algorithms of the Diffusion Models to the Lie Group SE(3) and exploit them in pick-and-place tasks.

1.2. Thesis Outline

This thesis is structured into six separate chapters (See Figure 1.2). We introduce the foundations and related work in Chapter 2. Then in Chapters 3-5, the main methods are introduced. The concluding Chapter 6 summarizes the thesis and states the conclusions.

Chapter 1 introduces the topic of this thesis and provides a motivation. Then, highlights the contributions of this work.

Chapter 2 sets the fundamentals on both motion generation and deep generative models and reviews the current state of the art in deep generative models for motion control and planning.

Chapter 3 proposes a model to represent globally stable policies using the expressivity of Normalizing Flows. In the first part, the approach to represent globally stable vector fields with Normalizing Flows is presented. In the second part, the method is extended to represent globally stable policies on non-Euclidean manifolds, such as Lie groups.

Chapter 4 builds a connection between EBM and multi-objective motion generation methods such as RMP [191] and MPC [13]. It is shown that both methods can be framed as the solution of a composition of multiple EBM from a probabilistic perspective. From this, we can integrate learned EBM with defined cost functions in a natural way, benefiting from both worlds.

Chapter 5 proposes using Diffusion Models in motion planning problems. The work explores the difference between Diffusion Models and Energy-Based Models for planning and showcases the benefits of a pick-and-place planning problem. Additionally, a novel deep generative model is introduced to generate SE(3) grasp distributions.

Chapter 6 summarizes the thesis and presents the conclusions. Then, open challenges in learning deep generative models for motion generation are presented and propose possible future directions.

2. Foundations and Related Work

This chapter is structured into two main sections. The first section offers a comprehensive overview of the various approaches applied to robot motion generation, framing these methods from a probabilistic perspective. This framing facilitates a natural connection with deep generative models. In the second section, we delve into an array of deep generative models, detailing their diverse training methodologies and providing an overview of their applications in the field of robotics.

2.1. Foundations on Motion Generation

Given the state of the robot $\boldsymbol{s} \in \mathbb{R}^n$ of dimension n, a motion generator can be described as a system or algorithm that produces a sequence of desired states over time $\boldsymbol{\tau} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \dots, \boldsymbol{s}_T) \in \mathbb{R}^{T \times n}$ where T denotes the temporal horizon. The sequence of states $\boldsymbol{\tau}$ is commonly referred as **trajectory**. For the trajectory to be considered physically feasible, there must exist a sequence of actions $(\boldsymbol{a}_1, \boldsymbol{a}_2, \dots, \boldsymbol{a}_{T-1})$ that enables the robot to follow this path. The generated trajectory is then utilized to define the desired motion from an initial state $\boldsymbol{s}_0$ to a desired state $\boldsymbol{s}_T$, in accordance with the motion behavior represented by the trajectory. In robot motion generation, the state $\boldsymbol{s}$ might represent positions $\boldsymbol{x}$, velocities $\dot{\boldsymbol{x}}$ or/and accelerations $\ddot{\boldsymbol{x}}$ and it can relate to robot limbs, joints, end-effectors, or the whole robot in the case of mobile robots.

The challenge of motion generation lies in the design of trajectory generation models $\boldsymbol{\tau} \sim \rho(\boldsymbol{\tau})$ that induce the desirable behavior to accomplish a given task. Due to the high dimensionality and typically large temporal horizon T of the trajectories, these trajectory generators are often designed in a structured manner. Instead of creating models that directly output the entire trajectory $\boldsymbol{\tau} \in \mathbb{R}^{T \times n}$, it is more common to design lower-dimensional models. These may include policies $\pi(\boldsymbol{a}|\boldsymbol{s})$ that produce an action given the current state, or cost functions $c(\cdot) : \mathbb{R}^n \to \mathbb{R}$ that assess the value of a particular

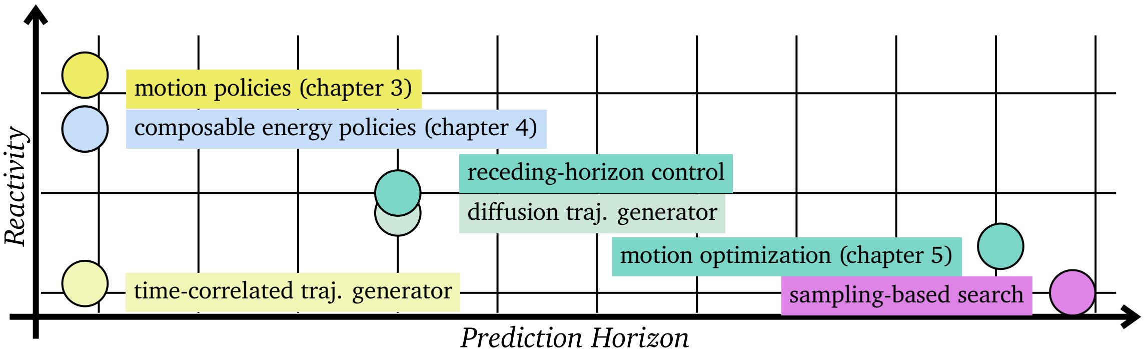


Figure 2.1.: A Comparative Landscape of Motion Generation Algorithms in the Reactivity-Prediction Horizon Plane

state at a given time. However, there are notable exceptions, such as time-correlated normal distributions [166, 128] and diffusion-based models [90, 25, 29], which explicitly model the entire trajectory distribution.

In the Robotics literature, there have been multiple approaches to represent robot motion generators that range from motion control methods [105, 191, 103] to motion planning algorithms [119, 188]. In Figure 2.1, we organize different motion generators along two axes: **Prediction Horizon** and **Reactivity**.

Reactivity refers to the capability of a motion generator to adapt to environmental changes. Motion policies, such as [105, 191] have relatively low computational requirements. This enables the generation of fast, closed-loop control actions, allowing for the prompt alteration of the motion to accommodate changes, such as the sudden appearance of obstacles. In contrast, sampling-based motion planners, such as Rapidly-exploring Random Trees (RRT) [119], generate motion offline due to their computational demands. As a result, they execute motions in an open-loop control mode, lacking the ability to react to changes in the scene.

Prediction Horizon refers to the capability of the motion generator to look ahead. Myopic motion generators [191, 241] create the desired robot movements without predictive look-ahead, making decisions based solely on the current state. In contrast, planning algorithms such as Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [188] or Stochastic Trajectory Optimization for Motion Planning (STOMP) [97], choose the motion based on the future implications of the choice. These algorithms generate trajectories by considering the task's fulfillment at each time step, thereby incorporating foresight into the motion generation process.

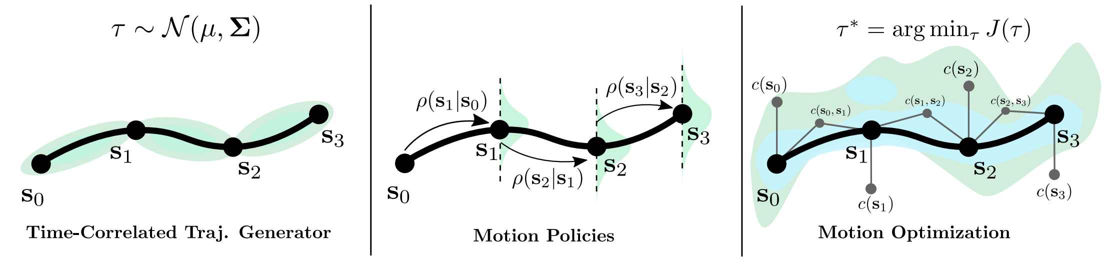

$\tau \sim \mathcal{N}(\mu, \mathbf{\Sigma})$
$\mathbf{s}_0$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
Time-Correlated Traj. Generator
$\rho(\mathbf{s}_1|\mathbf{s}_0)$
$\rho(\mathbf{s}_3|\mathbf{s}_2)$
$\rho(\mathbf{s}_2|\mathbf{s}_1)$
$\mathbf{s}_0$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
Motion Policies
$\tau^* = \arg\min_\tau J(\tau)$
$c(\mathbf{s}_0)$
$c(\mathbf{s}_2)$
$c(\mathbf{s}_0, \mathbf{s}_1)$
$c(\mathbf{s}_1, \mathbf{s}_2)$
$c(\mathbf{s}_2, \mathbf{s}_3)$
$\mathbf{s}_0$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$c(\mathbf{s}_1)$
$c(\mathbf{s}_3)$
Motion Optimization

Figure 2.2.: An illustration of the generation procedure for three different motion generation models. Time-Correlated Trajectory Generators sample the whole trajectory from a Gaussian distribution, Motion Policies generate trajectories autoregressively sampling the next state given the current state, and Motion Optimization algorithms generate the trajectory by solving an optimization problem over the whole trajectory.

In the following, we present an overview of a set of robot motion generators: Time-Correlated Trajectory Generators (Section 2.1.1), Motion Policies (Section 2.1.2), and Motion Optimization methods (Section 2.1.3). We illustrate the different generation processes for each motion generator in Figure 2.2.

2.1.1. Time-Correlated Trajectory Generators

Time-correlated trajectory generation models [97, 166, 154] are one of the simplest types of motion generators. These models propose representing the motion generator as a time-correlated Gaussian distribution

$$\rho(\boldsymbol{\tau}) = \mathcal{N}(\boldsymbol{\tau}|\boldsymbol{\mu}_{\boldsymbol{\tau}}, \boldsymbol{\Sigma}_{\boldsymbol{\tau}}), \tag{2.1}$$

where $\boldsymbol{\mu}_{\boldsymbol{\tau}} = [\boldsymbol{\mu}_{\boldsymbol{s}_1}, \boldsymbol{\mu}_{\boldsymbol{s}_2}, \dots, \boldsymbol{\mu}_{\boldsymbol{s}_T}]$ denotes the mean state for each time-step and $\boldsymbol{\Sigma}_{\boldsymbol{\tau}}$ denotes the covariance matrix. This covariance matrix is constructed to establish correlations between states that are temporally proximate. Owing to the time-correlated covariance, the generated trajectories exhibit smoothness, a critical attribute for ensuring physical plausibility.

Different works differ in the way the time-correlated covariance matrix is computed. In STOMP [97], the covariance is derived from a dynamic system. Let $\boldsymbol{A}$ be a finite difference matrix that relates the position $\boldsymbol{\tau}$, and accelerations $\ddot{\boldsymbol{\tau}}$, $\ddot{\boldsymbol{\tau}} = \boldsymbol{A}\boldsymbol{\tau}$,

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

Then, the covariance matrix is computed as $\boldsymbol{\Sigma}_{\boldsymbol{\tau}}^{-1} = \boldsymbol{A}^{\intercal}\boldsymbol{A}$, that encourages the generated trajectories to have small accelerations. In Probabilistic Movement Primitives (ProMP) [166], the covariance is derived given a set of smooth basis functions

$$\rho(\boldsymbol{\tau}) = \int_{\boldsymbol{\omega}} \mathcal{N}\left(\boldsymbol{\tau}|\Psi^{\intercal}\boldsymbol{\omega}, \boldsymbol{\Sigma}_{\boldsymbol{\tau}}\right) \mathcal{N}\left(\boldsymbol{\omega}|\boldsymbol{\mu}_{\boldsymbol{\omega}}, \boldsymbol{\Sigma}_{\boldsymbol{\omega}}\right) \mathrm{d}\boldsymbol{\omega} = \mathcal{N}(\boldsymbol{\tau}|\Psi^{\intercal}\boldsymbol{\mu}_{\boldsymbol{\omega}}, \Psi^{\intercal}\boldsymbol{\Sigma}_{\boldsymbol{\omega}}\Psi + \boldsymbol{\Sigma}_{\boldsymbol{\tau}}) \tag{2.3}$$

where Ψ denotes the smooth basis function, $\boldsymbol{\omega}$ denotes a set of parameters that are aligned to different temporal instances and $\boldsymbol{\Sigma}_{\boldsymbol{\omega}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{\tau}}$ denote diagonal covariance matrices.

A common application for these types of models is as variational approximation distribution in motion optimization problems [153, 107, 136]. These models approximate complex distributions in optimization problems, facilitating a fast trajectory generation. Another application is in Learning from Demonstration [166, 167, 214]. In [214], time-correlated trajectory generators were further developed to include a conditioning variable $\boldsymbol{c}$, represented as $\rho(\boldsymbol{\tau}|\boldsymbol{c})$. This advancement enables the trajectory generator to adapt to varying environments, enhancing its applicability and versatility.

An interesting extension of these models is by substituting the normal distribution with Diffusion Models. In [90, 29, 25, 197, 84, 71, 257], it was shown that Diffusion Models are expressive enough to learn multi-modal trajectory distributions. Despite this expressiveness, a notable drawback is the sampling process. It involves an iterative method, reminiscent of motion optimization algorithms, which unfortunately results in increased sampling times.

2.1.2. Motion Policies

Motion policies [104, 191, 52, 242] propose modeling the robot behavior with a state-conditioned policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ where $\boldsymbol{a}$ denotes the action applied at state $\boldsymbol{s}$. Given known dynamics $\boldsymbol{s}_{t+1} = \boldsymbol{f}(\boldsymbol{s}_t, \boldsymbol{a}_t)$, we can formulate autonomous transition dynamics $\rho_\pi(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t)$. Utilizing these dynamics, trajectories can be generated using an autoregressive model

$$\rho(\boldsymbol{\tau}) = \rho(\boldsymbol{s}_1) \prod_{t=2}^{T} \rho_\pi(\boldsymbol{s}_t|\boldsymbol{s}_{t-1}), \tag{2.4}$$

where $\rho(\boldsymbol{s}_1)$ denotes the initial state distribution. The process of trajectory generation commences with sampling an initial state $\boldsymbol{s}_1 \sim \rho(\boldsymbol{s}_1)$. Subsequently, the trajectory evolves iteratively by sampling from the autonomous transition dynamics $\boldsymbol{s}_{t+1} \sim \rho_\pi(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t)$ at each step in an autoregressive way.

Motion policies are particularly advantageous for generating reactive motion. Their efficiency in sample generation, coupled with their conditioning on the current state of the robot, enables swift and responsive adaptation to abrupt alterations in both the robot's state and the surrounding environment.

The literature is vast in motion policies. Most of the motion policies in the literature are modeled with deterministic [103, 191] or Gaussian [23] models. Nevertheless, some recent works have explored more complex density models such as Normalizing Flows [252, 141, 15, 242] or EBM [52, 240]. Regarding applications, motion policies have been applied for reactive navigation [105, 191], Reinforcement Learning (RL) [218, 127, 241] and Imitation Learning [103, 242, 259].
Different works differ in the policy architecture. Different architectures impose different inductive biases into the policy. A wide set of works have explored the problem of integrating stability guarantees [103, 102, 242, 108, 49, 206]. Stability is a strong bias in the motion policies to avoid diverging motions. Imposing composability is also common in motion policies in order to satisfy multiple objectives jointly (collision avoidance, reaching a target ...) [105, 240, 259, 191, 233]. Multiple works have explored representing the motion policies in non-Euclidean spaces [239, 132, 10]. This allows the design of policies in the robot's end-effector space (Lie Group SE(3)) [239] or to guarantee safety constraints by representing the motion in the constraint manifold [131].

In Chapter 3 and Chapter 4, we focus our research on motion policies. In Chapter 3, we explore stability in motion policies, proposing novel motion policy architectures that combine stability with deep generative models. In Chapter 4, we explore composability in motion policies. We study the connections between EBM composable property and composable motion generation, proposing novel approaches for motion generation.

2.1.3. Motion Optimization

Motion optimization algorithms [188, 97, 22, 212, 153, 120] describe the motion generation problem as a constraint optimization problem

$$\begin{aligned}\boldsymbol{\tau}^* = \arg\min_{\boldsymbol{\tau}} & J(\boldsymbol{\tau}) \\ \text{s.t.} \quad & \gamma_i(\boldsymbol{\tau}) \leqslant 0, \quad i = 0, \dots, m\end{aligned} \tag{2.5}$$

with $J(\boldsymbol{\tau})$ being the objective function that should be minimized and γ_i constraint functions such as joint limits or task-dependant constraints. It is common to represent the objective function as a composition of multiple cost functions $c(\boldsymbol{\tau})$

$$J(\boldsymbol{\tau}) = \sum_k c_k(\boldsymbol{\tau}), \tag{2.6}$$

with each cost representing a different objective the robot should satisfy. These costs could represent trajectory smoothness, obstacle avoidance, self-collision avoidance, pose manipulability, or reaching goal poses to name a few. Additionally, it is common for the cost functions not to depend on the whole trajectory. For example, collision avoidance is described with respect to the state $c(\boldsymbol{s}_t)$, and trajectory smoothness is evaluated given two adjacent states $c(\boldsymbol{s}_t, \boldsymbol{s}_{t+1})$.

Motion optimization can be framed from a probabilistic inference perspective [234, 235, 126], providing an intuitive connection to generative models. From this view, motion optimization is cast as a Maximum a Posteriori (MAP) problem

$$\boldsymbol{\tau}^* = \arg\max_{\boldsymbol{\tau}} \log \rho(\boldsymbol{\tau}|J = 0) = \arg\max_{\boldsymbol{\tau}} \log \rho(J = 0|\boldsymbol{\tau}) + \log \rho(\boldsymbol{\tau}) \tag{2.7}$$

with $\rho(J = 0|\boldsymbol{\tau})$ being the likelihood of $\boldsymbol{\tau}$ being optimal for the objective J and $\rho(\boldsymbol{\tau})$ a prior distribution over $\boldsymbol{\tau}$. The likelihood function is described in terms of the objective function $\rho(J = 0|\boldsymbol{\tau}) \propto \exp(-J(\boldsymbol{\tau}))$. Note that from this view, we can represent a cost function as an EBM, $\rho(\boldsymbol{\tau}) \propto \exp(-c(\boldsymbol{\tau}))$ and also the constraints $\rho(\boldsymbol{\tau}) \propto \exp(-\gamma(\boldsymbol{\tau}))$.

Finding $\boldsymbol{\tau}^*$ in Equation (2.7) requires solving a nonlinear optimization problem, that is commonly solved by iterative optimization algorithms. We classify the algorithms between gradient-based optimization algorithms [188, 153] and sampling-based optimization [97, 243] algorithms. We present an example of a gradient-based optimization algorithm (CHOMP [188, 274]) in Algorithm 1 and an example of sampling-based optimization algorithm (STOMP [97]) in Algorithm 2.

Algorithm 1: Gradient-based Motion Optimization example (CHOMP) [188, 274]

```
Given : $\boldsymbol{\tau}$: Initial trajectory, $J$ objective function, $\boldsymbol{M}$:trajectory metric
, $\alpha$: step-size
for $i \leftarrow 0$ to $I-1$ do
    $\nabla j_{\boldsymbol{\tau}} = \nabla_{\boldsymbol{\tau}} J(\boldsymbol{\tau})$;          // Compute objective gradient on current $\boldsymbol{\tau}$
    $\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} - \alpha \boldsymbol{M} \nabla j_{\boldsymbol{\tau}}$ ;          // Update trajectory with metric $\boldsymbol{M}$
return $\boldsymbol{\tau}$;
```

Algorithm 2: Sampling-based Motion Optimization example (STOMP) [97]

```
Given : $\boldsymbol{\mu}_{\boldsymbol{\tau}}$: Initial trajectory mean, $J$ objective function, $\boldsymbol{M}$:trajectory metric
for $i \leftarrow 0$ to $I-1$ do
    $\boldsymbol{\tau}_0, \dots, \boldsymbol{\tau}_n \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\tau}}, \boldsymbol{\Sigma})$;          // Sample $n$ trajectories with mean $\boldsymbol{\mu}_{\boldsymbol{\tau}}$
    $j_{\boldsymbol{\tau}_0}, \dots, j_{\boldsymbol{\tau}_n} = J(\boldsymbol{\tau}_0), \dots, J(\boldsymbol{\tau}_n)$;  // Evaluate the value of each trajectory
    $\rho_{\boldsymbol{\tau}_0}, \dots, \rho_{\boldsymbol{\tau}_n} = \text{Softmax}(-1/\lambda j_{\boldsymbol{\tau}_0}, \dots, -1/\lambda j_{\boldsymbol{\tau}_n})$;     // Compute probabilities
    $\delta\boldsymbol{\tau} = \sum_{i=0}^{n} \rho_{\boldsymbol{\tau}_i} (\boldsymbol{\tau}_i - \boldsymbol{\mu}_{\boldsymbol{\tau}})$;          // Compute weighted traj. improvement
    $\boldsymbol{\mu}_{\boldsymbol{\tau}} \leftarrow \boldsymbol{\mu}_{\boldsymbol{\tau}} + \alpha \boldsymbol{M} \delta\boldsymbol{\tau}$ ;          // Update trajectory
return $\boldsymbol{\mu}_{\boldsymbol{\tau}}$;
```

CHOMP (Algorithm 1) applies a classical gradient-descent optimization with a minor change. In CHOMP, the gradient update is multiplied with a metric $\boldsymbol{M}$ that will induce the neighboring trajectory waypoints to update with similar gradients that reduce the overall acceleration of the trajectory. In contrast with gradient-based optimization methods that require an explicit gradient of the objective function, sampling-based motion optimization algorithms approximate the gradient with samples. In particular, in STOMP (Algorithm 2), the gradient is approximated inspired by the path integral method [232]. Despite these examples, there exists a vast literature in motion optimization, in which the problem is solved by Cross-Entropy Method [31] or by Optimal Transport [120] to name a few.

Optimization algorithms aim to search the optimal trajectory $\boldsymbol{\tau}^*$ by minimizing or maximizing the objective function $J(\boldsymbol{\tau})$. However, there are scenarios where the goal extends beyond finding a single optimal solution. Instead, the interest might lie in sampling multiple trajectories from a distribution that is informed by the objective function, expressed as $\boldsymbol{\tau} \sim \rho(\boldsymbol{\tau})$, where $\rho(\boldsymbol{\tau})$ is proportional to $\exp(-J(\boldsymbol{\tau}))$. In such cases, particularly when dealing with high-dimensional distributions, Markov Chain Monte Carlo (MCMC) methods become a powerful tool for generating samples [5].

Langevin Monte Carlo methods, in particular, have shown notable success with learned

Algorithm 3: Langevin Monte Carlo

```
Given : $\boldsymbol{\tau}$: Initial trajectory, $J$ objective function, $\alpha$: step-size
for $i \leftarrow 0$ to $I - 1$ do
    $\nabla j_{\boldsymbol{\tau}} = \nabla_{\boldsymbol{\tau}} J(\boldsymbol{\tau})$;          // Compute objective gradient on current $\boldsymbol{\tau}$
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$;          // Sample noise vector of the trajectory lenght
    $\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} - \alpha \nabla j_{\boldsymbol{\tau}} + \sqrt{2\alpha}\boldsymbol{\epsilon}$ ;          // Update trajectory
return $\boldsymbol{\tau}$;
```

objective functions [90, 244, 29]. We present an example of the Langevin Monte Carlo method in Algorithm 3. This approach resembles gradient-based optimization but incorporates an additional noise component $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ in the trajectory update. Beyond Langevin Monte Carlo, there exists a plethora of algorithms for sampling from unnormalized distributions, including Metropolis-adjusted Langevin Monte-Carlo [201], Annealed Langevin Dynamics [224], or Hamiltonian Monte Carlo [12] to name some.

2.2. Learning Motion Generators from data

The primary goal of a generative model is to learn a probability density model $\rho_{\boldsymbol{\theta}}(\boldsymbol{x})$, that accurately captures the underlying probability distribution of the data, denoted as $\rho_{\mathcal{D}}(\boldsymbol{x})$, where $\boldsymbol{x}$ represents the data variable. In Generative Modeling, we operate under the assumption that the true data distribution $\rho_{\mathcal{D}}(\boldsymbol{x})$ is unknown, and that we only have access to a finite set of samples drawn from this distribution. These samples form a dataset $\mathcal{D} : \{\boldsymbol{x}_n\}_{n=1}^{N}$ where N is the number of samples. The task of learning the generative model can then be formulated as an optimization problem, where the objective is to minimize the divergence between the learned distribution $\rho_{\boldsymbol{\theta}}(\boldsymbol{x})$ and the true data distribution $\rho_{\mathcal{D}}(\boldsymbol{x})$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{D}(\rho_{\mathcal{D}}(\boldsymbol{x}), \rho_{\boldsymbol{\theta}}(\boldsymbol{x})), \tag{2.8}$$

where $\mathbb{D}$ denotes the divergence distance. Note that the choice of the divergence varies between different generative models. For example, the Jensen-Shannon divergence is applied when training Generative Adversarial Networks (GAN) [67], while the Kullback-Leibler (KL) divergence is used for Variational Autoencoders (VAE) [106]. Once this distribution is captured, the learned model $\rho_{\boldsymbol{\theta}}(\boldsymbol{x})$ can generate new samples that look as if they were drawn from the data distribution.

Model	Model Type	In Robotics Literature
Energy-Based Model	Scalar Field	[50, 52, 65, 96]
Affordance Model	Scalar Field	[268, 216, 217, 63]
Diffusion Model	Vector Field	[244, 90, 29, 133]
Generative Adversarial Network	Sampler	[161, 124, 270, 150]
Variational Auto-Encoder	Sampler	[87, 152, 176, 140]
Normalizing Flow	Scalar Field/Sampler	[242, 187, 114, 260]

Table 2.1.: List of deep generative models applied for Motion Generation in robotics. The models are classified based on the output they generate. We refer to them as Scalar Field when the model outputs a scalar value, Vector Field for the models that output a vector and Sampler for the models explicitly generate samples.

There exists a wide myriad of deep generative models, from VAE [106] to EBM [121, 78], or Denoising Diffusion Probabilistic Models (DDPM) [222, 80]. The models differ from each other based on their training algorithms and the nature of their outputs. While some generative models explicitly generate samples (GAN, VAE) others require MCMC algorithms to generate samples (EBM, Noise Conditioned Score Network (NCSN) [224]). While some output a scalar value representing the energy of the learned density model (EBM), others output the score of the density model (NCSN, DDPM). In the context of motion generation, we cluster them based on their application in motion generation problems. We classify them into three types: Sampling models, Scalar Fields, and Vector Fields. In Table 2.1, we present a classification of the different generative models based on their application in motion generation problems and we visualize their differences in Figure 2.3.

Sampling Models are models that allow a fast and direct generation of samples $\boldsymbol{x} \sim \rho_{\boldsymbol{\theta}}(\boldsymbol{x})$. These models have been applied as motion policies [242] or as initial sampling distributions for motion planning problems [87, 161]. GAN, VAE or NFlow are some examples of generative models that are sampling models.
Scalar Fields are models that output a scalar value assigned to each input. They can output the log probability $\log \rho(\boldsymbol{x})$ or the unormalized log probability. In the context of motion generation, they are commonly used as cost functions $c(\boldsymbol{x})$ [50, 268]. Affordance models or EBM are Scalar Field type generative models.

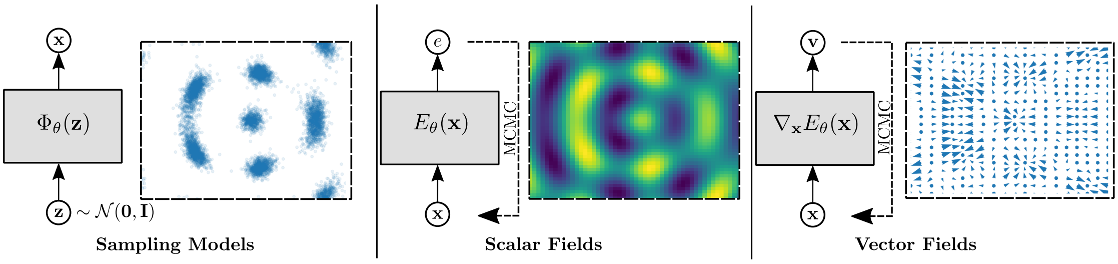

x
$\Phi_\theta(\mathbf{z})$
z $\sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Sampling Models
e
$E_\theta(\mathbf{x})$
MCMC
x
Scalar Fields
v
$\nabla_\mathbf{x} E_\theta(\mathbf{x})$
MCMC
x
Vector Fields

Figure 2.3.: An illustration of the three types of deep generative models: Sampling Models, Scalar Fields, and Vector Fields. Sampling Models generate the samples directly, while Scalar Fields and Vector Fields require an additional method such as MCMC to generate samples.

Vector Fields are models that output a vector representing the score of the learned distribution $\nabla_{\boldsymbol{x}} \log \rho(\boldsymbol{x})$. In the context of motion generation, they are integrated as the gradient of the cost function $\nabla_{\boldsymbol{x}} c(\boldsymbol{x})$. Gradient-based optimization methods do not use the cost value, but rather they follow the direction given by the gradient of the cost. Thus, it makes sense in these situations to represent the gradient field directly rather than the cost. Generative models such as NCSN, DDPM, or Flow Matching Models fall into this category.

In the following, we present the algorithms that have been used to train these models in robotics-related problems. We show the modifications that have been applied in the literature to adapt the training pipelines of these models to robotics-related problems.

2.2.1. Sampling Models

Sampling Models are the most straightforward approach to generate robot motion. In contrast with Scalar Fields or Vector Fields that require running a MCMC iterative process for n steps to generate samples (Algorithm 3), Sampling Models allow a direct generation of the samples, thus requiring a smaller computational time to obtain the samples. For the cases of VAE, GAN, and NFlow, the generation is divided into two steps. First, a sample is generated in a latent space $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Then, the sample is transformed given a learned function $\boldsymbol{x} = \Phi_{\boldsymbol{\theta}}(\boldsymbol{z})$.

In the field of Robotics, these types of models have been used both as policies $\pi(\boldsymbol{a}|\boldsymbol{s})$ learned by Behavioral Cloning [242], as initial sampling distribution for motion planning and optimization problems [161, 87, 107] or simply as generative models [124, 152]. Note that due to the high dimension of the trajectories, these types of models are usually exploited to generate lower dimensional outputs, such as grasp poses [161, 152], collision-free states [87] or inverse kinematics solutions [124, 4].

Training Sampling Models

There exist multiple deep generative models that fall in this class: GAN's generator and VAE's decoder are explicit sampling models. These models allow the generation of samples but don't allow the evaluation of samples. NFlow are both sampling models and scalar fields as they allow both the generation of samples and assesing the probability of a sample.

Normalizing Flows NFlow [198, 165, 27] are powerful because they can learn complex distributions through a series of invertible transformations, allowing for both efficient sampling and exact likelihood evaluation. Given a latent normal distribution $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I})$ and a learnable invertible neural network $\boldsymbol{x} = \Phi_{\boldsymbol{\theta}}(\boldsymbol{z})$, NFlow define the distribution on $\boldsymbol{x}$

$$\rho_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathcal{N}\left(\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x})|\boldsymbol{0}, \boldsymbol{I}\right) |\det \boldsymbol{J}_{\Phi_{\boldsymbol{\theta}}}(\boldsymbol{x})| \tag{2.9}$$

in terms of the latent distribution $\mathcal{N}(\boldsymbol{z}|\boldsymbol{0}, \boldsymbol{I})$ and the determinant of the Jacobian $\boldsymbol{J}_{\Phi_{\boldsymbol{\theta}}} = \partial\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x})/\partial\boldsymbol{x}$ [198]. Given a dataset $\boldsymbol{x} \sim \rho_{\mathcal{D}}(\boldsymbol{x})$, NFlow are trained by minimizing the negative log-likelihood

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= -\mathbb{E}_{\boldsymbol{x}\sim\rho_{\mathcal{D}}(\boldsymbol{x})}\left[\log \rho_{\boldsymbol{\theta}}(\boldsymbol{x})\right] \\ &= -\mathbb{E}_{\boldsymbol{x}\sim\rho_{\mathcal{D}}(\boldsymbol{x})}\left[\log \mathcal{N}\left(\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x})|\boldsymbol{0}, \boldsymbol{I}\right)\right] - \mathbb{E}_{\boldsymbol{x}\sim\rho_{\mathcal{D}}(\boldsymbol{x})}\left[\log |\det \boldsymbol{J}_{\Phi_{\boldsymbol{\theta}}}(\boldsymbol{x})|\right], \end{aligned} \tag{2.10}$$

that weights the probability between the latent distribution and the determinant of the Jacobian.

In robotics, NFlow are applied to learn a diverse set of variables. In [242, 206, 187], they are used to learn globally stable transition dynamics. In [115, 114], NFlow are used to generate informed states for sampling-based motion planning problems, improving the required time to find a solution. In [260], NFlow are used to generate grasp poses, while in [4], they are used to generate inverse kinematic solutions.

Generative Adversarial Networks Similar to NFlow, GAN frame the generation by first sampling from a latent normal distribution $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and then applying a learnable mapping $\boldsymbol{x} = \Phi_{\boldsymbol{\theta}}(\boldsymbol{z})$. Given a dataset $\boldsymbol{x} \in \rho_{\mathcal{D}}(\boldsymbol{x})$, GAN propose learning the generative model by a min-max problem between the generator model parameterized by $\boldsymbol{\theta}$ and a discriminator parameterized by ψ

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\psi}} \mathbb{E}_{\boldsymbol{x}\sim\rho_{\mathcal{D}}} \left[\log D_{\psi}(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \left[\log(1 - D_{\psi}(\Phi_{\boldsymbol{\theta}}(\boldsymbol{x})))\right]. \tag{2.11}$$

The discriminator aims to distinguish between real data samples and the fake samples generated by the generator, while the generator aims to produce samples that are indistinguishable from real data to the discriminator.

A common application of GAN in robotics is as prior sampling distribution for planning problems. In [270], an image of the collision scene is provided and generates images with collision-free informative states for sampling-based motion planning methods. Works such as [150] have explored the application of GAN for path planning. In [161], GAN are combined with optimization algorithms to generate grasp poses in a constrained manifold. In [124], GAN are applied to learn the solutions of an inverse kinematics problem.

Variational Auto Encoders Generating samples with VAE [106] follows the same procedure of GAN and NFlow. First, a sample is generated in a simple latent distribution $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and then a decoder maps the latent variable to our desired space $\boldsymbol{x} = \Phi_{\boldsymbol{\theta}}(\boldsymbol{z})$. The VAE also contains an encoder $\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z = \mathrm{Enc}(\boldsymbol{x})$ that generates a mean and standard deviation for a normal distribution in the latent space. The VAE is trained using a loss function that has two main parts: a reconstruction loss and KL divergence

$$\mathcal{L}(\boldsymbol{\theta}, \psi) = \mathbb{E}_{\boldsymbol{z}\sim\rho(\boldsymbol{z}|\boldsymbol{x})} \left[\mathbb{D}_{\mathrm{KL}}(\rho(\boldsymbol{z}|\boldsymbol{x}), \rho(\boldsymbol{z}))\right] + \mathbb{E}_{\boldsymbol{z}\sim\rho(\boldsymbol{z}|\boldsymbol{x})} \left[||\Phi(\boldsymbol{z}) - \boldsymbol{x}||_2^2\right] \tag{2.12}$$

where $\rho(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}|\mathrm{Enc}(\boldsymbol{x}))$ denotes a Gaussian with the mean $\boldsymbol{\mu}_z$ and standard deviation $\boldsymbol{\sigma}_z$ are the output of the encoder. $\rho(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is a Gaussian around zero. While the KL divergence term encourages the encoder to generate distributions close to $\rho(\boldsymbol{z})$, the reconstruction loss aims to decode a latent sample to look as similar as possible to the input $\boldsymbol{x}$.

In robotics, conditional VAE have been applied for different problems. In [87], a VAE is learned to generate collision-free states. Then, the generative model was exploited in sampling-based motion planning problems such as RRT [119], to generate informative samples. In [152], a VAE conditioned on a point cloud is trained to generate grasp poses. The generated poses are prior samples that are further optimized following the gradient of a classifier to improve the grasp pose quality. A common application is to learn skill embeddings [176, 140, 186]. Given a dataset, we can first train the decoder of a VAE to generate samples that resemble the data. Then, the learned model can be integrated into a RL problem, improving the exploration.

2.2.2. Scalar Fields

We use the term Scalar Fields to describe models that, in contrast to Sampling Models, output a scalar value assessing the quality of an input sample. Generating samples from these models, therefore, necessitates additional algorithms, such as rejection sampling or MCMC methods for high-dimensional variables. **An important property of Scalar Fields in contrast with Sampling models is their composability.** Given two scalar fields, we can combine them, generating samples that aim to maximize the probability of both distributions.

Learning scalar fields have been widely explored in IOC [96, 50] or IRL [273, 56] fields. Given a set of trajectory demonstrations $\boldsymbol{\tau} \in \mathcal{D}$ with $\boldsymbol{\tau} : \{\boldsymbol{s}_t, \boldsymbol{a}_t\}_{t:0}^{T}$, maximum-entropy IOC and IRL methods learn the stationary state-action distribution $\rho_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a})$. The distribution is usually represented with implicit generative models, such as EBM. Once the model is learned, it can be integrated as a cost function in a motion planning problem. In IRL, the learned model is used as a reward function in a RL problem. Nevertheless, the problem of learning scalar fields has been also explored in different situations in which the demonstrations are not trajectories. Affordance models [268, 216] propose learning discrete scalar fields, inpainting the probability over scene images.

In the following, we present the different algorithms that have been applied to train these scalar fields.

Training Scalar Fields

For the case in which the space is discrete, Cross-Entropy (CE) loss is the most common approach [268, 216, 217, 68, 63] to fit density models. For continuous spaces, Contrastive Divergence (CD)[78] and Noise Contrastive Estimation (NCE) [70] are two of the most popular training algorithms for EBM and widely exploited in robotics.

Cross-Entropy Loss While the most popular application is on classification problems, a wide set of works [142, 266, 268, 149, 216, 217, 68] have applied CE to train robot policies as **discrete distributions**. Given a state $\boldsymbol{s}$ or an observation $\boldsymbol{o}$, the learned model outputs n one-hot encoded probabilities $E_{\boldsymbol{\theta}}^{i}(\boldsymbol{o})$ for every action the robot can take, with i being the action chosen. Given a set of demonstrations $\boldsymbol{x} \in \mathcal{D}$, the CE loss is

$$\mathcal{L}(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x}\in\mathcal{D}}[\log \rho_{\boldsymbol{\theta}}(\boldsymbol{x})] = -\mathbb{E}_{\boldsymbol{x}\in\mathcal{D}}\left[\log \frac{\exp(E_{\boldsymbol{\theta}}^{\boldsymbol{x}})}{\sum_{\boldsymbol{y}\in\mathcal{X}} \exp(E_{\boldsymbol{\theta}}^{\boldsymbol{y}})}\right]. \tag{2.13}$$

with $\mathcal{X}$ being the set of all possible actions. The probability for a particular action $\boldsymbol{x} \in \mathcal{X}$ is computed via Soft-max.

A wide set of works known as Affordance models apply CE loss to train discrete action distributions. In these models, the learned energy model $E_{\boldsymbol{\theta}}(\boldsymbol{o})$ receives as input a visual context input $\boldsymbol{o}$ (2D image [142, 268], scene voxel [217], pointcloud [149] ...). Then, the model outputs an energy heatmap of the same shape as the visual context. In practice, each "pixel" is considered as a possible location the robot could move and the pixel one-hot map provides the likelihood of each pixel to solve the task. In [142, 266, 268, 216] the visual context is a 2D image and the output is a one-hot pixel map of the same shape. In [217], the input is a voxelized scene, and the output is a one-hot voxel map. In [63], a set of 3D ghost points are sampled as possible actions rather than the voxelized space, reducing the computational complexity and increasing in accuracy. In [68], the input is a list of 2D images from different views and the output is a heatmap on each image. The 3D action is then computed by triangularization. In [62, 149], the action space is placed directly in the point cloud, selecting the point at which we should interact with the objects in the scene.

Contrastive Divergence EBM represents a learnable distribution as a Boltzmann distribution

$$\rho_{\boldsymbol{\theta}}(\boldsymbol{x}) = \exp\left(-E_{\boldsymbol{\theta}}(\boldsymbol{x})\right)/Z_{\boldsymbol{\theta}}, \tag{2.14}$$

with $E_{\boldsymbol{\theta}}$ the learnable energy model and $Z_{\boldsymbol{\theta}} = \int_{\mathcal{T}} \exp\left(-E_{\boldsymbol{\theta}}(\boldsymbol{x})\right) \mathrm{d}\boldsymbol{x}$ the normalization constant. Given a set of demonstrations $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=0}^{k}$, the negative log-likelihood is given by

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= -\mathbb{E}_{\boldsymbol{x}_i \sim \mathcal{D}}\left[\log \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right] = \mathbb{E}_{\boldsymbol{x}_i \sim \mathcal{D}}\left[E_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right] + \log Z_{\boldsymbol{\theta}} \\ &= \mathbb{E}_{\boldsymbol{x}_i \sim \mathcal{D}}\left[E_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right] + \log \int_{\mathcal{T}} \exp\left(-E_{\boldsymbol{\theta}}(\boldsymbol{x})\right) \mathrm{d}\boldsymbol{x}. \end{aligned} \tag{2.15}$$

The computation of the normalization constant $Z_{\boldsymbol{\theta}}$ might be intractable when applying the negative log-likelihood for EBM. The computation of $Z_{\boldsymbol{\theta}}$ involves an integral over all possible trajectories, being computationally infeasible for sufficiently high-dimensional spaces.

A solution to tackle the computation of the normalization constant $Z_{\boldsymbol{\theta}}$ was proposed in CD [78]. In [78], it is shown that $\nabla_{\boldsymbol{\theta}} \log Z_{\boldsymbol{\theta}} = -\mathbb{E}_{\boldsymbol{x}\sim\rho_{\boldsymbol{\theta}}(\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\boldsymbol{x})\right]$. Then, we can numerically approximated by drawing samples from $\rho_{\boldsymbol{\theta}}(\boldsymbol{x})$

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[E_{\boldsymbol{\theta}}(\boldsymbol{x})\right] - \mathbb{E}_{\boldsymbol{x}\sim\rho_{\boldsymbol{\theta}}(\boldsymbol{x})}\left[E_{\boldsymbol{\theta}}(\boldsymbol{x})\right]. \tag{2.16}$$

In Equation (2.16), the energy is pushed down for the trajectories in the dataset (positive samples) and pushes up the energy for the rest of the trajectories not belonging to the dataset. On every iteration, we sample a set of points from the current energy model $\boldsymbol{x} \sim \rho_{\boldsymbol{\theta}}(\boldsymbol{x})$ (negative samples) where the energy is pushed up. Nevertheless, if the dimension of $\boldsymbol{x}$ is high, it might be difficult to properly sample $\boldsymbol{x} \sim \rho_{\boldsymbol{\theta}}(\boldsymbol{x})$.

Maximum entropy IOC and IRL methods apply a similar approach to CD to learn the stationary state-action distributions. Different methods propose different approaches to generate negative samples. In [96], rather than sampling from $\boldsymbol{x} \sim \rho_{\boldsymbol{\theta}}(\boldsymbol{x})$, the negative samples are generated by adding noise to the dataset trajectories. In [50], a maximum-entropy motion optimization problem is solved to generate the negative samples, and an importance sampling weight is added to balance the mismatch between $\rho_{\boldsymbol{\theta}}$ and the samples from the optimization problem. In [273], a maximum entropy RL problem is solved every step to find the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ that best matches the current learned model $E_{\boldsymbol{\theta}}$. Then, in [256], the problem in [273] is extended to model $E_{\boldsymbol{\theta}}$ with a neural network.

CD have been also applied for other tasks beyond IOC and IRL. In [65], multiple EBM are learned to represent different arrangements of the objects in a scene. Given two object poses, the EBM represents if object A is on the left or right of object B. It can also be used to represent if multiple objects are generating a shape such as a circle. Then, multiple EBM can be composed to generate complex arrangements. In [42], CD is applied to learn a state transition EBM, given a known policy, while in [41], the learned EBM represents the world dynamics.

Noise-Contrastive Estimation An alternative approach to train EBM is by framing the problem as a binary classification problem between the training data and a noise distribution [70, 45]. Given a distribution $\boldsymbol{x} \sim \rho_{\mathcal{D}}(\boldsymbol{x})$ of data samples with a binary class

$c = 1$ and a noise distribution $\boldsymbol{x} \sim q(\boldsymbol{x})$ with a binary class $c = 0$, the probability of c given a trajectory is

$$\rho_{\boldsymbol{\theta}}(c = 1|\boldsymbol{x}) = \frac{\rho_{\boldsymbol{\theta}}(\boldsymbol{x})}{\rho_{\boldsymbol{\theta}}(\boldsymbol{x}) + q(\boldsymbol{x})} \quad , \quad \rho_{\boldsymbol{\theta}}(c = 0|\boldsymbol{x}) = 1 - \rho_{\boldsymbol{\theta}}(c = 1|\boldsymbol{x}) \tag{2.17}$$

with $\rho_{\boldsymbol{\theta}} \approx \rho_{\mathcal{D}}$. Then, we frame a binary logistic regression problem

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x} \sim \rho_{\mathcal{D}}} \left[\log \rho_{\boldsymbol{\theta}}(c = 1|\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{x} \sim q} \left[\log(1 - \rho_{\boldsymbol{\theta}}(c = 1|\boldsymbol{x}))\right] \tag{2.18}$$

that implicitly learns the model $\rho_{\boldsymbol{\theta}}(\boldsymbol{x})$. The problem on NCE then lies in the choice of the negative samples distribution $q(\boldsymbol{x})$.

In robotics, similar approaches to NCE have been applied to learn EBM, both as cost functions [51, 56] and as policies [53]. In [50], the negative samples distribution $q(\boldsymbol{x})$ is iteratively learned by solving a maximum-entropy motion optimization problem. Then, in [56], a similar approach is extended but learning a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ in a maximum-entropy RL problem. In [53], InfoNCE is applied to learn a policy in an implicit behavioral cloning problem.

2.2.3. Vector Fields

During the last years, vector field-based generative models such as DDPM [80] or NCSN [224, 225] have shown state-of-the-art results in image generation and have started to be used in robotics. Rather than learning explicit sampling algorithms such as GAN or VAE, or rather than learning a scalar field as an EBM, these models propose learning a vector field $\boldsymbol{v} : \mathbb{R}^n \to \mathbb{R}^n$. This vector field is related to the score of the data distribution and it is integrated into MCMC algorithms such as Langevin Monte Carlo (Algorithm 3) to generate samples. Intuitively, given a dataset $\boldsymbol{x} \in \mathcal{D}$, the learned vector field represents a field that for any point in the space, outputs a vector that points towards the dataset. By following the vector fields, a randomly generated sample is transported towards high-probability areas of the data distribution.

In robotics, vector field models have been applied to generate trajectories [90, 29, 25, 71], grasp poses [244], object placements [220], scene arrangements [133] or video plans [43, 44]. In the context of motion optimization, vector field generative models have been integrated as cost gradients, defining task constrains [262], grasp costs [244] or trajectory priors [25].

Training Vector Fields

Vector field generative models are commonly trained by **score-matching**. Given a dataset $\rho_{\mathcal{D}}(\boldsymbol{x})$, we aim to learn a model that matches the score of the data distribution

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim\rho_{\mathcal{D}}(\boldsymbol{x})}\left[||\nabla_{\boldsymbol{x}}\log\rho_{\boldsymbol{\theta}}(\boldsymbol{x}) - \nabla_{\boldsymbol{x}}\log\rho_{\mathcal{D}}(\boldsymbol{x})||_2^2\right] \tag{2.19}$$

with $\log\rho_{\boldsymbol{\theta}}(\boldsymbol{x})$ the learned model's score. In practise, it is common to directly learn the score function $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}}\log\rho_{\boldsymbol{\theta}}(\boldsymbol{x})$, as a vector field; yet some works [226, 244] have explored learning EBM by matching its gradient, $\nabla_{\boldsymbol{x}}E_{\boldsymbol{\theta}}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}}\log\rho_{\boldsymbol{\theta}}(\boldsymbol{x})$.

Applying score-matching directly is difficult. In Generative Modeling the data distribution $\rho_{\mathcal{D}}(\boldsymbol{x})$ is unknown and we have access only to samples from it. Thus, we don't have access to the score of the data distribution $\nabla_{\boldsymbol{x}}\log\rho_{\mathcal{D}}(\boldsymbol{x})$. A possibility to tackle this problem is by denoising score matching.

Denoising Score Matching An approach to approximate the score of the data distribution $\boldsymbol{x}\sim\rho_{\mathcal{D}}(\boldsymbol{x})$ is by Denoising Score Matching (DSM) [249, 224]. In DSM, we design a noisy data distribution by adding a noise kernel over the data distribution $\rho_{\hat{\mathcal{D}}}(\hat{\boldsymbol{x}}) = \int_{\boldsymbol{x}}\rho(\hat{\boldsymbol{x}}|\boldsymbol{x})\rho_{\mathcal{D}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$, with $\rho(\hat{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}, \sigma\boldsymbol{I})$, a Gaussian kernel. A sample from the distribution $\rho_{\hat{\mathcal{D}}}(\hat{\boldsymbol{x}})$ can be generated by first sampling from the data distribution $\boldsymbol{x}\sim\rho_{\mathcal{D}}(\boldsymbol{x})$ and then adding white noise $\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\epsilon}$, with $\boldsymbol{\epsilon}\sim\mathcal{N}(\boldsymbol{0}, \sigma\boldsymbol{I})$. Then, in DSM, the score matching is computed with respect to $\rho_{\hat{\mathcal{D}}}(\hat{\boldsymbol{x}})$ with

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim\rho_{\mathcal{D}}(\boldsymbol{x}),\hat{\boldsymbol{x}}\sim\rho(\hat{\boldsymbol{x}}|\boldsymbol{x})}\left[||\nabla_{\hat{\boldsymbol{x}}}\log\rho_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}) - \nabla_{\hat{\boldsymbol{x}}}\log\rho(\hat{\boldsymbol{x}}|\boldsymbol{x})||_2^2\right]. \tag{2.20}$$

Note that computing the score-matching for $\rho_{\hat{\mathcal{D}}}(\hat{\boldsymbol{x}})$ only requires to compute the score of the kernel that is trivial to compute.

A limitation of DSM is that it learns a model that matches the noisy distribution $\rho_{\hat{\mathcal{D}}}(\hat{\boldsymbol{x}})$. Then, if we use this model to generate samples, they might be noisy. To tackle this problem, **Diffusion Models** propose learning a time-dependant score model that smoothly transitions from high-entropy simple distributions to the data distribution by smoothly reducing the noise.

Diffusion Models propose learning a time-conditioned score-based model that informs about the evolution of distribution $\rho_t(\hat{\boldsymbol{x}})$. When $t = 0$, the model matches an easy-to-sample distribution as a normal distribution $\rho_0(\hat{\boldsymbol{x}}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and at $t = T$, the distribution matches the data distribution $\rho_T(\hat{\boldsymbol{x}}) = \rho_{\mathcal{D}}(\hat{\boldsymbol{x}})$. By smoothly transitioning from ρ_0 to ρ_T, we can transport a sample generated in ρ_0 to the data distribution.

Similarly to DSM, we define a kernel that transforms the data distribution. In the case of Diffusion Models, the kernel is a time-conditioned kernel $\rho_t(\hat{\boldsymbol{x}}) = \int_{\boldsymbol{x}} \rho_t(\hat{\boldsymbol{x}}|\boldsymbol{x})\rho_{\mathcal{D}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$. Then, the model is learned to match the score of the time-conditioned model

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}\sim\rho_{\mathcal{D}}(\boldsymbol{x}),\hat{\boldsymbol{x}}\sim\rho_t(\hat{\boldsymbol{x}}|\boldsymbol{x})} \left[||\nabla_{\hat{\boldsymbol{x}}} \log \rho_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}, t) - \nabla_{\hat{\boldsymbol{x}}} \log \rho_t(\hat{\boldsymbol{x}}|\boldsymbol{x})||_2^2 \right]. \tag{2.21}$$

Different approaches have proposed different kernels to represent the distribution $\rho_t(\hat{\boldsymbol{x}})$. In NCSN [224], the kernel is $\rho_t(\hat{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}, \sigma_t \boldsymbol{I})$, with σ_t a scalar value that is $\sigma_0 = \sigma_{\text{max}}$ when time is $t = 0$ and $\sigma_T \approx 0$ when time is $t = T$. This kernel leads to a high-entropy distribution at the beginning and smoothly reduces the entropy to $\rho_{\mathcal{D}}$. In DDPM [80], the kernel is $\rho_t(\hat{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\sqrt{1-\alpha_t}\boldsymbol{x}, \alpha_t \boldsymbol{I})$, with α_t is a scalar that at $t = 0$, $\alpha_0 = 1$ and at $t = T$, $\alpha_T = 0$. This kernel induces the distribution to be $\rho_0(\hat{\boldsymbol{x}}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ at time $t = 0$ and data distribution at time $t = T$.

In robotics, both NCSN and DDPM have been applied to represent trajectories [90, 29, 257, 25], scene arrangements [133], grasp poses [244], objects placing poses [220] or to generate tactile images [77]. We highlight [244, 220] that adapt NCSN and DDPM to the Lie group SE(3) to properly represent the grasp and placing poses. Diffusion Models have been also integrated into motion optimization problems as additional cost functions [244, 269, 25]. Given the Diffusion Model outputs a vector, they have been integrated as the gradient of the cost function to optimize with gradient-based optimization methods.

3. Globally Stable Policies with Flow-Based Models

In this chapter, we introduce ImitationFlows (iFlows) , a novel deep generative model that allows learning globally stable, stochastic, nonlinear dynamics, also known as Stable Vector Fields (SVF). Our approach extends the Normalizing Flows framework to learn **stable** Stochastic Differential Equations. Thanks to the architectural properties of the model, we prove that any solution our model generates is asymptotically stable in terms of Lyapunov. Our model extends the set of stable dynamical systems that can be represented by state-of-the-art approaches (limited to mixture of linear models) outperforming the previous models in terms of representation accuracy.

After presenting iFlows, we show how to extend the model to non-Euclidean manifolds. Learning robot motions from demonstration requires models able to specify policies for the full robot pose (6D) when the task is defined in the end-effector space. Despite the translation movement can be represent in an Euclidean space, representing vector fields for rotations requires to consider the geometrical properties of the rotation space. Rotations in 3D are usually represented in the SO(3) group or in the Quaternions manifold. In this second part, we present a novel vector field model that can guarantee most of the properties of iFlows i.e., stability, smoothness, and reactivity beyond the Euclidean space. In the experimental evaluation, we show the performance of our proposed vector field model to learn stable vector fields for full robot poses as SE(2) and SE(3) in both simulated and real robotics tasks.

3.1. Introduction

Data-driven motion generation methods (Imitation Learning (IL) [209, 1]) bring the promise of teaching our robots the desired behavior from a set of demonstrations without further programming of the robot skill. Similarly to the benefits in generalization when choosing a CNN networks to represent models in computer vision, choosing good models for motion generators might help in the quality of the robot's performance, when learning a policy directly from data. During the last two decades, there has been vast research on learning policy architectures [207, 88, 103, 166, 23] that guarantee a set of desirable inductive biases for robotics. Popularized as Movement Primitive (MP), the community explored a wide set of policy architectures with inductive biases such as Smoothness [166], Stability [207, 103] or cyclic performance [112, 88].

One of the main difficulties on MP is to guarantee globally stable behaviors: While several models exist [210, 76, 85]; most of them are not good ensuring stable dynamics out of the region of the demonstrated trajectories. **Global stability in dynamic systems refers to the system's ability to return to a steady state or equilibrium, regardless of the initial conditions or disturbances it experiences (See Figure 3.1).** This property is essential to guarantee safe robot behaviors beyond the demonstrated area. A precursor approach to tackle this difficulty had been given by the Stable Estimator of Dynamical Systems (SEDS) algorithm [103]. In this model, the global asymptotically stability is ensured by a structured mixture of linear dynamics. However, SEDS assumes a quadratic Lyapunov function. Due to the proposed Lyapunov function, the learned dynamics are restricted to continuously decreasing distance towards the attractor. In order to overcome the limitation of SEDS, different approaches were proposed [125, 102, 192, 159, 175, 194, 221, 108]. However, most of the approaches lack expressivity, leading to poorly representing the demonstrations.

Contribution In this chapter, we present *ImitationFlow*, a novel model to represent and learn globally stable nonlinear dynamical systems. Our methodology merges deep generative models like Normalizing Flows [198, 38, 164] with stable dynamical systems. Our approach not only is capable of representing a wider class of dynamical systems w.r.t. previous works in the field, but it can also describe both strike-based and periodic movements in a single framework, without changing the core learning algorithm.

The rest of the chapter is divided into five sections:

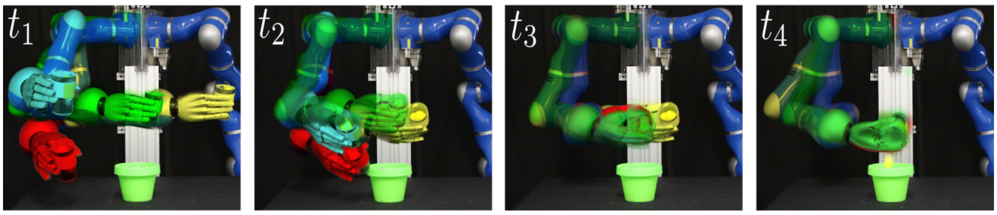


Figure 3.1.: Robot pouring trajectories generated by $SE(3)$-stable vector fields. Each color represents a trajectory starting from a different initial configuration. Given the stability properties, all the trajectories end up with the same orientation and position on the end effector.

Section 2 presents our dynamics model, ImitationFlows. We provide the required background and describe our method.

Section 3 extends ImitationFlows to Lie Groups. We motivate the need for representing stable vector fields in Lie Groups and show the required modifications for adapting ImitationFlows to non-Euclidean manifolds.

Section 4 shows the experimental evaluation. We evaluate the model to a different set of tasks in different manifolds.

Section 5 describes the state of the art. We present previous works on stable vector fields and Normalizing Flows on Manifolds.

Section 6 presents the conclusions and provide directions for future work.

3.2. Learning Stable Vector Fields with Normalizing Flows

iFlows are build on top of two pillars: Stochastic Differential Equations and Normalizing Flows. To properly understand the work, we first provide a background in these two topics. Then, we present our proposed model and show the different training algorithms to learn desired behaviors. We present the stability guarantees of our model in Appendix A.1.

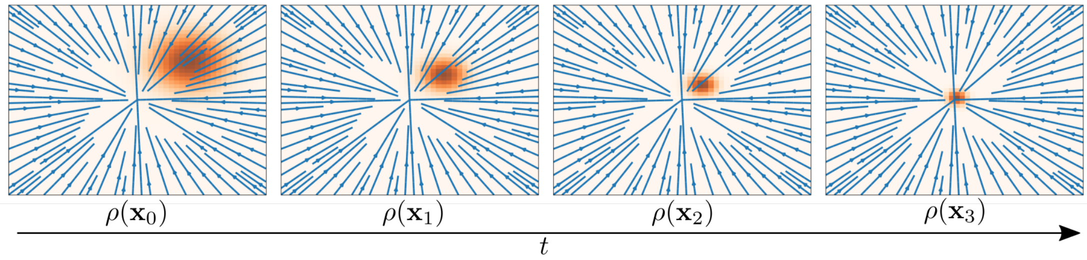


Figure 3.2.: Evolution of the state distribution under linear dynamics. Given the dynamics are linear, the state distribution is Gaussian in any instant of time.

3.2.1. Preliminaries

Stochastic Differential Equations Given $\boldsymbol{x}_t \in \mathbb{R}^n$ is the n-dimensional position at time t for a robot, we assume its motion is generated by a Stochastic Differential Equation (SDE)

$$d\boldsymbol{x}_t = \boldsymbol{g}(\boldsymbol{x}_t)\mathrm{d}t + \boldsymbol{\sigma}(\boldsymbol{x}_t)\mathrm{d}\boldsymbol{W}_t, \tag{3.1}$$

where $\boldsymbol{g}$ is the drift function, $\boldsymbol{\sigma}$ is the diffusion function, and $\boldsymbol{W}_t$ is the *Brownian motion* (also called *Wiener process*).

For the particular case in which $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ is a linear function and $\boldsymbol{\sigma}$ constant, we can write the discrete time SDE

$$\begin{aligned} \boldsymbol{x}_{t+1} &= (\boldsymbol{I} + \boldsymbol{A}\Delta t)\boldsymbol{x}_t + \boldsymbol{\epsilon} \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}), \end{aligned} \tag{3.2}$$

with $\boldsymbol{\Sigma} = \Delta t \boldsymbol{\sigma}^2 \boldsymbol{I}$ and $\Delta t > 0$ the discretization time. From a probabilistic view, we can represent the stochastic dynamics in Equation (3.2) as a transition probability function

$$\rho(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) = \mathcal{N}((\boldsymbol{I} + \boldsymbol{A}\Delta t)\boldsymbol{x}_t, \Delta t \boldsymbol{\sigma}^2 \boldsymbol{I}), \tag{3.3}$$

with $\boldsymbol{\mu} = (\boldsymbol{I} + \boldsymbol{A}\Delta t)\boldsymbol{x}_t$ the mean and $\boldsymbol{\Sigma} = \Delta t \boldsymbol{\sigma}^2 \boldsymbol{I}$ the covariance matrix.

Let us consider a probability distribution over the initial state, $\rho(\boldsymbol{x}_0)$. Then, given the transition dynamics $\rho(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$, the probability for the trajectory $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$

$$\rho(\boldsymbol{\tau}) = \rho(\boldsymbol{x}_0) \prod_{t=0}^{T-1} \rho(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t), \tag{3.4}$$

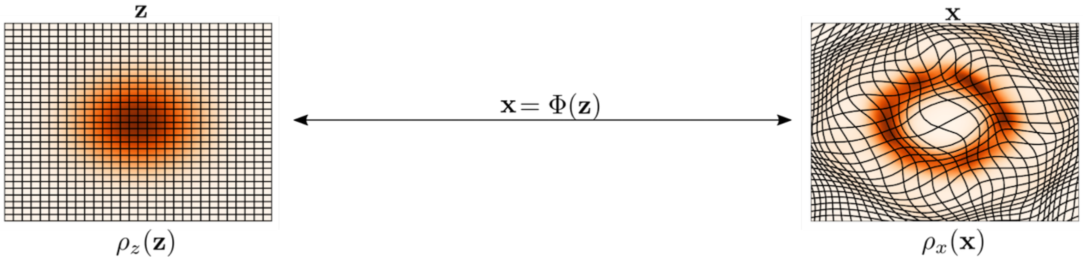


Figure 3.3.: A diffeomorphism Φ can be thought as a space deformation, depicted with black lines. Due to this deformation, a density in $\boldsymbol{z}$, $\rho_z(\boldsymbol{z})$ will be reshaped in $\boldsymbol{x}$, $\rho_s(\boldsymbol{x})$ and is described with Equation (3.7).

is an autoregressive model.To generate a trajectory, we can initially sample from $\boldsymbol{x}_0 \sim \rho(\boldsymbol{x}_0)$ and iteratively call to the transition dynamics, $\rho(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$.

An interesting property for our work happens when the initial distribution is a normal distribution $\rho(\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \Sigma_0)$. Given the transition dynamics are linear wrt. the conditioning state, we can easily observe that the state distribution is a normal distribution for any instant of time (See Figure 3.2)

$$\rho(\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t), \tag{3.5}$$

that can be written in terms of the initial distribution parameters and the transition dynamics

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \boldsymbol{A}\boldsymbol{\mu}_t \\ \boldsymbol{\Sigma}_{t+1} &= \boldsymbol{A}^{\intercal}\boldsymbol{\Sigma}_t\boldsymbol{A} + \boldsymbol{\Sigma}. \end{aligned} \tag{3.6}$$

Normalizing Flows Let us consider a normal distribution in a latent space $\boldsymbol{z} \in \mathbb{R}^n$, $\rho_z(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and a diffeomorphic function $\Phi_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^n$ that maps the latent space $\boldsymbol{z}$ and the base space $\boldsymbol{x} \in \mathbb{R}^n$. A diffeomorphic function is a function that is both bijective $\boldsymbol{z} = \Phi^{-1}(\Phi(\boldsymbol{z}))$ and differentiable. Given the function Φ is a diffeomorphism, the density over the variable $\boldsymbol{x}$ is computed by

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = p_z(\boldsymbol{z}) \left|\det \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right| = p_z(\Phi^{-1}(\boldsymbol{x})) \left|\det \frac{\partial \Phi^{-1}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|. \tag{3.7}$$

NFlow are a class of deep generative models that propose representing the diffeomorphic function Φ with a deep neural network. To be diffeomorphic these functions have to be invertible and thus, they are popularly known as Invertible Neural Networks (INN). NFlow literature has explored different network architectures to represent diffeomorphic networks [198, 164, 38, 27]. Given a dataset of demonstrations $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=0}^{N}$, NFlow learn the parameters by solving a Maximun Likelihood Estimation (MLE) problem

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\log \rho_{\boldsymbol{\theta}}(\boldsymbol{x})\right] \\ &= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\log p_z(\Phi^{-1}(\boldsymbol{x}))\right] + \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\log\left|\det \frac{\partial\Phi^{-1}(\boldsymbol{x})}{\partial\boldsymbol{x}}\right|\right].\end{aligned} \tag{3.8}$$

3.2.2. Modeling Stable Vector Fields with Normalizing Flows

ImitationFlows extend the Normalizing Flows to represent dynamic systems. The proposed work is similar intuitively to Structured Inference Networks for Nonlinear State Space Models [110].

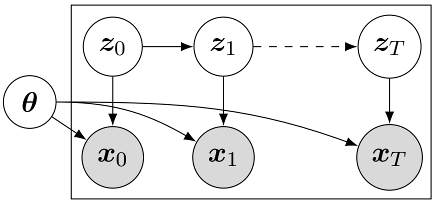


Figure 3.4.: ImitationFlows architecture as a graphical model. We run linear stochastic dynamics in the latent space $\boldsymbol{z}$. The mapping from $\boldsymbol{x}_t$ to $\boldsymbol{z}_t$ is an Invertible Neural Network.

The proposed model's architecture is presented in Figure 3.4, and the dynamics are modelled using Equation (3.9). Our model is composed of two main components. In the latent space $\mathcal{Z}$, the transition model follows linear stable stochastic dynamics. Then, as emission function, a diffeomorphic function $\Phi_{\boldsymbol{\theta}} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ transforms the state variable from the latent space $\mathcal{Z}$ to the base space $\mathcal{X}$

$$\begin{aligned} d\boldsymbol{z}_t &= \boldsymbol{A}\boldsymbol{z}_t\mathrm{d}t + \boldsymbol{\sigma}\mathrm{d}\boldsymbol{W}_t \\ \boldsymbol{x}_t &= \Phi_{\boldsymbol{\theta}}(\boldsymbol{z}_t), \end{aligned} \tag{3.9}$$

where $\boldsymbol{\theta}$ are the learnable parameters of the model. Given that the Jacobian of $\Phi_{\boldsymbol{\theta}}$, $\boldsymbol{J}_{\boldsymbol{\theta}} = \frac{dx}{dz}$, is easy to compute, we can reframe Equation (3.9) to compute the stochastic dynamic model for $\boldsymbol{x}_t$

$$d\boldsymbol{x}_t = J_{\boldsymbol{\theta}}(\boldsymbol{x}_t)\left(\boldsymbol{A}\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_t)\mathrm{d}t + \boldsymbol{\sigma}\mathrm{d}\boldsymbol{W}_t\right). \tag{3.10}$$

Algorithm 4: Density Matching for iFlows

```
Given : $\mathcal{D} = \{\boldsymbol{\tau}\}_N$, with $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$
for $i \leftarrow 0$ to $I - 1$ do
    $(\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T) \sim \mathcal{D}$;          // Sample trajectory from the dataset
    $\mathcal{L}(\boldsymbol{\theta}) = \log \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_0) + \sum_{t=0}^{T-1} \log \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$; // Match distribution with model
     eq. (3.13) and eq. (3.15)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ ;                    // Update model parameters $\boldsymbol{\theta}$
return $\boldsymbol{\theta}$;
```

Learning with ImitationFlows Given a set of trajectory demonstrations $\mathcal{D} = \{\boldsymbol{\tau}_k\}_{k=0}^N$ where $\boldsymbol{\tau}_i = (\boldsymbol{x}_0, \boldsymbol{x}_1 \dots, \boldsymbol{x}_T)$, we aim to learn the parameters of a density model to represent the data distribution. In this work, the distribution of the trajectory is framed as an autoregressive model

$$\rho_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_0) \prod_{t=0}^{T-1} \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t), \tag{3.11}$$

where $\rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$ denotes the transition dynamics and $\rho_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$ the initial distribution.

Given the model in Figure 3.4, the transition dynamics can be represented in terms of the latent dynamics and the diffeomorphism. Given $\boldsymbol{z} = \Phi^{-1}(\boldsymbol{x})$ we can substitute the conditioning variable

$$\rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) = \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{z}_t) = \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\Phi^{-1}(\boldsymbol{x}_t)). \tag{3.12}$$

Note that this change in the conditioning variable is valid as long as the mapping between $\boldsymbol{x}$ and $\boldsymbol{z}$ is deterministic. Additionally, given Equation (3.7)

$$\rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) = \rho_z(\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_{t+1})|\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_t)) \left|\det \boldsymbol{J}(\boldsymbol{x}_{t+1})\right|, \tag{3.13}$$

we can represent the transition dynamics in $\boldsymbol{x}$ in terms of the latent transition dynamics $\rho_z(\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_{t+1})|\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_t)) = \rho_z(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t)$ and the determinant of the Jacobian of the diffeomorphism $|\det \boldsymbol{J}(\boldsymbol{x}_{t+1})|$. We consider the latent transition dynamics $\rho_z(\boldsymbol{z}_{t+1}|\boldsymbol{z}_t)$ to be linear as in Equation (3.3).

Given a set of demonstrations $\mathcal{D} : \{\boldsymbol{\tau}_k\}_{k=0}^N$, we can now frame the problem as MLE problem

Algorithm 5: Density Matching in velocity for iFlows

```
Given : $\mathcal{D} = \{\boldsymbol{\tau}\}_N$, with $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$
for $i \leftarrow 0$ to $I-1$ do
    $(\boldsymbol{x}_0, \dot{\boldsymbol{x}}_0, \boldsymbol{x}_1, \dot{\boldsymbol{x}}_1 \dots, \boldsymbol{x}_T) \sim \mathcal{D}$;        // Sample trajectory from the dataset
    $\mathcal{L}(\boldsymbol{\theta}) = \log \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_T) + \sum_{t=0}^{T-1} \log \rho_{\boldsymbol{\theta}}(\dot{\boldsymbol{x}}_t|\boldsymbol{x}_t)$;    // Match distribution with model
      eq. (3.16) and eq. (3.15)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ ;                          // Update model parameters θ
return $\boldsymbol{\theta}$;
```

over the model $\rho_{\boldsymbol{\theta}}(\boldsymbol{\tau})$

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\tau}\sim\mathcal{D}} \left[\log \rho_{\boldsymbol{\theta}}(\boldsymbol{\tau})\right] \\ &= \mathbb{E}_{\boldsymbol{\tau}\sim\mathcal{D}} \left[\log \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_0) + \sum_{t=0}^{T-1} \log \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)\right] \end{aligned} \tag{3.14}$$

that decouples the distribution between the initial distribution $\rho_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$ and the transition dynamics distribution $\rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t)$ (See Algorithm 4). Note that the initial distribution in $\boldsymbol{x}_0$ can be also represented as a Normalizing Flow

$$\rho_{\boldsymbol{\theta}}(\boldsymbol{x}_0) = \rho_z(\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_0)) \left|\det \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}_0)\right| \tag{3.15}$$

An interesting alternative to represent the transition dynamics distribution is in terms of the the stochastic dynamics in Equation (3.10). The stochastic dynamics are Gaussian

$$\rho_{\boldsymbol{\theta}}(\dot{\boldsymbol{x}}_t|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)\boldsymbol{A}\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_t), \boldsymbol{J}_{\boldsymbol{\theta}}^{\intercal}(\boldsymbol{x}_t)\boldsymbol{\sigma}^2\boldsymbol{I}\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)) \tag{3.16}$$

with mean $\boldsymbol{\mu} = \boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)\boldsymbol{A}\Phi_{\boldsymbol{\theta}}^{-1}(\boldsymbol{x}_t)$ and variance $\boldsymbol{\Sigma} = \boldsymbol{J}_{\boldsymbol{\theta}}^{\intercal}(\boldsymbol{x}_t)\boldsymbol{\sigma}^2\boldsymbol{I}\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$. Then, we can represent the transition dynamics as the marginalization over the dynamics

$$\rho_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t) = \int_{\dot{\boldsymbol{x}}_t} \rho_{\boldsymbol{\theta}}(\dot{\boldsymbol{x}}_t|\boldsymbol{x}_t) q(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \dot{\boldsymbol{x}}_t) \mathrm{d}\dot{\boldsymbol{x}}_t \tag{3.17}$$

with $q(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \dot{\boldsymbol{x}}_t)$ being the deterministic dynamics integration. Then, the MLE problem can be represented wrt. $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$

$$\begin{aligned} \boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\tau}\sim\mathcal{D}} \left[\log \rho_{\boldsymbol{\theta}}(\boldsymbol{\tau})\right] \\ &= \mathbb{E}_{\boldsymbol{\tau}\sim\mathcal{D}} \left[\log \rho_{\boldsymbol{\theta}}(\boldsymbol{x}_T) + \sum_{t=0}^{T} \log \rho_{\boldsymbol{\theta}}(\dot{\boldsymbol{x}}_t|\boldsymbol{x}_t)\right] \end{aligned} \tag{3.18}$$

with the transition dynamics now represented in terms of the stochastic dynamics. An interesting difference of this approach is that dynamics model is Gaussian, leading to a simpler learning algorithm (See Algorithm 5).

3.3. From Euclidean spaces to Lie Groups

Learning Movement Primitives for orientations requires additional insights in the architecture of the model. There exist multiple representation forms for the orientation, such as Euler angles, rotation matrices, or quaternions. Euler angles have an intuitive representation, but the representation is not unique and might get stuck in singularities (i.e. gimbal lock). These properties make Euler angles undesirable for reactive motion generation [263]. Instead of Euler angles, rotation matrices and quaternions are preferred representations for reactive motion generation. Nevertheless, they require special treatment, given they are not defined in the Euclidean space. Rotation matrices are represented by the special orthogonal group, SO(3), while quaternions are represented in the 3-sphere, $\mathcal{S}^3$. Thus, in the context of modeling orientation MP, there has been wide research integrating manifold constraints and MP. In [171, 109, 237], Dynamic Movement Primitives (DMP) [207] were adapted to learn orientation DMP, by representing DMP for quaternions [171, 109, 237] or rotation matrices [237]. More recently, orientation MP have been also considered to adapt Kernelized Movement Primitives (KMP) [85, 86], Task Parameterized GMM (TP-GMM) [23, 265] and ProMP [166, 202]. Nevertheless, most of the MP are rather phase dependant or lack stability guarantees.

In the following, we introduce a novel learnable SVF function that can generate stable motions on Lie Groups. The proposed function generalizes iFlows to arbitrary smooth manifolds such as Lie Groups. Representing SVF on Lie groups allow us designing robot motion for end-effector's position and orientations. To learn these SVF, we propose a neural network architecture that represents diffeomorphic functions in robotic-relevant Lie Groups such as SE(2) and SE(3). In the experimental section, we compare the performance of our proposed model w.r.t. learning the vector fields for Euler angles and learning the vector fields in the configuration space of the robot.

3.3.1. Background

A n-*manifold* $\mathcal{M}$ is called *smooth* if it is locally diffeomorphic to an Euclidean space $\mathbb{R}^n$ [122]. For each point $\boldsymbol{x} \in \mathcal{M}$, there exist a coordinate chart (U, ψ), were U is an open

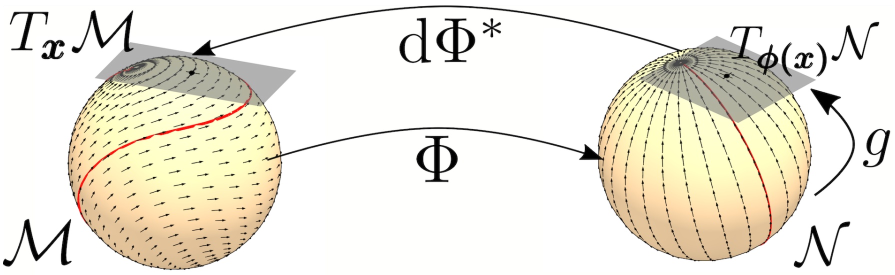

$T_{\boldsymbol{x}}\mathcal{M}$
$\mathrm{d}\Phi^*$
$T_{\boldsymbol{\phi}(\boldsymbol{x})}\mathcal{N}$
Φ
g
$\mathcal{M}$
$\mathcal{N}$

Figure 3.5.: In our work, we compute the vector field in $\mathcal{M}$ by pulling back the vector field from the latent manifold $\mathcal{N}$. Given a point $\boldsymbol{x} \in \mathcal{M}$, we first map it to the latent manifold $\boldsymbol{z} = \Phi(\boldsymbol{x})$ with $\boldsymbol{z} \in \mathcal{N}$. Then, we compute the vector in the latent manifold. Given a vector field $\boldsymbol{g} : \mathcal{N} \to T\mathcal{N}$, we compute $\dot{\boldsymbol{z}} \in T_{\Phi(\boldsymbol{x})}\mathcal{N}$. Finally, we apply the pullback linear operator to compute $\dot{\boldsymbol{x}} = \mathrm{d}\Phi^*_{\boldsymbol{x}}(\dot{\boldsymbol{z}})$ in the tangent space of $\mathcal{M}$, $\dot{\boldsymbol{x}} \in T_{\boldsymbol{x}}\mathcal{M}$. As we can observe, the diffeomorphic function Φ will deform the space and a trajectory (red line) or a vector field in the manifold $\mathcal{N}$ will be deformed in $\mathcal{M}$.

subset in the manifold, $U \subseteq \mathcal{M}$ and $\psi : U \to \hat{U}$, is a diffeomorphism from the subset U to a subset in the Euclidean space $\hat{U} \subseteq \mathbb{R}^n$. This chart allows us to represent a section of the manifold $\mathcal{M}$ in a Euclidean space and do calculus.

For any point in the manifold, $\boldsymbol{x} \in \mathcal{M}$, we can attach a *tangent space*, $T_{\boldsymbol{x}}\mathcal{M}$ that contains all the possible vectors that are tangential at $\boldsymbol{x}$. Intuitively, for any possible curve in $\mathcal{M}$ passing through $\boldsymbol{x}$, the velocity vector of the curve at $\boldsymbol{x}$ will belong to the tangent space, $\boldsymbol{v} \in T_{\boldsymbol{x}}\mathcal{M}$. Thus, a *vector field* in the manifold $\mathcal{M}$ is a function that maps any point in the manifold to a vector in the tangent space[1], $\boldsymbol{g} : \mathcal{M} \to T\mathcal{M}$. The *LogMap* is the map that moves a point in the manifold $\mathcal{M}$ to the tangent space, and the *ExpMap* is the map that moves a point from the tangent space to the manifold.

A map $\Phi : \mathcal{M} \to \mathcal{N}$ between smooth manifolds induces a linear map between their corresponding tangent spaces. For any point $\boldsymbol{x} \in \mathcal{M}$, the differential of Φ at $\boldsymbol{x}$ is a linear map, $\mathrm{d}\Phi_{\boldsymbol{x}} : T_{\boldsymbol{x}}\mathcal{M}, \to T_{\Phi(\boldsymbol{x})}\mathcal{N}$, from the tangent space at $\boldsymbol{x} \in \mathcal{M}$ to the tangent space at $\Phi(\boldsymbol{x}) \in \mathcal{N}$ (Figure 3.5). The differential, $\mathrm{d}\Phi_{\boldsymbol{x}}$, is used to map vectors between tangent spaces. The *pullback* operator is the linear operation $\mathrm{d}\Phi^*_{\boldsymbol{x}} : T_{\Phi(\boldsymbol{x})}\mathcal{N} \to T_{\boldsymbol{x}}\mathcal{M}$ that maps a vector from $T_{\Phi(\boldsymbol{x})}\mathcal{N}$ to $T_{\boldsymbol{x}}\mathcal{M}$.

[1] The precise term for $T\mathcal{M}$ is tangent bundle. The tangent bundle is the disjoint set of all tangent spaces. The tangent space is defined at a certain point $\boldsymbol{x}$, $T_{\boldsymbol{x}}\mathcal{M}$. For simplicity, with a slight terminology abuse, in this work, we use the term tangent space.

3.3.2. Problem Statement

We aim to solve the problem of modeling SVF on Lie Groups. In particular, we model our SVF by diffeomorphisms. Diffeomorphism-based SVF represent the vector field in the observation space as the deformed vector field of a certain latent space [159, 175, 242, 187]. These models assume there exist a **stable vector field in a latent space** $\boldsymbol{g} : \mathcal{N} \rightarrow T\mathcal{N}$. Then, given a **parameterized diffeomorphic mapping** Φ, that maps any point in observation space $\mathcal{M}$ to the latent space $\mathcal{N}$, $\Phi : \mathcal{M} \rightarrow \mathcal{N}$, we can represent the dynamics in the observation space

$$\dot{\boldsymbol{x}} = \mathrm{d}\Phi^*_{\boldsymbol{x}} \circ \boldsymbol{g} \circ \Phi(\boldsymbol{x}), \tag{3.19}$$

in terms of the latent dynamics $\boldsymbol{g}$ and the diffeomorphism Φ. $\mathrm{d}\Phi^*_{\boldsymbol{x}}$ is the **pullback operator** that maps a velocity vector from the latent space to the observation space. Intuitively, as shown in Figure 3.5, the diffeomorphic function Φ deforms the space changing the direction of the vector field in the observation space.

Previous *diffeomorphism-based* SVF are limited to Euclidean spaces, without representing motion policies in the orientation. Euclidean SVF assumes (i) that $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^n$ defines a bijective mapping between Euclidean spaces, (ii) in Euclidean spaces, the tangent space and the manifold are in the same space, and then, the latent dynamics are $\boldsymbol{g} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and, (iii) given Φ defines a mapping between Euclidean spaces, the pullback operator is represented by the Jacobian pseudoinverse of Φ, $\mathrm{d}\Phi^*_{\boldsymbol{x}} = \boldsymbol{J}^{\dagger}_{\Phi}$.

In our work, given we are required to model the SVF on Lie Groups, we need to (i) model a Φ function that is bijective between Lie Groups, (ii) investigate how to model stable latent dynamics for Lie Groups and (iii) investigate how to model the pullback operator given the diffeomorphism Φ.

3.3.3. Stable Vector Fields on Lie Groups

As introduced in Section 3.3.2, modelling *diffeomorphism-based* SVF on Lie Groups requires additional insights in the modelling of the three main elements Φ, $\boldsymbol{g}$ and $\mathrm{d}\Phi^*$. In the following, we introduce our proposed models to represent each of these elements and we add a control block diagram on Figure 3.7 to provide intuition on how to use the proposed SVF in practice.

Diffeomorphic Mapping Φ

We introduce our proposed function to learn diffeomorphisms between Lie Groups, $\Phi : \mathcal{M} \to \mathcal{N}$. Both $\mathcal{M}$ and $\mathcal{N}$ are manifolds for the same Lie group, with $\mathcal{M}$ representing the Lie group in the observation space and $\mathcal{N}$, the Lie group in the latent space. A simple example of Φ is given by the rotation function. Given $\boldsymbol{X} \in \mathcal{M} = SO(3)$ and $\boldsymbol{Z} \in \mathcal{N} = SO(3)$, the rotation function $\boldsymbol{Z} = \Phi(\boldsymbol{X}) = \boldsymbol{R}\boldsymbol{X}$, applies a linear diffeomorphic mapping between $\mathcal{M}$ and $\mathcal{N}$.

Nevertheless, representing nonlinear diffeomorphic mappings for Lie groups is challenging. In our work, we propose to exploit the tangent space to learn these mappings. In contrast with the manifold, the tangent space is a Euclidean space, making it easier to model nonlinear diffeomorphic functions.

The topology of the Lie groups and their Lie algebra are not the same. Then, it is impossible to define a single diffeomorphic function Φ that maps all the points in the group to the Lie algebra. To make proper use of the Lie algebra and still guarantee the diffeomorphism for the whole Lie group, we propose to model the diffeomorphism by parts. We visualize an example of the proposed function in Figure 3.6. The points in the Lie Group are split into two sets. We consider a coordinate chart $U_{\mathcal{M}} \subseteq \mathcal{M}$ that defines a set of almost all the points in the Lie group. Then, we group all the points not belonging to the set $U_{\mathcal{M}}$ in a different set, $\boldsymbol{x} \in \mathcal{M} \ominus U_{\mathcal{M}}$. For example, in the example on Figure 3.6, we group all the points except the antipodal point in $U_{\mathcal{M}}$ and put the antipodal point in the set $\mathcal{M} \ominus U_{\mathcal{M}}$. The points in the set $U_{\mathcal{M}}$ are mapped to a set in the latent manifold, $U_{\mathcal{N}} \subseteq \mathcal{N}$. The points in the set $\mathcal{M} \ominus U_{\mathcal{M}}$ are mapped to the latent space set $\mathcal{N} \ominus U_{\mathcal{N}}$. Given that $\mathcal{M}$ and $\mathcal{N}$ are represented in the same Lie Group, the sets in the observation space and the latent space are also the same.

$$\Phi(\boldsymbol{x}) = \begin{cases} \text{ExpMap} \circ \boldsymbol{f_\theta} \circ \text{LogMap}(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in U_{\mathcal{M}} \\ \boldsymbol{x} & \text{if } \boldsymbol{x} \in \mathcal{M} \ominus U_{\mathcal{M}}. \end{cases} \tag{3.20}$$

For any element in the coordinate chart $\boldsymbol{x} \in U_{\mathcal{M}}$, we define the map from $U_{\mathcal{M}}$ to $U_{\mathcal{N}}$, through the tangent space, Φ : ExpMap $\circ\, \boldsymbol{f_\theta} \circ$ LogMap. The function first maps a point in the Lie group to the Lie algebra by the LogMap. For any point $\boldsymbol{x} \in U_{\mathcal{M}}$, it will map to a point in a subset of the tangent space, $\hat{\boldsymbol{x}} \in \hat{U}_{\mathcal{M}} \subseteq T_{\boldsymbol{x}_H}\mathcal{M}$. We call **first cover of the tangent space** to $\hat{U}_{\mathcal{M}}$. The map between $U_{\mathcal{M}}$ and $\hat{U}_{\mathcal{M}}$ is guaranteed to be diffeomorphic given the LogMap properties [122]. Then, we apply a Euclidean diffeomorphism $\boldsymbol{f_\theta}$ between the first covers of the observation space $\hat{U}_{\mathcal{M}}$ and the first covers of the latent space, $\hat{U}_{\mathcal{N}}$. We introduce our proposed $\boldsymbol{f_\theta}$ in Section 3.3.4. Finally, we can map the points

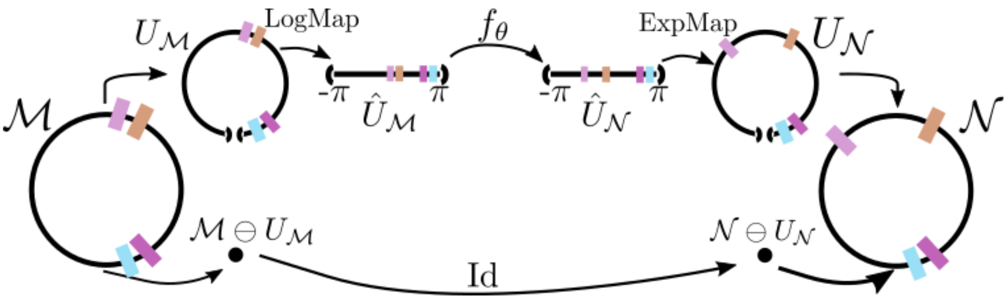


Figure 3.6.: A visual representation of the Φ function for 1-sphere ($\mathcal{S}^1$). The points in $\mathcal{S}^1$ are split into two groups. For the points in $U_{\mathcal{M}}$, the diffeomorphism is composed by first, mapping the points to the first-cover $\hat{U}_{\mathcal{M}}$ by the LogMap, then applying a bounded Euclidean diffeomorphism between $\hat{U}_{\mathcal{M}}$ and $\hat{U}_{\mathcal{N}}$ and mapping the points back to the manifold, by the ExpMap. For the points not belonging to $U_{\mathcal{M}}$, we simply apply the identity map. If $f_{\boldsymbol{\theta}}$ is the identity map close to the boundaries $-\pi$ and π; the map is diffeomorphic for the whole $\mathcal{S}^1$. We add a few markers to represent the space deformation along the mappings.

$\hat{\boldsymbol{z}} \in \hat{U}_{\mathcal{N}}$ back to $\boldsymbol{z} \in U_{\mathcal{N}} \subseteq \mathcal{N}$ by the ExpMap and represent it in the Lie Group. Given the three steps are diffeomorphic, we can guarantee that Φ applies a diffeomorphism between $U_{\mathcal{M}}$ and $U_{\mathcal{N}}$. For the points not belonging to the set $U_{\mathcal{M}}$, we apply the identity map. The identity map is also diffeomorphic.

Even if each part in Equation (3.20) is diffeomorphic in itself, to guarantee the function Φ is diffeomorphic in the whole Lie group, we require to guarantee the function is continuous and differentiable in the boundaries between $U_{\mathcal{M}}$ and $\mathcal{M} \ominus U_{\mathcal{M}}$. To do so, we impose structurally $\boldsymbol{f_\theta}$ to become the identity map $\boldsymbol{f_\theta}(\hat{\boldsymbol{x}}) = \hat{\boldsymbol{x}}$ when approaching to the boundaries of the set $\hat{U}_{\mathcal{M}}$. Thus,

$$\begin{aligned}\Phi(\boldsymbol{x}) &= \text{ExpMap} \circ \boldsymbol{f_\theta} \circ \text{LogMap}(\boldsymbol{x}) \\ &= \text{ExpMap} \circ \text{LogMap}(\boldsymbol{x}) = \boldsymbol{x}\end{aligned} \tag{3.21}$$

when $\boldsymbol{x}$ is close to the boundaries of $U_{\mathcal{M}}$.

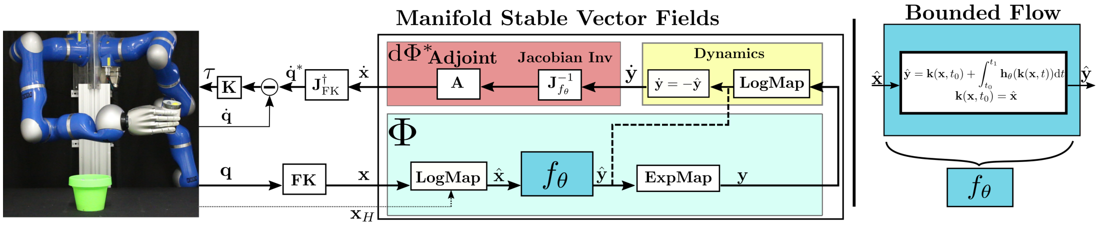

Manifold Stable Vector Fields
$d\Phi^*$ Adjoint
Jacobian Inv
Dynamics
A
$\mathbf{J}_{f_\theta}^{-1}$
$\dot{\mathbf{y}} = -\hat{\mathbf{y}}$
LogMap
$\dot{\mathbf{y}}$
$\dot{\mathbf{x}}$
$\mathbf{J}_{\text{FK}}^{\dagger}$
$\dot{\mathbf{q}}^*$
K
τ
$\dot{\mathbf{q}}$
Φ
q
FK
x
LogMap
$\hat{\mathbf{x}}$
f_θ
$\hat{\mathbf{y}}$
ExpMap
y
$\mathbf{x}_H$
Bounded Flow
$\hat{\mathbf{x}}$
$\hat{\mathbf{y}} = \mathbf{k}(\mathbf{x}, t_0) + \int_{t_0}^{t_1} \mathbf{h}_\theta(\mathbf{k}(\mathbf{x}, t))\mathrm{d}t$
$\mathbf{k}(\mathbf{x}, t_0) = \hat{\mathbf{x}}$
$\hat{\mathbf{y}}$
f_θ

Figure 3.7.: Left: Manifold stable vector fields block diagram. Right: Proposed architecture for our diffeomorphic function $\boldsymbol{f_\theta}$. As shown in Equation (3.19), our manifold SVF is composed of three elements: a diffeomorphism Φ (light blue box)(*for simplicity, we only visualize the part related with the set $U_{\mathcal{M}}$*), the latent dynamics g (yellow box) and, the pullback operator $\mathrm{d}\Phi^*$ (red box). The diffeomorphism Φ is composed of three elements: the LogMap, a bounded diffeomorphism between first covers $\boldsymbol{f_\theta}$ (blue) and, the ExpMap. The pullback operator $\mathrm{d}\Phi^*$ has two elements: the Jacobian inverse, computed for the diffeomorphism $\boldsymbol{f_\theta}$, and the Adjoint operator. Additionally, to control a robot, we first map the current joint configuration $\boldsymbol{q}$ to $\boldsymbol{x} \in SE(3)$ by Forward Kinematics. And once $\dot{\boldsymbol{x}} \in \mathfrak{se}(3)$ is computed, we map it back to the configuration space by $\boldsymbol{J}_{\text{FK}}^{\dagger}$. Then, we apply a velocity controller in the configuration space. The dashed line from the output of $\boldsymbol{f_\theta}$ and the dynamics input represents a shortcut we consider in practice as long as the latent ExpMap and LogMap are computed in the same origin frame.

An intuitive example for 1-sphere (S^1) manifold

The 1-sphere manifold is composed by all the points in a circle of radius r, $\mathcal{S}^1 : \{\boldsymbol{x} \in \mathbb{R}^2 ; ||\boldsymbol{x}|| = r\}$. We visualize this manifold in Figure 3.6. To model a diffeomorphic transformation between $\mathcal{M}$ and $\mathcal{N}$, we propose to split the manifold in two sets: the set $U_{\mathcal{M}}$ considers all the points in the manifold except the point in the south $\boldsymbol{x}_S = (0, r)$, $U_{\mathcal{M}} = \mathcal{S}^1_{\neq \boldsymbol{x}_S}$. Equally, the set in the latent manifold $U_{\mathcal{N}}$, also consider $U_{\mathcal{N}} = \mathcal{S}^1_{\neq \boldsymbol{x}_S}$. The other set $\mathcal{M} \ominus U_{\mathcal{M}} = \{\boldsymbol{x}_S\}$ is composed of the point not belonging to $U_{\mathcal{M}}$. We can observe that $U_{\mathcal{M}}$ is diffeomorphic to the open line segment $\hat{U}_{\mathcal{M}} = (-\pi, \pi)$. We refer to this set as first-cover of the tangent space $\hat{U}_{\mathcal{M}} = \hat{U}_{\mathcal{N}} = (-\pi, \pi)$. We can map any point from $U_{\mathcal{M}}$ to $\hat{U}_{\mathcal{M}}$ by the LogMap function. Inversely, we can map the points from the open line segment to the set $U_{\mathcal{M}}$ by the ExpMap function. We remark that points in $U_{\mathcal{M}}$ are two-dimensional while points in $\hat{U}_{\mathcal{M}}$ are one-dimensional. Once the points are in the $\hat{U}_{\mathcal{M}}$, we model a bounded diffeomorphic function $\boldsymbol{f_\theta}$ that maps the points in $\hat{U}_{\mathcal{M}}$ to $\hat{U}_{\mathcal{N}}$. We present in Section 3.3.4 how we model this bounded diffeomorphism $\boldsymbol{f_\theta}$. This map can be thought of as a deformation of the line $\hat{U}_{\mathcal{M}}$, stretching or contracting the line. We highlight that while representing directly a diffeomorphism between the open line segments $\hat{U}_{\mathcal{M}}$ and $\hat{U}_{\mathcal{N}}$ is easy, representing it between the groups $U_{\mathcal{M}}$ and $U_{\mathcal{N}}$ is hard, given that $U_{\mathcal{M}}$ and $U_{\mathcal{N}}$ are not Euclidean spaces.

As shown before, to guarantee that Φ is diffeomorphic for the whole manifold $\mathcal{S}^1$, we need to guarantee that $\boldsymbol{f_\theta}$ becomes the identity map close to the boundaries of $\hat{U}_{\mathcal{M}}$. For the case of $\mathcal{S}^1$, the function $\boldsymbol{f_\theta}$ should approximate the identity map the closer the points are to $-\pi$ and π. Intuitively, the function $\boldsymbol{f_\theta}$ represents a space deformation in $(-\pi, \pi)$ that becomes the identity close to the boundaries $-\pi$ or π. We illustrate this diffeomorphic map in Figure 3.6.

Latent Stable Dynamics g

For a given manifold $\mathcal{N}$, the vectors are represented in the tangent space of the manifold, $T\mathcal{N}$. Thus, a dynamic system in a manifold is a function that for any point in the manifold outputs a vector in the tangent space, $\boldsymbol{g} : \mathcal{N} \to T\mathcal{N}$. Similarly to the transformation map Φ, we propose to model the dynamics by parts

$$\dot{\boldsymbol{z}} = g(\boldsymbol{z}) = \begin{cases} -\mathrm{LogMap}_{\boldsymbol{z}_H}(\boldsymbol{z}) & \text{if } \boldsymbol{z} \in U_{\mathcal{N}} \\ \boldsymbol{0} & \text{if } \boldsymbol{z} \in \mathcal{N} \ominus U_{\mathcal{N}} \end{cases} . \tag{3.22}$$

For any element in $U_{\mathcal{N}}$, we first map the point to the tangent space centered at $\boldsymbol{z}_H$ and then, compute the velocity vector as $\dot{\boldsymbol{z}} = \boldsymbol{g}(\hat{\boldsymbol{z}}) = -\hat{\boldsymbol{z}}$. These dynamics will induce a stable dynamic system in the manifold $U_{\mathcal{N}}$, with a sink in $\boldsymbol{z}_H$. For any point out of the set $U_{\mathcal{N}}$, we set the velocity to zero. This will set an unstable equilibrium point for any point in $\mathcal{N} \ominus U_{\mathcal{N}}$. In practice, given the LogMap in our dynamics Equation (3.22) is the inverse of the ExpMap in Φ, we can directly compute the dynamics using as input the output of $f_{\boldsymbol{\theta}}$ without moving to $\mathcal{N}$ (dashed line in Figure 3.7).

Pullback Operator $\mathrm{d}\Phi^*$

The pullback operator unrolls all the steps to the latent space, $\mathcal{N}$, done by the diffeomorphism, Φ, back to the observation manifold, $\mathcal{M}$. Additionally, given the velocity vector is defined on the tangent space, the unrolling steps are done on the tangent space. The pullback operator for the mapping, $\boldsymbol{f_\theta}$, is the Jacobian $\boldsymbol{J_f}$. The inverse of the Jacobian, maps the velocity vector from the latent tangent space to the observation tangent space, centered in the origin, $\boldsymbol{J}_{\boldsymbol{f}}^{-1} : T_{\boldsymbol{z}_H}\mathcal{N} \to T_{\boldsymbol{x}_H}\mathcal{M}$. Additionally, we apply a second pullback operator to map the vector from the tangent space in the origin $\boldsymbol{x}_H$ to the tangent space in the current pose $\boldsymbol{x}$, $\boldsymbol{A} : T_{\boldsymbol{x}_H}\mathcal{M} \to T_{\boldsymbol{x}}\mathcal{M}$. This linear map is known as the adjoint map and it can be understood as a change of reference frame for the velocity vectors. We direct the reader to [223] to find more information on how to model it. The whole pullback operator is then, $\mathrm{d}\Phi^* = \boldsymbol{A} \circ \boldsymbol{J}_{\boldsymbol{f}}^{-1}$.

3.3.4. Bounded Flows as transformation $\boldsymbol{f}_{\boldsymbol{\theta}}$

In Section 3.3.3, we propose to model the diffeomorphism between two subsets of the manifolds ($U_{\mathcal{M}}$ and $U_{\mathcal{N}}$) through the tangent space. To properly model the diffeomorphism, we have introduced a function $\boldsymbol{f}_{\boldsymbol{\theta}}$ and defined its required properties. The function $\boldsymbol{f}_{\boldsymbol{\theta}}$ should be a diffeomorphism and should become identity when approximating the boundaries of the tangent space sets $\hat{U}_{\mathcal{M}}$ and $\hat{U}_{\mathcal{N}}$. To represent our function $\boldsymbol{f}_{\boldsymbol{\theta}}$, we build on top of the research on INN for Normalizing Flows [198, 27].

We propose to model the function $\boldsymbol{f}_{\boldsymbol{\theta}}$ by adapting Neural ODEs [27] to our problem. Neural ODEs propose to model the diffeomorphism between two spaces by the flow of a parameterized vector field $\boldsymbol{h}_{\boldsymbol{\theta}}$. The flow $\boldsymbol{k}(\boldsymbol{x}, t) : \mathbb{R}^{n+1} \to \mathbb{R}^n$, represents the motion of a point for the time t, given the ODE, $\mathrm{d}\boldsymbol{x}/\mathrm{d}t = \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x}) \equiv \mathrm{d}(\boldsymbol{k}(\boldsymbol{x}, t))/\mathrm{d}t = \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{k}(\boldsymbol{x}, t))$

$$\boldsymbol{x}_{t_1} = \boldsymbol{k}(\boldsymbol{x}, t_1) = \boldsymbol{x} + \int_0^{t_1} \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{k}(\boldsymbol{x}, t))\mathrm{d}t. \tag{3.23}$$

with t_1 being a certain time instant and $\boldsymbol{x}$ the position of the particle in the instant $t = 0$. The flow function represents the position of a particle $\boldsymbol{x}$ follows given the vector field $\boldsymbol{h}_{\boldsymbol{\theta}}$ at the instant t_1. In Neural ODEs, the function $\boldsymbol{f}_{\boldsymbol{\theta}}$ is represented by the output of the flow at time 1

$$\boldsymbol{z} = \boldsymbol{f}_{\theta}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x}, t = 1) = \boldsymbol{x} + \int_0^{1} \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{k}(\boldsymbol{x}, t))\mathrm{d}t. \tag{3.24}$$

As presented in [146, 27], the function is a diffeomorphism, as long as $\boldsymbol{h}_{\boldsymbol{\theta}}$ is a uniformly Lipschitz continuous vector field (Picard–Lindelöf theorem).

Additionally, to compute the pullback operation, we are required to compute the Jacobian matrix of $\boldsymbol{f}_{\boldsymbol{\theta}}$, $\boldsymbol{J}_f = \nabla_{\boldsymbol{x}}\boldsymbol{k}(\boldsymbol{x}, t_1)$. Given the vector field $\boldsymbol{h}_{\boldsymbol{\theta}}$, there exists an ODE representing the time evolution of the Jacobian

$$\begin{aligned}\dot{\boldsymbol{J}}_f(\boldsymbol{x},t) &= \nabla_{\boldsymbol{k}} \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{k}(\boldsymbol{x},t))\boldsymbol{J}_f(\boldsymbol{x},t) \\ \boldsymbol{J}(\boldsymbol{x},t_0) &= \boldsymbol{I}.\end{aligned} \tag{3.25}$$

In practice, we can use an arbitrary ODE solver and find the values for $\boldsymbol{J}(\boldsymbol{x},t_1)$ and $\boldsymbol{k}(\boldsymbol{x},t_1)$ solving Equation (3.24) and Equation (3.25). In our case, to guarantee a high control frequency rate, we apply the forward Euler method to solve the ODE and then compute the Jacobian by backward differentiation. It is important to remark that these dynamics are used to represent the diffeomorphism $\boldsymbol{f}_{\boldsymbol{\theta}}$ between two spaces and not to represent the desired vector fields.

Relevant consideration for our problem is that the function $\boldsymbol{f}_{\boldsymbol{\theta}}$ should define a diffeomorphism between two bounded sets $\hat{U}_{\mathcal{M}}$ and $\hat{U}_{\mathcal{N}}$ and the transformation should become identity close to the boundaries of these sets. Nevertheless, without any additional considerations on $\boldsymbol{h}_{\boldsymbol{\theta}}$, the flow could move a point in $\hat{U}_{\mathcal{M}}$ to any point in $\mathbb{R}^n$, with n the dimension of the Euclidean space in which the set $\hat{U}_{\mathcal{N}}$ is. To bound the flow between the sets, we impose structurally that the vector field $\boldsymbol{h}_{\boldsymbol{\theta}}$ vanishes when approaching the boundaries. If the flow dynamics are zero, then, the input and the output are the same and we don't apply space deformation at that point. Given a distance function $\alpha(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ that measures how close we are to the boundaries, we define the vector field as

$$\boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \alpha(\boldsymbol{x})\boldsymbol{\psi}_{\boldsymbol{\theta}}(\boldsymbol{x}), \tag{3.26}$$

with $\boldsymbol{\psi}$ an arbitrarily chosen uniformly Lipschitz continuous parameterized vector field and α the scaling function of the dynamics to satisfy the desired constraints, preventing to move out of the set. α becomes zero close to the boundaries. Then, close to the boundaries,

$$\boldsymbol{z} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x},t_1) \approx \boldsymbol{k}(\boldsymbol{x},t_0) = \boldsymbol{x}. \tag{3.27}$$

Thus, the function $\boldsymbol{f}_{\boldsymbol{\theta}}$ is guaranteed to approximate the identity in the boundaries.

Given the set $\hat{U}_{\mathcal{M}}$ varies between the manifolds, we consider different distance functions α for each possible manifold. For the case of $SO(2)$, the first covers are $\hat{U}_{\mathcal{M}} = \hat{U}_{\mathcal{N}} = (-\pi,\pi)$. To impose identity map in the boundaries, the dynamics are weighted with $\alpha(x) = (\pi - |x|)/\pi$. α is a function that moves from 1 to 0 when we approach the $\pm\pi$ boundaries.

For the case of $\mathcal{S}^2$, the sets are $\hat{U}_{\mathcal{M}} = \hat{U}_{\mathcal{N}} = \{\boldsymbol{x} \in \mathbb{R}^2; ||\boldsymbol{x}|| < \pi\}$. This set is diffeomorphic to the set in $U_{\mathcal{M}} = U_{\mathcal{N}} \subset \mathcal{S}^2$, which considers all the points in the manifold except the

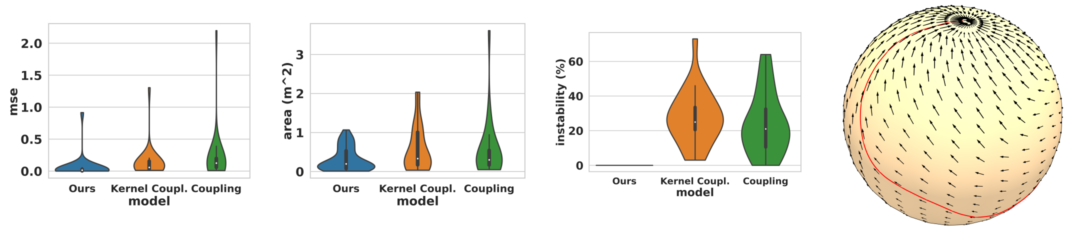


Figure 3.8.: Left: Kernel Coupling, Coupling, and Ours(Smooth Piecewise Linear) Layers compared in terms of Mean Squared Error (MSE), Area and Instability %. Kernel Coupling and Coupling Layer apply a diffeomorphism between $\mathbb{R}^n$ and Ours between the first covers. Right: Example of LASA trajectory and learned vector field.

antipodal point. To impose the dynamics to become zero close to the boundaries of the set, the distance function is $\alpha(\boldsymbol{x}) = (||\boldsymbol{x}|| - \pi)/\pi$.

For the case of $SO(3)$, the sets are $\hat{U}_{\mathcal{M}} = \hat{U}_{\mathcal{N}} = \{\boldsymbol{x} \in \mathbb{R}^3; ||\boldsymbol{x}|| < \pi\}$. The sets are diffeomorphic to the SO(3) sets $U_{\mathcal{M}} = U_{\mathcal{N}} = SO(3)_{\neq \boldsymbol{R}_\pi} \subset SO(3)$, that consider all possible rotation matrices except the ones that have a π rotation from the origin. The dynamics are weighted by the function $\alpha(\boldsymbol{x}) = (||\boldsymbol{x}|| - \pi)/\pi$.

For the case of the special Euclidean groups SE(2) and SE(3), the orientation-related dimensions maintain the same first covers of the special orthogonal groups. For the position-related dimensions, we bound the first cover to the desired workspace. Given $(\boldsymbol{p}, \boldsymbol{\theta}) \in \mathfrak{se}(3)$, with $\boldsymbol{p}$ the position related variables and $\boldsymbol{\theta}$, orientation related variables. We consider two scaling functions, one for orientations and one for positions. The orientation scaling function $\alpha_{\text{ori}}(\boldsymbol{\theta})$ is computed given the scaling functions above. The scaling function for the positions $\alpha_{\text{pos}}(\boldsymbol{p})$ can be used to enforce workspace limits and varies depending on the chosen workspace boundaries. We compute the distance function by $\alpha(\boldsymbol{p}, \boldsymbol{\theta}) = \alpha_{\text{pos}}(\boldsymbol{p})\alpha_{\text{ori}}(\boldsymbol{\theta})$.

3.4. Experimental Results

We present three experiments to evaluate the performance of our approach. In the first experiment, we illustrate, in a $\mathcal{S}^2$ manifold, the performance of our proposed $\boldsymbol{f_\theta}$ w.r.t. functions that do not take into consideration the manifold and treat is as Euclidean. Even if S^2 is not a Lie Group, we can apply the proposed approach also on it and serves as a useful manifold for illustration.

In the second and third experiments, we evaluate the performance of our model in the Lie Groups SE(2) and SE(3), for a 2D peg-in-a-hole task and a pouring task respectively.

3.4.1. Network Evaluation in S^2 manifold

We study the problem of learning stable vector fields in 2-sphere, S^2 by behavioral cloning (Algorithm 5). The objective of this experiment is to evaluate the influence of choosing different INN as mapping $\boldsymbol{f_\theta}$.

For evaluation, we consider three models. The three models use our proposed architecture in Figure 3.7 and vary in the used diffeomorphism $\boldsymbol{f_\theta}$. We consider two models using the INN from previous works [187, 242] that considers a diffeomorphism in the whole Euclidean space $\boldsymbol{f_\theta} : \mathbb{R}^n \to \mathbb{R}^n$ and our proposed INN that learns a diffeomorphism in bounded domains, $\boldsymbol{f_\theta} : \hat{U}_{\mathcal{M}} \to \hat{U}_{\mathcal{N}}$. We modified the LASA dataset [103] to $\mathcal{S}^2$ manifolds. We consider 22 different shape trajectories and evaluate the models given three metrics: MSE, Area, and Instability percentage. For measuring the instability percentage, we initialized a set of points in random positions on S^2 and generated a trajectory with the learned vector fields. Then, we measured how many trajectories reach the target position after a certain period.

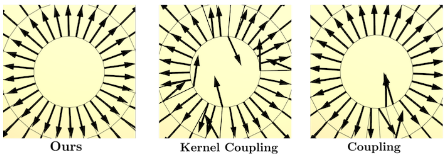


Figure 3.9.: Vector fields in the antipodal point of the Sphere. Our proposed diffeomorphism guarantees a source in the antipodal, while the unbounded INN does not.

From Figure 3.8, we can observe that the three architectures performed similarly in both MSE and Area measures and were able to mimic the performance of the demonstrations properly. This indicates that the proposed algorithm can learn vector fields on smooth manifolds. Nevertheless, as shown in the Instability % metric, the performance of the Kernel Coupling Layer [187] and the Coupling Layer [242] decay when initializing the trajectories in a random position. Given the Kernel Coupling Layer and the Coupling Layer define a diffeomorphism in the whole Euclidean space, they lack any guarantee of being bijective between $\hat{U}_{\mathcal{M}}$ and $\hat{U}_{\mathcal{N}}$. Thus, these approaches lack guarantees about the stability of the vector field in $\hat{U}_{\mathcal{M}}$. We can observe the instability of the vector fields by observing the antipodal point of the sphere, where the boundaries of the first cover $\hat{U}_{\mathcal{M}}$ are defined. As shown in Figure 3.9, while our INN can guarantee all the vectors pointing out of the antipodal (a source in the antipodal point), the kernel coupling layer and coupling layer are not able to guarantee stability close to the boundaries generating oscillatory behaviors around the antipodal point.

3.4.2. Evaluation of $SE(2)$ Stable vector fields in a 2D peg-in-a-hole task

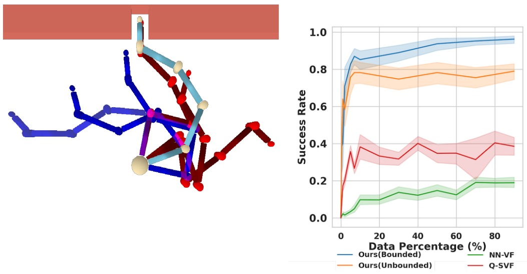


Figure 3.10.: Left: Peg-in-a-hole environment. We show in different colors, generated trajectories from different initial configurations. Right: Success rate Vs. Data percentage. We evaluate the performance of a set of models when trained with different amounts of data.

We consider the environment presented in Figure 3.10. The robot is a 5-DOF robot moving in a 2d plane. The goal of the task is to move the end-effector of the robot into the hole while avoiding collisions against the walls. We generated a 1K trajectory demonstration to train our models by applying RRT-Connect [111] on the environment. We compare the performance of our model w.r.t. three baselines. First, we consider a vector field modeled by a naive fully connected neural network in the tangent space of $SE(2)$. Second, we trained a stable vector field in the configuration space, $\mathcal{Q}$. Third, similarly to the experiment in $\mathcal{S}^2$, we model a vector field with the architecture in Figure 3.7, but consider a vanilla INN as $\boldsymbol{f}_{\boldsymbol{\theta}}$ instead of the proposed INN. To evaluate the performances, we initialize the robot in a random configuration and reactively evolve the dynamics. To control the robot, we apply operational space control [104]. Given the current end-effector pose, $\boldsymbol{x} \in SE(2)$, we compute the desired velocity at the end effector $\dot{\boldsymbol{x}} \in \mathbb{R}^3$ and pullback to the configuration space by the Jacobian pseudoinverse.

We present the results in Figure 3.10. We measure the success of the different methods to approach the goal without colliding under different amounts of training data. The vanilla neural network model performed the worst with any amount of trained data. A vanilla-NN is not limiting the family of possible vector fields, thus it may learn vector

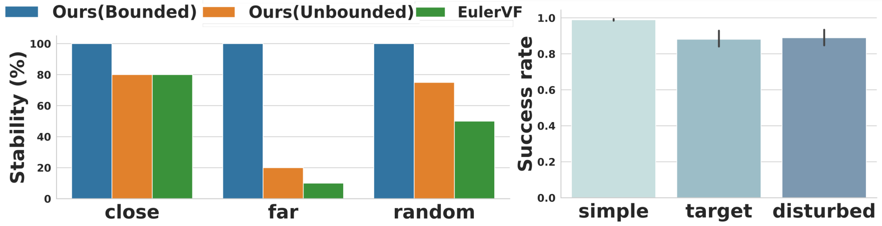


Figure 3.11.: Results for the pouring experiment. Right: simulated experiment results. We compare the stability property of the three models given three possible types of initial configurations (close to the target, far from the target, and random configuration). Left: real robot experiments results.

fields with multiple equilibrium points, limit cycles, or even unstable ones. This results in highly unstable vector fields with poor performance. The results also show the relevance of choosing a good task space representation. Learning in $SE(2)$ outperforms the configuration space approach. The difference in performance might be related to the vector field dimensionality, 5 for the configuration space and 3 for $SE(2)$ and also, with the task itself: as the peg-in-a-hole task is defined in the operational space the $SE(2)$ vector fields fit better the problem. Finally, we observe the benefit of our proposed INN w.r.t. vanilla INN approach. Given that the vanilla INN lacks global stability guarantees, the robot gets stuck in limit cycles and the performance decays.

In conclusion, we have observed that (i) stability guarantees greatly improves the performance of the policy for behavioral cloning problems (ii) representing the vector field in a proper manifold can boost the performance, and (iii) a bounded INN guarantees stability, while the unbounded one does not, given Φ is not diffeomorphic anymore.

3.4.3. Learning a pouring task with $SE(3)$ stable vector fields

In this experiment, we evaluate the performance of our method on a pouring task (Fig. 1). To properly pour, the robot requires to combine multiple positions and orientation changes. First, we compare in simulation our method with Euler angle-based vector fields. We consider two version of our model: One with bounded $\boldsymbol{f_\theta}$, introduced in Section 3.3.4 and one with a vanilla unbounded INN as $\boldsymbol{f_\theta}$ [38]. Then, we evaluate the performance of our model in a real robot under target modifications and human disturbances.

For this experiment, we use a 7 DoF Kuka LWR arm. The provided task demonstrations consist of 30 kinesthetic teaching trajectories with a wide variety of initial configurations. We considered different end-effector positions and orientations and trained the three models by behavioral cloning (Algorithm 5). To control the robot, we apply operational space control [104] for our proposed model (Figure 3.7) and position control for the Euler angles vector field. Note that our proposed method adapts to any other type of robot (prismatic joints, parallel robot) by changing the forward kinematics function. We evaluate the three models in three scenarios, robot performance with an initial configuration close to the target, initial configuration far from the target, and random initial configuration. We consider 10 different initial configuration and measure the robot's performance. In the three cases, we measured the stability guarantees of the models (i.e. the guarantee of arriving at the target pose after a certain time). We present the experiment results in Figure 3.11. From this figure, we can see that our model with the bounded function $f_{\boldsymbol{\theta}}$ outperformed the other models in the three cases. These results validate our claims on the requirements of defining a function $f_{\boldsymbol{\theta}}$ between the first covers, to guarantee stability in the whole Lie Group. Euler angle-based vector fields perform quite well for the case of close initial configuration. Euler Angles are an undesirable representation for feedback control due to their singularities and non-uniqueness. Nevertheless, we can assume these types of situations are rare close to the target and can perform relatively well. Nevertheless, their performance decay considering initial configurations far from the target. Given the non-uniqueness of the Euler-angles, representing globally stable vector fields in Euler-angles is not possible. In the case of our model with vanilla INN, it shows unstable behavior far from the target, while it remains quite stable close to it. Diffeomorphism-based SVF lack stability guarantees if the function Φ is not bijective. This lack of bijectiveness is more prone to happen close to the boundaries of the first cover and Φ remains bijective close to the target, with the guarantee of being stable.

We also evaluate the performance of our model on a real robot, measuring the model's performance under target modifications and human disturbances. To adapt to different target positions, we use the current one $\boldsymbol{x}_{\text{target}} \in SE(3)$ as the origin of the LogMap (Figure 3.7). This allows us to represent the vector fields relative to the current target position. We track the target pot by Optitrack motion capture systems. The control signal is computed in a close-loop at a rate of 100Hz.

For the system evaluation, we predefined 10 different initial configurations covering the whole workspace. The robot holds a glass with 4 balls and we measured the number of balls that enter the pot after executing the trajectory. We considered 3 scenarios: normal execution, physical disturbance, and target modification.

Looking at the results in Figure 3.11, it is clear that the robot achieves a very robust performance. In the normal execution, it pours almost all the balls in the pot, given any initial configuration. This result shows the generalization properties of our model: the robot was initialized in a position that does not belong to the demonstration set, but was able to solve the task. We also tested the system under heavy physical disturbances, including pushing and holding the robot. In this scenario, the performance decays, but the robot was able to succeed most of the time. Finally, we observe the vector field was able to properly adapt to different pot positions. The robot succeeded to put almost all the balls in the pot except for some target positions that were beyond the workspace limits of the robot.

3.5. Related Work

Stable Vector Fields SVF models are powerful motion generators in robotics given they are robust to perturbations and generalize the motion generation beyond the demonstrated trajectories. After the seminal work by Khansari et al. [103], several works [158, 175, 238, 242, 187] have proposed novel SVF models covering a wider family of solutions. Our work is particularly close to diffeomorphism based SVF models [158, 175, 242, 187].

Invertible Neural Networks (INN) in Smooth Manifolds INN are a family of neural networks that guarantee to represent bijective functions. The study of modeling INN for smooth manifolds has been mainly developed for density estimation. A set of previous works [199, 146, 61] have proposed INN for specific manifolds, such as Tori or Sphere manifolds. A more recent work [134] proposes a manifold agnostic approach, on which Neural ODE [27, 69] are adapted to manifolds. In [48], INN are proposed for Lie Groups. Similar to our work, they also exploit the Lie algebra to learn expressive diffeomorphisms, but the proposed model is limited to density estimation.

3.6. Discussion & Conclusions

In this chapter we have introduced a novel Motion Primitive that can learn stable vector fields exploiting the expresiveness of Normalizing Flows. We also extended the work to model stable vector fields on Lie Groups. The proposed model allows us to generate reactive and stable robot motions for the full pose (orientation and position). Through an extensive evaluation phase, we have validated the modeling decisions to guarantee stability and the importance of representing the vector fields on Lie Groups to properly solve robot tasks.

We have many directions to improve our model. First, the chosen diffeomorphic function Φ has some limitations. Our proposed model cannot set the sink in the antipodal points, given the map in antipodal points is an identity map. In practice, we can set the attractor in an arbitrary pose by adding a linear transformation that moves the sink. Nevertheless, we consider that this limitation might influence the performance when modeling complex motion skills with significant changes in orientation. In the future, we aim to explore novel functions to represent the diffeomorphism Φ. The experiments we have carried out focus on the performance evaluation of our proposed stable vector fields. However, these models are of particular interest combined with additional motion skills, such as obstacle avoidance or joint limit avoidance vector fields, as done in RMP [191] or CEP [240]. We will investigate how to combine vector fields in future works.

Another possibility is to use the proposed method as a cost function. Indeed, the architecture encodes in itself a Lyapunov-stable potential function. We can use this function as a terminal cost function (value function) or as a cost function in trajectory optimization problems, allowing the integration of additional cost functions. This approach could be beneficial in long-horizon planning problems [243].

4. Composability and Geometry on Energy-Based Policies

In this chapter, we introduce Composable Energy Policies (CEP) , a novel framework for multi-objective motion generation. We frame the problem of composing multiple policy components from a probabilistic view. We consider a set of stochastic policies represented in arbitrary task spaces, where each policy represents a distribution of the actions to solve a particular task. Then, we aim to find the action in the configuration space that optimally satisfies all the policy components. The presented framework allows the fusion of motion generators from different sources: optimal control, data-driven policies, motion planning, handcrafted policies. Classically, the problem of multi-objective motion generation is solved by the composition of a set of deterministic policies, rather than stochastic policies. However, there are common situations where different policy components have conflicting behaviors, leading to oscillations or the robot getting stuck in an undesirable state. While our approach is not directly able to solve the conflicting policies problem, we claim that modeling each policy as a stochastic policy allows more expressive representations for each component in contrast with the classical reactive motion generation approaches. In some tasks, such as reaching a target in a cluttered environment, we show experimentally that CEP's additional expressivity allows us to model policies that reduce these conflicting behaviors.

A field that benefits from these reactive motion generators is the one of robot reinforcement learning. Integrating these policy architectures with reinforcement learning allows us to include a set of inductive biases in the learning problem. These inductive biases guide the reinforcement learning agent towards informative regions or improve collision safety while exploring. In our work, we show how to integrate our proposed reactive motion generator as a structured policy for reinforcement learning. Combining the reinforcement learning agent exploration with the prior-based CEP, we can improve the learning performance and explore safer.

4.1. Introduction

Many robotic tasks deal with finding control actions satisfying multiple objectives. A seemingly simple task such as watering plants requires satisfying multiple objectives to perform it properly. The robot should reach the targets (the plants) with the watering can, avoid pouring water on the floor while approaching, and avoid colliding with and breaking the plant's branches by its arms. In contrast with sequential tasks [229, 94, 95], in which the objectives to be satisfied are concatenated in time, in this work we consider tasks in which multiple geometric objectives must be satisfied in parallel.

The problem has been faced with a spectrum of solutions that balance between global optimality and computational complexity. Path planning methods [119, 118, 100] find a global trajectory from start to goal by a computationally intense Monte-Carlo sampling process. Trajectory optimization methods [234, 188, 97, 212, 154] reduce the computational burden of planning methods by searching the global trajectory given initial trajectory candidates. These methods reshape the global trajectory to satisfy the objectives. However, they still require solving an optimization problem over long temporal horizon trajectories. The computational requirements of these algorithms limit the possibility of exploiting them for reactive motion generation. Computationally lighter, MPC methods [151, 179, 255, 13] consider the problem of solving a short-horizon trajectory optimization problem reactively. Rather than assuming the problem of solving the trajectory optimization problem for the whole trajectory, these methods consider the problem of solving the trajectory optimization problem in a receding horizon reducing the computational requirements.

Artificial Potential Fields (APF) methods [105, 60] and more recently RMP methods [191, 99, 28, 21, 215, 3] are one of the most popular approaches for reactive motion generation in manipulators. In contrast with path planning or trajectory optimization methods, these methods propose to solve a myopic (one-step ahead) control problem. Given the problem is local, the computational cost is very low and they can be used with high control frequencies. These methods propose to solve the multi-objective control problem by the composition of a set of deterministic actions. Each action is computed to satisfy a particular objective. Then, the optimal composing action is found by solving a least-squares optimization problem given all the action components. The solution of the optimization problem can be analytically represented by a weighted sum of the action components. The sum of the actions defines a trade-off between the components with the weighting term (the metric) giving the relevance of each component. In these methods, it is common to have conflicting action components that make the robot get stuck.

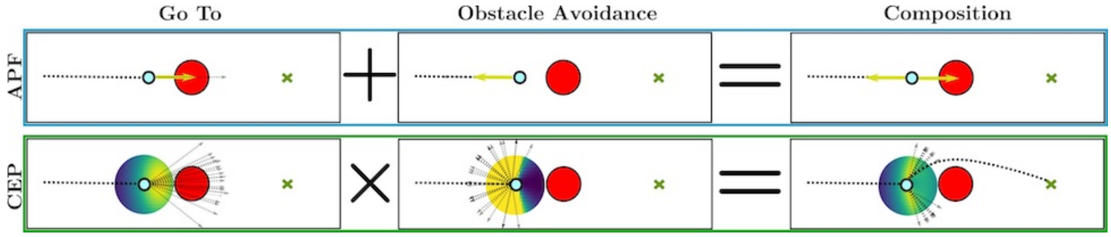


Figure 4.1.: Visual Representation of modular control for Goto + Obstacle Avoidance. In the top box, we show Artificial Potential Fields (APF) [105]. In the bottom box, we show Composable Energy Policies (CEP). In contrast, with the APF method, which sums deterministic actions (goto, avoid an obstacle), CEP computes the product of the policy distributions and then finds the maxima of the composition. The composition will provide a high probability to those actions that satisfy both components and low to the rest. *Robot: blue circle, obstacle: red circle, and target: green cross. Thick dotted line: performed trajectory, lightly dotted line: possible future trajectories.*[1]

Similarly to RMP and APF, we consider the problem of solving a one-step-ahead control problem. Considering a short-horizon problem reduces the variables to optimize and thus, we can guarantee sufficiently high control frequencies to be reactive for a high dimensional robot. In contrast with RMP and APF methods that assume deterministic policies to model the optimal behavior for each objective, we propose combining stochastic policies π_k. We hypothesize that considering arbitrary stochastic policies increases the expressivity on how to model the policies and we might represent policies that reduce the conflicting solutions in the composition. To find the composed action, we propose an optimization problem defined as a maximization over the log of the product of a set of stochastic policies [166, 72]

$$\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} \log \left(\prod_k \pi_k(\boldsymbol{a}|\boldsymbol{s}) \right), \tag{4.1}$$

with action $\boldsymbol{a}$ and state $\boldsymbol{s}$. One can view the product of policies as a probabilistic instance of a logical conjunction (AND operator) [40, 231] between the action distributions (see Figure 4.1 for visual representation). The product of policies will set a high probability to the actions that are likely to sample in all the components and low to the rest. If we would like to sample an action from the product of policies, we might apply Markov Chain Monte Carlo (MCMC) sampling process over the product of policies. In contrast, if we aim to obtain the action that satisfies best the composition, we can compute the maximum likelihood action Equation (4.1).

Beyond local reactive navigation, policy composition has become a relevant approach to integrate inductive biases in robot RL [218, 174, 93, 127]. These policy architectures allow the integration of reinforcement learning agents with prior knowledge. Rather than directly sampling the action commands for the robot, the reinforcement learning agent explores the parameter space of a structured policy. This structured model allows exploring safer or biasing the exploration towards informative regions. Nevertheless, prior methods assume explicit functions [218, 93, 127] to represent the structured policies. In contrast with previous methods, in our work, we propose to consider the optimization problem in Equation (4.1) as the structured policy. We show empirically, that considering an implicit function to represent the structured policy allows us to explore with fewer collisions with respect to previous structured policies.

Notation As our discussion will involve a set of policies and a set of spaces in which these policies are represented, we will use superscript (π^k) to represent the space in which the policy is and subscript (π_k) to represent the policy index. We represent the state by ($\boldsymbol{s}$) and the action by ($\boldsymbol{a}$). The space ($\mathcal{Q}$) represents the state-action space in the configuration $(\boldsymbol{s}^q, \boldsymbol{a}^q) \in \mathcal{Q}$ with $\boldsymbol{s}^q$ and $\boldsymbol{a}^q$ being, respectively, the state and action in the configuration space. ($\mathcal{X}_k$) represents the state-action space in the k task space $(\boldsymbol{s}^{x_k}, \boldsymbol{a}^{x_k}) \in \mathcal{X}_k$. $\boldsymbol{f}_q^x : \mathcal{Q} \to \mathcal{X}_k$ represents a transformation map from space $\mathcal{Q}$ to space $\mathcal{X}_k$. This map moves the state-action pairs from the configuration space to a task space.

4.1.1. Overview of APF and RMP

In reactive motion generation, we deal with the problem of generating the robot's motion online. The developed methods are required to have a low computational cost so that the robot responds fast to unexpected situations. Additionally, the generated motion should be able to deal with multiple tasks concurrently and represented in arbitrary spaces, such as avoiding collisions with multiple robot links, reaching a target with the end-effector, or avoiding joint limits. In APF and RMP, the optimal action is proposed to be computed by a weighted sum of the accelerations (RMP) or torques (APF) components solving

[1] Artificial Potential Fields can be framed as Composable Energy Policies, with each energy component represented by a quadratic function. For visualization purposes, we choose the classical representation of the sum of deterministic actions.

each particular task. In RMP, each acceleration component is the output of a task space second-order dynamic system.

Let us assume a set of transformation maps $\boldsymbol{f}_q^{x_0}, \dots, \boldsymbol{f}_q^{x_K}$, mapping the position, velocity and acceleration in the configuration space $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) \in \mathcal{Q}$, to a set of task spaces $(\boldsymbol{x}_k, \dot{\boldsymbol{x}}_k, \ddot{\boldsymbol{x}}_k) \in \mathcal{X}_k$; with $\boldsymbol{f}_q^{x_k} : \mathcal{Q} \to \mathcal{X}_k$ represented by

$$\begin{aligned}
\boldsymbol{x}_k &= \phi_q^{x_k}(\boldsymbol{q}) \\
\dot{\boldsymbol{x}}_k &= \frac{\mathrm{d}}{\mathrm{d}t}\phi_q^{x_k}(\boldsymbol{q}) = \boldsymbol{J}^{x_k}(\boldsymbol{q})\dot{\boldsymbol{q}} \\
\ddot{\boldsymbol{x}}_k &= \frac{\mathrm{d}^2}{\mathrm{d}t^2}\phi_q^{x_k}(\boldsymbol{q}) = \boldsymbol{J}^{x_k}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}^{x_k}(\boldsymbol{q})\dot{\boldsymbol{q}} \approx \boldsymbol{J}^{x_k}(\boldsymbol{q})\ddot{\boldsymbol{q}},
\end{aligned} \tag{4.2}$$

with $\boldsymbol{J}^{x_k} = \partial\phi_q^{x_k}(\boldsymbol{q})/\partial\boldsymbol{q}$ the Jacobian of the forward kinematic function $\phi_q^{x_k}$. The transformation for the acceleration is usually approximated dropping out the curvature term $\dot{\boldsymbol{J}}^{x_k}(\boldsymbol{q})\dot{\boldsymbol{q}}$. Given the integration steps in control loops (running between 100 Hz and 1 kHz) are small it is a valid approximation [191].

Let us also consider a set of task space second-order dynamics systems $\ddot{\boldsymbol{x}}_k = \boldsymbol{g}^{x_k}(\boldsymbol{x}_k, \dot{\boldsymbol{x}}_k)$, with a metric $\boldsymbol{\Lambda}^{x_k}$ associated to them. APF and RMP methods deviates in how the metric $\boldsymbol{\Lambda}^{x_k}$ is represented and applied to weight the components. In APF, the metric is conditioned on the position, while in RMP, the metric is conditioned on both position and velocity. Given the kinematics model in Equation (4.2), in [191], the dynamics and the metric in the configuration space $\mathcal{Q}$ are represented by

$$\begin{aligned}
\boldsymbol{g}_k^q(\boldsymbol{q}, \dot{\boldsymbol{q}}) &= \boldsymbol{J}^{x_k\dagger}\boldsymbol{g}^{x_k}(\phi_q^{x_k}(\boldsymbol{q}), \boldsymbol{J}^{x_k}(\boldsymbol{q})\dot{\boldsymbol{q}}) \\
\boldsymbol{\Lambda}_k^q &= \boldsymbol{J}^{x_k\intercal}\boldsymbol{\Lambda}^{x_k}\boldsymbol{J}^{x_k},
\end{aligned} \tag{4.3}$$

with $\boldsymbol{J}^{x_k\dagger}$ being the Jacobian pseudoinverse.

Finally, the acceleration in the configuration space is computed by a weighted-sum of all the dynamics systems represented in the configuration space

$$\ddot{\boldsymbol{q}} = \left(\sum_j \boldsymbol{\Lambda}_j^q\right)^{-1} \sum_k \boldsymbol{\Lambda}_k^q \boldsymbol{g}_k^q. \tag{4.4}$$

Instead, APF [105] methods do not compute the metric normalization

$$\boldsymbol{\tau} = \textstyle\sum_k \boldsymbol{\Lambda}_k^q \boldsymbol{g}_k^q. \tag{4.5}$$

The solution in Equation (4.4) is proven to be the optimal action for a least-squares optimization problem [191]

$$\ddot{\boldsymbol{q}}^* = \arg\min_{\ddot{\boldsymbol{q}}} \sum_{k} \frac{1}{2} ||\ddot{\boldsymbol{q}} - \boldsymbol{g}^q_k||^2_{\boldsymbol{\Lambda}^q_k}. \tag{4.6}$$

Each dynamic component represents the policy to satisfy a particular objective, while the metric weights the influence of each component in the composed action.

4.2. Composable Energy Policies

We will first motivate our approach and, subsequently, we introduce the different elements our policy architecture is composed of.

4.2.1. Motivation

CEP aims to provide a novel framework for multi-objective reactive motion generation. Our proposed method should be able to compute high frequency (100Hz-1kHz) control actions to apply in the robot. The computed action should be able to jointly satisfy multiple objectives. Additionally, we aim to model each component by an arbitrary stochastic policy.

The key idea of our proposed model is that in contrast to APF and RMP methods, we rather consider arbitrary stochastic policies to model each component. This leads to an optimisation problem where the cost of each component is no longer a quadratic function as in Equation (4.6). We expect that, given that we can model each component arbitrarily, our method will be able to more easily find an action that satisfies all the objectives if the policies are chosen correctly. We visualize this intuition in Figure 4.1.

4.2.2. Problem statement

Let us consider a set of stochastic policies, $\pi_k^{x_k}(\boldsymbol{a}^{x_k}|\boldsymbol{s}^{x_k})$, where each policy represents the optimal distribution in the action space to satisfy a particular objective. Each policy is represented in an arbitrary state-action space $\mathcal{X}_k$. Let us also consider a set of transformation maps $\boldsymbol{f}_{\boldsymbol{q}}^{\boldsymbol{x}_k}$

$$\begin{aligned} \boldsymbol{s}^{x_k} &= \boldsymbol{f}_{\boldsymbol{q}s}^{\boldsymbol{x}_k}(\boldsymbol{s}^q) \\ \boldsymbol{a}^{x_k} &= \boldsymbol{f}_{\boldsymbol{q}a}^{\boldsymbol{x}_k}(\boldsymbol{s}^q, \boldsymbol{a}^q), \end{aligned} \tag{4.7}$$

that relates the configuration space $\mathcal{Q}$, with the task spaces $\mathcal{X}_k$, in which each policy is defined. We aim to find the action in the configuration space, $\boldsymbol{a}^q$ that better satisfies all the policies. We frame the multi-objective reactive motion generation problem as an optimization problem defined by the policies and the task maps

$$\begin{aligned} &\boldsymbol{a}^{q*} = \arg\max_{\boldsymbol{a}^q} \log \left(\prod\nolimits_k \pi^{x_k}(\boldsymbol{a}^{x_k}|\boldsymbol{s}^{x_k}) \right) \\ &\text{s.t.} \quad (\boldsymbol{a}^{x_k}, \boldsymbol{s}^{x_k}) = \boldsymbol{f}_q^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q) \quad \forall k, \end{aligned} \tag{4.8}$$

with $\boldsymbol{f}_q^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q) \equiv (\boldsymbol{f}_{\boldsymbol{q}a}^{\boldsymbol{x}_k}(\boldsymbol{s}^q, \boldsymbol{a}^q), \boldsymbol{f}_{\boldsymbol{q}s}^{\boldsymbol{x}_k}(\boldsymbol{s}^q))$ and $\boldsymbol{s}^q$ the current state in the configuration space. We assume $\boldsymbol{s}^q$ is given. The optimization in Equation (4.8) represents our proposed reactive motion generation. Thus, we aim to solve this optimization in low computational time to guarantee high frequency control commands.

4.2.3. Composable energy policies method

Let us assume a set of independent stochastic policies $\pi_1(\boldsymbol{a}|\boldsymbol{s}), \dots, \pi_K(\boldsymbol{a}|\boldsymbol{s})$ modeled by a Gibbs distribution

$$\pi_i(\boldsymbol{a}|\boldsymbol{s}) = \frac{\exp(E_i(\boldsymbol{a}, \boldsymbol{s}))}{Z_i(\boldsymbol{s})}, \tag{4.9}$$

where $E : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is an arbitrarily represented energy function and $Z(\boldsymbol{s}) = \int_{\boldsymbol{a}} \exp(E(\boldsymbol{a}, \boldsymbol{s})) d\boldsymbol{a}$ is the normalization factor. The choice of Gibbs distribution is not arbitrary. Gibbs distribution allows representing an arbitrary distribution by a suitable definition of the energy function E [64]. Additionally, computing the product of experts

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \prod_k \pi_k(\boldsymbol{a}|\boldsymbol{s}) \propto \exp\left(\sum_k E_k(\boldsymbol{a}, \boldsymbol{s}) \right), \tag{4.10}$$

will end up in a weighted sum over the individual energy components in the exponential. Having the energy components linearly related is computationally beneficial. Given a set of energy components, in practice, we can parallelize the computation of all the components by multi-processing increasing the control frequency. Even if modeling the policy as a product of experts might seem an arbitrary choice, we show in Section 4.3 that, given a set of energy policies $\pi_1, \dots, \pi_K$, the product of experts represents the distribution of the optimal behavior to satisfy all the policy components [91].

Energy trees

Inspired by APF [104] and RMP [191], we propose to model the composition of energies in different task spaces. In the composition proposed in Equation (4.10), each energy function is considered to be in the same state-action space. However, in most of the robotics scenarios, we might be interested in composing together energies defined in different task spaces. Reaching a target while avoiding the obstacles, composes skills defined in different task spaces. To solve the task, we might require to define an attractor policy in the end-effector space of the robot and additional obstacle avoidance policies in different cartesian points in the links of the robot.

Our architecture is composed of two main components. First, we have a set of policies $\pi^{x_k}(\boldsymbol{a}^{x_k}|\boldsymbol{s}^{x_k})$, defined in different state-action task spaces $(\boldsymbol{s}^{x_k}, \boldsymbol{a}^{x_k}) \in \mathcal{X}_k$. Second, we consider a set of deterministic mappings that transform the state-action pairs in the configuration space $(\boldsymbol{s}^q, \boldsymbol{a}^q) \in \mathcal{Q}$ to the state-action task spaces $\mathcal{X}_k$, $\boldsymbol{f}_q^{x_k} : \mathcal{Q} \to \mathcal{X}_k$.

Consider the optimisation problem from Equation (4.8). For a single policy component. The problem is written as

$$\begin{aligned} \boldsymbol{a}^{q^*} &= \arg\max_{\boldsymbol{a}^q} \log \pi^{x_k}(\boldsymbol{a}^{x_k}|\boldsymbol{s}^{x_k}) \\ \text{s.t.} \quad \boldsymbol{a}^{x_k} &= \boldsymbol{f}_{q^a}^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q) \qquad (4.11) \\ \boldsymbol{s}^{x_k} &= \boldsymbol{f}_{q^s}^{x_k}(\boldsymbol{s}^q). \qquad (4.12) \end{aligned}$$

The unconstrained representation of the optimization problem in Equation (4.11) is

$$\boldsymbol{a}^{q^*} = \arg\max_{\boldsymbol{a}^q} \log \pi^{x_k}(\boldsymbol{f}_{q^a}^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q)|\boldsymbol{f}_{q^s}^{x_k}(\boldsymbol{s}^q)). \tag{4.13}$$

Moreover, given we are considering Gibbs distributions to represent each policy component, we can represent the optimization problem by

$$\boldsymbol{a}^{q^*} = \arg\max_{\boldsymbol{a}^q} E^{x_k}(\boldsymbol{f}_q^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q)) - \log Z^{x_k}(\boldsymbol{s}^{x_k}). \tag{4.14}$$

The objective function is represented in terms of the energy function E^{x_k} and the log of the normalization function Z^{x_k}. We can follow similar derivation for the multi-objective problem. The objective function $\mathcal{J}$ for the unconstrained problem of Equation (4.8) is represented as

$$\begin{aligned}\mathcal{J}(\boldsymbol{a}^q) &= \log \prod\nolimits_k \pi^{x_k}(\boldsymbol{f}_q^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q)) \\ &= \sum\nolimits_k \log \pi^{x_k}(\boldsymbol{f}_q^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q)) \\ &= \sum\nolimits_k E^{x_k}(\boldsymbol{f}_q^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q)) - \log Z^{x_k}(\boldsymbol{s}^{x_k}).\end{aligned} \tag{4.15}$$

Additionally, from Equation (4.7), given the normalization term does not depend on the action $\boldsymbol{a}^q$, we can neglect it from our objective function and optimize over the sum of the energy functions. In our work, we propose to compute the control action for our robot by the maximization of Equation (4.15), $\boldsymbol{a}^q = \arg\max_{\boldsymbol{a}^q} \mathcal{J}(\boldsymbol{a}^q)$. In the general case, this optimization function lacks an analytical solution and we will use stochastic optimization methods to optimize it [36].

Framing the robot control in terms of an implicit function has several interesting properties in contrast with the explicit counterpart [104, 191]. The first is that we are not constrained in the policy function. In CEP we can assume an arbitrary stochastic policy to represent each component, whereas explicit models assume deterministic policies. As we show in Section 4.3, RMP components can be thought as normal distributions from CEP lenses. This policy model freedom provides the practitioner with a much wider range of opportunities to design policies or learn them with arbitrary energy based models [242, 52]. An implicit representation has additional relevant properties with respect to APF and RMP methods. To compute the desired acceleration in the configuration space; explicit methods require to invert the transformation map $\ddot{\boldsymbol{x}} \approx \boldsymbol{J}\ddot{\boldsymbol{q}}$, to move the desired acceleration from the task space to the configuration space. If the robot's configuration is close to a singularity, a small velocity in task space will result in a big velocity in joint space. In contrast, in CEP, given we are considering the implicit representation, we do not require to invert the transformation map as the energy is directly evaluated in the task space.

4.2.4. Optimization of composable energy policies

In the following, we introduce the algorithm we use to solve our optimization problem. In our problem, the optimal action is obtained by a maximization over the logarithm of the product of a set of expert policies Equation (4.8). From the derivation in Equation (4.15), if each policy is defined by a Gibbs distribution, the optimization function is

$$\boldsymbol{a}^{q*} = \arg\max_{\boldsymbol{a}^q} \sum\nolimits_k E^{x_k}(\boldsymbol{f}_q^{x_k}(\boldsymbol{a}^q, \boldsymbol{s}^q)), \tag{4.16}$$

with E^{x_k} being a set of given energy functions defined in arbitrary task spaces $\mathcal{X}^k$ and $\boldsymbol{f}_q^{x_k}$ being the transformation map that transforms a state-action pair in the configuration space $(\boldsymbol{s}^q, \boldsymbol{a}^q) \in \mathcal{Q}$ to the different task spaces $\mathcal{X}^k$.

We aim to solve the optimization in Equation (4.16) in high control frequencies (100Hz-1kHz) to run it as a reactive motion generator. Additionally, the energy function might be non-differentiable. We propose to solve the optimization problem in Equation (4.16) by cross-entropy methods [17]. The proposed method is presented in Algorithm 6.

We initialize our algorithm transforming the state in the configuration space, $\boldsymbol{s}^q$ to the different task spaces $\boldsymbol{s}^{x_k}$. As shown in Equation (4.7), the task space states do not depend on the action and we directly compute them given the configuration state. In terms of computational efficiency, it is relevant to compute the task space state out of the optimization loop as it might be computationally demanding (*usually, we compute the forward kinematics in this stage*). Then, we initialize the cross-entropy optimization for the configuration space action. We define a proposed sampling distribution $q(\boldsymbol{a}^q) = \mathcal{N}(\boldsymbol{a}^q|\boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I})$. Then, for I optimization steps, we first sample a set of N action candidates in the configuration space $\boldsymbol{a}^q_{0:N}$. To evaluate the samples, we first transform the samples to the set of K task spaces $\boldsymbol{a}^{x_k}_{0:K}$. In practise, we consider an affine map between $\boldsymbol{a}^q$ and $\boldsymbol{a}^{x_k}$.Thus, we can apply tensor multiplication and transform all the samples $\boldsymbol{a}^q_{0:N}$ to all the task spaces accelerations in a single step. Finally, the energies are computed on each energy component and the contributions summed.

In our problem, we consider two approaches to update the sampling distribution $q(\boldsymbol{a}^q)$. As proposed in [17], the mean and variance are updated by first selecting the M particles with the highest energy value. Then, the optimal mean and variance are computed by

$$\begin{aligned} \boldsymbol{\mu}^* &= \frac{1}{M}\sum_{m=0}^{M} \boldsymbol{a}^q_m \\ \boldsymbol{\sigma}^{*2} &= \frac{1}{M}\sum_{m=0}^{M} (\boldsymbol{a}^q_m - \boldsymbol{\mu}^*)^2. \end{aligned} \tag{4.17}$$

Algorithm 6: Composable Energy Policies

```
Given : $N$: Number of samples;
$\boldsymbol{s}^q$: Current state in configuration space;
$K$: Number of energy components;
$(\boldsymbol{f}_q^{x_1}, E^{x_1}), \dots, (\boldsymbol{f}_q^{x_K}, E^{x_K}))$: Task maps and energies;
$I$: Optimization steps;
$(\mu_0, \Sigma_0)$: Initial sampling distribution mean and variance;
$(\boldsymbol{a}^*, e^*)$: Initial optimal action and energy;

for $k \leftarrow 1$ to $K$ do
    $\boldsymbol{s}^{x_k} = \boldsymbol{f}_{qs}^{x_k}(\boldsymbol{s}^q)$;    Map configuration state to task states
for $i \leftarrow 1$ to $I$ do
    $\boldsymbol{a}_{0:N}^q \sim \mathcal{N}(\mu_i, \Sigma_i)$;    Sample N action candidates
    for $k \leftarrow 1$ to $K$ do
        $\boldsymbol{a}_n^{x_k} = \boldsymbol{f}_q^{x_k}(\boldsymbol{s}^q, \boldsymbol{a}_{0:N}^q)$;    Map actions to task spaces
        $e_{0:N}^{x_k} = E^{x_k}(\boldsymbol{s}^{x_k}, \boldsymbol{a}_{0:N}^{x_k})$;    Evaluate energy
    $e_{0:N}^q = \sum_{k=1}^K e_{0:N}^{x_k}$;    Sum all energies
    $\mu_{i+1} \leftarrow \text{Update}_\mu(\mu_i, \boldsymbol{a}_{0:N}^q, e_{0:N}^q)$;    With Equation (4.17) or Equation (4.18)
    $\Sigma_{i+1} \leftarrow \text{Update}_\Sigma(\Sigma_i, \boldsymbol{a}_{0:N}^q, e_{0:N}^q)$;    With Equation (4.17) or Equation (4.18)
    $\boldsymbol{a}_i^*, e_i^* \leftarrow \text{arg}_{\boldsymbol{a}^q} \text{max}_e(e_{0:N}^q)$;    pick optimal action
    if $e^* < e_i^*$ then
        $\boldsymbol{a}^* \leftarrow \boldsymbol{a}_i^*$;
        $e^* \leftarrow e_i^*$;
return $\boldsymbol{a}^*$;
```

Alternatively, we also considered a soft update version. We represent the update by a reward weighted regression [177]

$$\begin{aligned}\boldsymbol{\mu}^* &= \frac{1}{\sum_{k=0}^{N} \omega_k} \sum_{n=0}^{N} \omega_n \boldsymbol{a}_n^q \\ \boldsymbol{\sigma}^{*2} &= \frac{1}{\sum_{k=0}^{N} \omega_k} \sum_{n=0}^{N} \omega_n (\boldsymbol{a}_n^q - \boldsymbol{\mu}^*)^2,\end{aligned} \tag{4.18}$$

with $\omega_n = \beta \exp(-\beta e_n^q)$. e_n^q is the total energy for the n action sample and $\beta > 0$ is a temperature parameter that scales the energies for the weighted mean and variance in Equation (4.18).

Rather by cross-entropy [17] or by reward weighted regression [177], the mean and standard deviation for the next optimization step $\boldsymbol{\mu}_{i+1}$ is computed by smoothing the solution between the optimal one $\boldsymbol{\mu}^*$ and the previous one $\boldsymbol{\mu}_i$

$$\begin{aligned}\boldsymbol{\mu}_{i+1} &= \alpha \boldsymbol{\mu}_i + (1-\alpha)\boldsymbol{\mu}^* \\ \boldsymbol{\sigma}_{i+1} &= \alpha \boldsymbol{\sigma}_i + (1-\alpha)\boldsymbol{\sigma}^*.\end{aligned} \tag{4.19}$$

Smoothing is often crucial to prevent premature shrinking of the sampling distribution [17].

It is common in model predictive control algorithms [160] to assume that consecutive optimal control problems are similar to each other. This allows initializing the optimization with the previously computed optimal solution. In our work, we assume our optimization problem is myopic (*we only optimize for a single look ahead step*) and the energy functions might be non-continuous. Thus, we lack any guarantee of the consecutive optimal control problems to be similar to each other. In conclusion, we always initialize our optimization problem with zero mean and a sufficiently wide standard deviation.

4.3. An inference view on policy composition

In this section, we derive from an inference view the proposed optimization problem in Equation (4.1). We additionally highlight the connections between RMP and CEP and prove that RMP methods can be considered a particular case of CEP.

Let us assume we aim to find the action distribution that satisfies in the optimal way a set of stochastic policies $\pi_k(\boldsymbol{a}|\boldsymbol{s})$. Similarly to control-as-inference [195, 126] approaches, we introduce an additional variable o_{π_k}. This variable is a binary random variable, where $o_{\pi_k} = 1$ denotes how likely state-action pair is optimal for the policy π_k and $o_{\pi_k} = 0$ denotes how unlikely. We choose to model the distribution over the ”likelihood variable” o_{π_k} by

$$p(o_{\pi_k} = 1|\boldsymbol{s}, \boldsymbol{a}) \propto \pi_k(\boldsymbol{a}|\boldsymbol{s}) \propto \exp(E_k(\boldsymbol{s}, \boldsymbol{a})). \tag{4.20}$$

From Equation (4.20), we can observe that for those cases in which the distribution is conditioned on the optimal state-action pairs for a particular policy π_k, the probability for $o_{\pi_k} = 1$ is going to be high. While if the action is not an action with a high likelihood for $\pi_k(\cdot|\boldsymbol{s})$, the optimality probability $p(o_{\pi_k} = 1|\boldsymbol{s}, \boldsymbol{a})$ will be low.

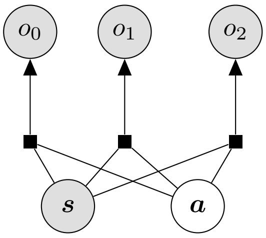


Figure 4.2.: Graphical model for Composable Energy Policies. o_k is an auxiliary variable that represents the optimality of $\boldsymbol{s}_0$ and $\boldsymbol{a}_0$ for a particular policy.

Let us consider we aim to be optimal for a set of policies π_k. We can represent the Bayes net relating the optimality variables and the state and action as in Figure 4.2. The likelihood for the graphical model in Figure 4.2 can be computed as the product of the terms

$$p(\boldsymbol{s}, \boldsymbol{a}, o_{0:2}) = q(\boldsymbol{a})p(\boldsymbol{s}) \prod_{k=0}^{2} p(o_k|\boldsymbol{s}, \boldsymbol{a}). \tag{4.21}$$

with $p(\boldsymbol{s})$ and $q(\boldsymbol{a})$ the prior distributions for the state and the action consecutively.

Following the Bayes net, we can represent the posterior distribution over the action space when conditioned to $o_{\pi_k} = 1$ and $\boldsymbol{s} = \boldsymbol{s}_0$ by

$$\begin{aligned} p(\boldsymbol{a}|\boldsymbol{s} = \boldsymbol{s}_0, o_{0:K} = \boldsymbol{1}) &\propto q(\boldsymbol{a}) \prod_{k=0}^{K} p(o_{\pi_k} = 1|\boldsymbol{s}_0, \boldsymbol{a}) \\ &\propto q(\boldsymbol{a}) \prod_{k=0}^{K} \pi_k(\boldsymbol{a}|\boldsymbol{s}) \propto q(\boldsymbol{a}) \exp\left(\sum_{k=0}^{K} E_k(\boldsymbol{a}, \boldsymbol{s})\right). \end{aligned} \tag{4.22}$$

Considering the prior distribution over the action space to be uniform, $q(\boldsymbol{a}) = 1/\mathcal{A}$, we observe that the posterior distribution given all the optimality variables are 1 is the product of policies. By taking the *maximum a posteriori* estimate, $\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}} p(\boldsymbol{a}|\boldsymbol{s} = \boldsymbol{s}_0, o_{0:K} = \mathbf{1})$, we compute the action that optimizes over the composition of all optimal distributions. The optimization problem is the one of Equation (4.1).

In our work, we additionally consider a hard constraint that relates the state-action pairs in the configuration space with the state-action pairs in the task space Equation (4.8). We integrate the constraints between the configuration space and the task space by a deterministic node in the graphical model (Figure 4.3). The deterministic node in the transformation will induce a delta distribution relating the state action pairs in the configuration space and in the task spaces

$$p(\boldsymbol{s}^{x_k}, \boldsymbol{a}^{x_k}|\boldsymbol{s}^q, \boldsymbol{a}^q) = \delta((\boldsymbol{s}^{x_k}, \boldsymbol{a}^{x_k}) - \boldsymbol{f}_q^{x_k}(\boldsymbol{s}^q, \boldsymbol{a}^q)). \tag{4.23}$$

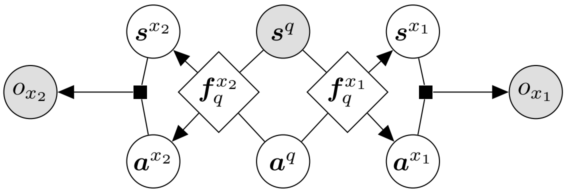


Figure 4.3.: Graphical model for Composable Energy Policies with task space policies. o_{x_k} is an auxiliary variable that represents the optimality of $\boldsymbol{s}^{x_k}$ and $\boldsymbol{a}^{x_k}$ for a particular policy in that task space.

The likelihood function for the graphical model is represented by the product of the terms

$$p(\boldsymbol{s}^q, \boldsymbol{a}^q, \boldsymbol{s}^{x_{0:2}}, \boldsymbol{a}^{x_{0:2}}, o_{x_{0:2}}) = \\ p(\boldsymbol{s}^q)q(\boldsymbol{a}^q)\prod_{k=0}^{2} p(\boldsymbol{s}^{x_k}, \boldsymbol{a}^{x_k}|\boldsymbol{s}^q, \boldsymbol{a}^q)p(o_{x_k}|\boldsymbol{s}^{x_k}, \boldsymbol{a}^{x_k}). \tag{4.24}$$

Given that we aim to compute the posterior for $\boldsymbol{a}^q$ and assuming $\boldsymbol{s}^q$ and $o^{x_{0:2}}$ are given, we marginalize the joint distribution with respect to the rest of the variables

$$p(\boldsymbol{s}^q, \boldsymbol{a}^q, o_{x_{0:2}}) = \\ \int_{\boldsymbol{s}^{x_{0:2}}}\int_{\boldsymbol{a}^{x_{0:2}}} p(\boldsymbol{s}^q, \boldsymbol{a}^q, \boldsymbol{s}^{x_{0:2}}, \boldsymbol{a}^{x_{0:2}}, o_{x_{0:2}})\mathrm{d}\boldsymbol{s}^{x_{0:2}}\mathrm{d}\boldsymbol{a}^{x_{0:2}}. \tag{4.25}$$

The relation between the configuration state-action pairs and the task space action pairs is given by Equation (4.23). Given the relation is defined by a delta distribution, the marginal distribution can be represented by a simple substitution of variables

$$p(\boldsymbol{s}^q, \boldsymbol{a}^q, o_{x_{0:2}}) = q(\boldsymbol{a}^q)p(\boldsymbol{s}^q)\prod_{k=0}^{2} p(o_{x_k}|\boldsymbol{s}^q, \boldsymbol{a}^q), \tag{4.26}$$

with $p(o_k|\boldsymbol{s}^q, \boldsymbol{a}^q) \propto \exp(E_k(\boldsymbol{f}_q^{x_k}(\boldsymbol{s}^q, \boldsymbol{a}^q)))$. Now, we can follow a similar derivation to Equation (4.22) and compute the posterior distribution for the configuration space action $\boldsymbol{a}^q$.

4.3.1. Riemannian Motion Policies as Composable Energy Policies

The control-as-inference literature [8, 234, 126] have widely studied the connections between the cost functions and the distributions related to them. From this viewpoint, RMP objective Equation (4.6) can be framed as a particular case of Equation (4.8) where each policy component is represented by a normal distribution. In the following, we derive the Riemannian motion policies from an inference perspective and show its relation with CEP.

Suppose each policy π is modelled by a normal distribution, where the mean, $\boldsymbol{g}^{x_k}$, is the desired optimal action in the task space $\mathcal{X}_k$, and the precision matrix, $\boldsymbol{\Lambda}^{x_k}$, is the metric on the task space

$$\pi(\boldsymbol{a}^{x_k}|\boldsymbol{s}^{x_k}) = \mathcal{N}(\boldsymbol{g}^{x_k}(\boldsymbol{s}^{x_k}), \boldsymbol{\Lambda}^{x_k}(\boldsymbol{s}^{x_k})). \tag{4.27}$$

We consider the action is defined by the acceleration, $\boldsymbol{a}^{x_k} = \ddot{\boldsymbol{x}}^k$ and the state by the position and velocity $\boldsymbol{s}^{x_k} = (\boldsymbol{x}^k, \dot{\boldsymbol{x}}^k)$.

In RMP methods the action in the task space $\mathcal{X}_k$ and in the configuration space $\mathcal{Q}$ are approximately related by the pseudo-inverse Jacobian of the forward kinematics, $\ddot{\boldsymbol{q}} \approx \boldsymbol{J}^{x_k^\dagger}\ddot{\boldsymbol{x}}^k$. Given the map is linear, the policy distribution in the task space Equation (4.27) remains a normal distribution in the configuration space

$$\pi(\boldsymbol{a}^q|\boldsymbol{s}^q) = \mathcal{N}(\boldsymbol{J}^{x_k^\dagger}\boldsymbol{g}^{x_k}, \boldsymbol{J}^{x_k^\intercal}\boldsymbol{\Lambda}^{x_k}\boldsymbol{J}^{x_k}), \tag{4.28}$$

with the mean $\boldsymbol{g}^q = \boldsymbol{J}^{x_k^\dagger}\boldsymbol{g}^{x_k}$ and the precision matrix $\boldsymbol{\Lambda}^q = \boldsymbol{J}^{x_k^\intercal}\boldsymbol{\Lambda}^{x_k}\boldsymbol{J}^{x_k}$. In CEP, we assume the posterior distribution to maximize is modeled by the product of each policy Equation (4.22). For the particular case in which every policy is represented by Equation (4.28), the product of the policies remains a Gaussian, $p(\boldsymbol{a}|\boldsymbol{s}=\boldsymbol{s}_0, o_{0:K}=\mathbf{1}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Lambda})$ with

$$\begin{aligned}\boldsymbol{\mu} &= (\sum_j \boldsymbol{\Lambda}_j^q)^{-1} \sum_k \boldsymbol{\Lambda}_k^q \boldsymbol{g}_k^q \\ \boldsymbol{\Lambda} &= \sum_k \boldsymbol{\Lambda}_k^q.\end{aligned} \tag{4.29}$$

As we observe, the mean of the product of Gaussians is just a weighted-sum of the mean of each independent component and the precision matrix is the sum of each component. In CEP, the action is computed by a maximum a posteriori estimate over the posterior distribution $p(\boldsymbol{a}|\boldsymbol{s}=\boldsymbol{s}_0, o_{0:K}=\mathbf{1})$. For the particular case in which the posterior is the normal distribution in Equation (4.29), the maximum a posteriori is the mean of the Gaussian

$$\ddot{\boldsymbol{q}}^* = (\sum_j \boldsymbol{\Lambda}_j^q)^{-1} \sum_k \boldsymbol{\Lambda}_k^q \boldsymbol{g}_k^q. \tag{4.30}$$

As expected, the solution is the one from the RMP Equation (4.4). As shown in [240], a similar derivation can be follow to represent APF as special cases of CEP.

In conclusion, we can derive the RMP solution as a special case of Equation (4.8). To do so, we assume each policy component is represented by a normal distribution with the mean equal to the desired acceleration and the precision matrix equal to the metric. We hypothesize that in some tasks, representing all the policy components by a normal distribution might not be expressive enough to solve the task properly. Normal distributions assume that (1) there is a unique optimal action (the mean) to solve the task and (2) the quality of the actions is related to the Mahalanobis distance to the optima. While tasks like reaching a target might satisfy (1) and (2); tasks like obstacle avoidance might require richer representations to properly solve the task. Rather than limiting the policies to normal distributions, we consider arbitrary shape distributions to represent each objective. In the experimental section, we show empirically that for some tasks, modeling the policy with non-normal distributions might lead to better cooperation with the other components.

The scope of this work is to study the relations and performances of one-step control horizon controllers. Nevertheless, given the clear relations of CEP with control-as-inference in longer horizon problems, we study these relations in the Appendix A.2.

Task definition	Input space $(\boldsymbol{x}, \dot{\boldsymbol{x}}, \ddot{\boldsymbol{x}})$	transformation $(\boldsymbol{f})$	advantage function (A)	reward function (r)
reach target position	Cartesian Task Space $(\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)$	$\boldsymbol{x} = \boldsymbol{x}$ $\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}$ $\ddot{\boldsymbol{x}} = \ddot{\boldsymbol{x}}$	$A(\ddot{\boldsymbol{x}}\vert\boldsymbol{x}, \dot{\boldsymbol{x}}) = -\vert\vert\ddot{\boldsymbol{x}} - \ddot{\boldsymbol{x}}_g(\boldsymbol{x}, \dot{\boldsymbol{x}})\vert\vert^2_{\boldsymbol{\Lambda}^{\ddot{x}}}$ $\ddot{\boldsymbol{x}}_g(\boldsymbol{x}, \dot{\boldsymbol{x}}) = \frac{2}{\Delta t^2}(\boldsymbol{x}_g - \boldsymbol{x} - \Delta t\dot{\boldsymbol{x}})$ $\boldsymbol{\Lambda}^{\ddot{x}} = \frac{\Delta t^4}{4}\boldsymbol{\Lambda}$	$r(\boldsymbol{x}) = -\vert\vert\boldsymbol{x} - \boldsymbol{x}_g\vert\vert^2_{\boldsymbol{\Lambda}}$
reach target orientation	Orientation Task Space $(SO(3), \mathbb{R}^3, \mathbb{R}^3)$	$\boldsymbol{\theta} = \mathrm{LogMap}_{\boldsymbol{R}_g}(\boldsymbol{R})$ $\boldsymbol{\omega}' = \boldsymbol{R}_g^{-1}\boldsymbol{\omega}$ $\dot{\boldsymbol{\omega}}' = \boldsymbol{R}_g^{-1}\dot{\boldsymbol{\omega}}$	$A(\dot{\boldsymbol{\omega}}'\vert\boldsymbol{\theta}, \boldsymbol{\omega}') = -\vert\vert\dot{\boldsymbol{\omega}}' - \dot{\boldsymbol{\omega}}_g(\boldsymbol{\theta}, \boldsymbol{\omega}')\vert\vert^2_{\boldsymbol{\Lambda}^{\dot{\omega}'}}$ $\dot{\boldsymbol{\omega}}_g(\boldsymbol{\theta}, \boldsymbol{\omega}') = \frac{2}{\Delta t^2}(\boldsymbol{\theta} - \Delta t\boldsymbol{\omega}')$ $\boldsymbol{\Lambda}^{\dot{\omega}'} = \frac{\Delta t^4}{4}\boldsymbol{\Lambda}$	$r(\boldsymbol{\theta}) = -\vert\vert\boldsymbol{\theta}\vert\vert^2_{\boldsymbol{\Lambda}}$
avoid obstacles	Cartesian Task Space $(\mathbb{R}^3, \mathbb{R}^3, \mathbb{R}^3)$	$d_o = \vert\vert\boldsymbol{x} - \boldsymbol{x}_o\vert\vert$ $\hat{\boldsymbol{v}}_o = (\boldsymbol{x} - \boldsymbol{x}_o)/d_o$ $\dot{x}_p = \dot{\boldsymbol{x}} \cdot \hat{\boldsymbol{v}}_o$ $\ddot{x}_p = \ddot{\boldsymbol{x}} \cdot \hat{\boldsymbol{v}}_o$	$A(\ddot{x}_p\vert\dot{x}_p, d_o) = \begin{cases} 0 & \text{if} \quad \ddot{x}_p > \alpha^{\ddot{x}_p}(\dot{x}_p, d_o) \\ -\infty & \text{if} \quad \ddot{x}_p \leqslant \alpha^{\ddot{x}_p}(\dot{x}_p, d_o) \end{cases}$ $\alpha^{\ddot{x}_p}(\dot{x}_p, d_o) = \frac{2}{\Delta t^2}(\alpha - d_o - \dot{x}_p\Delta t)$	$r(d_o) = \begin{cases} 0 & \text{if} \quad d_o > \alpha \\ -\infty & \text{otherwise} \end{cases}$
avoid joint limits	Configuration Space $(\mathbb{R}^7, \mathbb{R}^7, \mathbb{R}^7)$	$\boldsymbol{q} = \boldsymbol{q}$ $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}$ $\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{q}}$	$A(\ddot{\boldsymbol{q}}\vert\dot{\boldsymbol{q}}, \boldsymbol{q}) = 0 \begin{cases} 0 & \text{if} \quad \ddot{\boldsymbol{q}} > \underline{\ddot{\boldsymbol{q}}} \quad \text{and} \quad \ddot{\boldsymbol{q}} < \bar{\ddot{\boldsymbol{q}}} \\ -\infty & \text{otherwise} \end{cases}$ $\underline{\ddot{\boldsymbol{q}}}(\dot{\boldsymbol{q}}, \boldsymbol{q}) = \frac{2}{\Delta t^2}(\underline{\boldsymbol{q}} - \boldsymbol{q} - \dot{\boldsymbol{q}}\Delta t)$ $\bar{\ddot{\boldsymbol{q}}}(\dot{\boldsymbol{q}}, \boldsymbol{q}) = \frac{2}{\Delta t^2}(\bar{\boldsymbol{q}} - \boldsymbol{q} - \dot{\boldsymbol{q}}\Delta t).$	$r(\boldsymbol{q}) = \begin{cases} 0 & \text{if} \quad \boldsymbol{q} > \underline{\boldsymbol{q}} \text{ and } \boldsymbol{q} < \bar{\boldsymbol{q}} \\ -\infty & \text{otherwise} \end{cases}$
joint velocity control	Configuration Space $(\mathbb{R}^7, \mathbb{R}^7, \mathbb{R}^7)$	$\boldsymbol{q} = \boldsymbol{q}$ $\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}$ $\ddot{\boldsymbol{q}} = \ddot{\boldsymbol{q}}$	$A(\ddot{\boldsymbol{q}}\vert\boldsymbol{q}, \dot{\boldsymbol{q}}) = -\vert\vert\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_g(\dot{\boldsymbol{q}})\vert\vert^2_{\boldsymbol{\Lambda}^{\ddot{q}}}$ $\ddot{\boldsymbol{q}}_g(\dot{\boldsymbol{q}}) = \dot{\boldsymbol{q}}/\Delta t$ $\boldsymbol{\Lambda}^{\ddot{q}} = \Delta t^2\boldsymbol{\Lambda}$	$r(\dot{\boldsymbol{q}}) = -\vert\vert\dot{\boldsymbol{q}}\vert\vert^2_{\boldsymbol{\Lambda}}.$

Table 4.1.: Resume of the proposed basic local reactive energies. To compute a particular energy, first the input position $\boldsymbol{x}$, velocity $\dot{\boldsymbol{x}}$ and acceleration $\ddot{\boldsymbol{x}}$ are transformed to a latent space by the maps in the third column. Then, the advantage is computed in the latent space. The advantage function represents the energy function of our policy. Last column shows the reward that each policy is trying to maximize.

4.4. Composable energy policies for robot reinforcement learning

In Reinforcement Learning, we deal with the problem of finding the policy π that maximizes the accumulated reward, $\mathcal{R}$

$$\max_{\pi} \mathbb{E}_{p_{\pi}(\boldsymbol{s},\boldsymbol{a})}[\mathcal{R}(\boldsymbol{s},\boldsymbol{a})]. \tag{4.31}$$

with $\rho_{\pi}(\boldsymbol{s},\boldsymbol{a})$ being the stationary state action distribution, given some transition dynamics, $p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})$ and initial state distribution $p(\boldsymbol{s}_0)$. Applying reinforcement learning in real robot environments usually consider high dimensional state-action spaces and sparse rewards. Thus, finding a good policy might require many iterations in the environment before a desirable policy is found.

A common approach to reduce the sample complexity is by integrating as many priors as possible in the problem. Properly chosen priors might accelerate the learning process, biasing the exploration towards meaningful states. Additionally, with the proper priors, we could increase the safety guarantees in the exploration process.

There are multiple ways to integrate priors in a reinforcement learning problem. A common option is to do reward shaping. Adding additional reward signals to the problem, we can guide the learning process to informative states. Another common option is assuming a set of expert demonstrations are given, we can pretrain our policy to match the expert demonstrations. This approach is known as behavioral cloning. In our work, we explore the option of using structured policies.

A structured policy can be represented as follows:

$$\begin{aligned} \boldsymbol{\psi} &\sim \pi_{\text{RL}}(\boldsymbol{\psi}|\boldsymbol{s}) \\ \boldsymbol{a} &= \pi_{\text{struct}}(\boldsymbol{s};\boldsymbol{\psi}). \end{aligned} \tag{4.32}$$

A structured policy allows modifying the action space in which the RL agent learns the policy. Rather than directly sampling an action $\boldsymbol{a}$ from the RL agent; we sample a set of parameters $\boldsymbol{\psi}$. Then, these parameters are input in a low-level structured policy π_{struct} and the action is computed. Through the action space transformation, structured policies allow a faster and safer learning process.

There are several type of structured policies. In residual policy learning [93, 218], after sampling an action from the RL agent, an expert policy π_E action is summed to bias the exploration towards meaningful regions, $\pi_{\text{struct}}(\boldsymbol{s},\boldsymbol{\psi}) = \pi_E(\boldsymbol{s}) + \boldsymbol{\psi}$. The RL agent learns the residual actions around the expert policy. In [34] the parameters $\boldsymbol{\psi}$ select a DMP and sets some parameters of the DMP, such as the target. Then, the DMP is executed for a certain period, before sampling new parameters from π_{RL}.

In our work, we propose to model the low-level structured policy by the maximization over a composition of policies

$$\pi_{\text{struct}}(\boldsymbol{s}, \boldsymbol{\psi}) = \arg\max_{\boldsymbol{a}} \log \left(\prod_{k=0}^{K} \pi_k(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\psi}) \right). \tag{4.33}$$

In contrast with previous works that define an explicit model to represent the structured policy, we propose to model the structured policy by a maximization over an implicit function. In our approach, we first sample a set of parameters from the RL agent. Then, these parameters condition some of the policies on the objective function. Finally, we solve the optimization problem in Equation (4.33) to obtain the action to apply in the system. Considering an implicit function to represent the low-level policy has multiple benefits. An important one is related to safe exploration. Given we are solving a search problem, we can guarantee the robot is not choosing an action that would move the robot to a collision. We could also set some prior policies that encourage smooth behaviors and we could avoid high trembling while exploring. In conclusion, the robot explores the parameter space of an objective function. Then, given we have set some prior knowledge in this objective function, we can solve an optimization problem and apply the optimal action satisfying the objective.

There are multiple choices to parameterize the objective function. We show an example of how the energy policies and the RL action can be combined.

A simple option is to parameterize a reaching policy. We choose the target reaching policy proposed in Appendix A.4. We can both parameterize the target position $\boldsymbol{x}_g$ or the metric $\boldsymbol{\Lambda}$

$$\begin{aligned} A(\ddot{\boldsymbol{x}}|\boldsymbol{x}, \dot{\boldsymbol{x}}) &= -||\ddot{\boldsymbol{x}} - \ddot{\boldsymbol{x}}_g(\boldsymbol{x}, \dot{\boldsymbol{x}})||^2_{\boldsymbol{\Lambda}^{\ddot{x}}} \\ \ddot{\boldsymbol{x}}_g(\boldsymbol{x}, \dot{\boldsymbol{x}}) &= \frac{2}{\Delta t^2}(\boldsymbol{\psi}_{\boldsymbol{x}_g} - \boldsymbol{x} - \Delta t \dot{\boldsymbol{x}}) \\ \boldsymbol{\Lambda}^{\ddot{x}} &= \frac{\Delta t^4}{4}\boldsymbol{\psi}_{\boldsymbol{\Lambda}}. \end{aligned}$$

Parameterizing $\boldsymbol{x}_g$ allows the reinforcement learning policy to set the desired target location given the current state $\boldsymbol{s}$, while parameterizing $\boldsymbol{\Lambda}$ allows the reinforcement learning agent to weight the influence of this component. It is important to remark, that for those cases in which $\boldsymbol{\Lambda}$ is big, the influence of this component in the composition increases and then, the influence of the reinforcement learning action. When $\boldsymbol{\Lambda}$ is small the contribution of this component decays and the inductive biases will define the movement.

Nevertheless, we remark, that we are not limited to reaching policies. We could parameterize any energy policy presented in Appendix A.4, probabilistic motion primitives [166] or handcrafted policies.

In our experiments we combine parameterized policies with fixed prior policies

$$\pi(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\psi}) = \prod\nolimits_k \pi_{k\text{prior}}(\boldsymbol{a}|\boldsymbol{s}) \prod\nolimits_j \pi_j(\boldsymbol{a}|\boldsymbol{s},\boldsymbol{\psi}). \tag{4.34}$$

Integrating a parameterized reaching target policy with a fixed obstacle avoidance policy, allows the robot to explore with the guarantee of exploring safely. We also consider combining attractor policies to a certain target and parameterized policies. Combining both, the reinforcement learning agent explores while the attractor policy guides the robot to informative regions. The performance is similar to a residual policy, in which the RL agent learns in the residuals of the guiding policy.

4.5. Experimental evaluation

The experimental evaluation is split into three parts. In the first part (Section 4.5.1), we evaluate qualitatively the performance of CEP in a 2D navigation environment. The experiment is performed to provide a visual intuition on how CEP represents its policy.

In the second part (Section 4.5.2), we investigate the performance of CEP for local reactive navigation in cluttered environments. The experiments are performed in a 7 dof Kuka-LWR robot. In a set of simulated environments (Section 4.5.2), we evaluate how the energy composition performs by observing the success rate and the collisions in a set of obstacle avoidance environments. Additionally, we perform ablation studies to find the optimal parameters and also to find the maximum control frequencies. Then, in a real robot environment (Section 4.5.2), we investigate the performance of CEP to solve a pick and place task in a cluttered environment. We measure the performance under human disturbances, picking position changes and placing position changes.

In the third part (Section 4.5.3), we investigate the benefit of integrating CEP as a structured policy for robot reinforcement learning. We first evaluate the performance of CEP as a structured policy while learning how to hit a puck and place it in a target position (Section 4.5.3). We want to observe if using CEP as prior boosts the learning performance

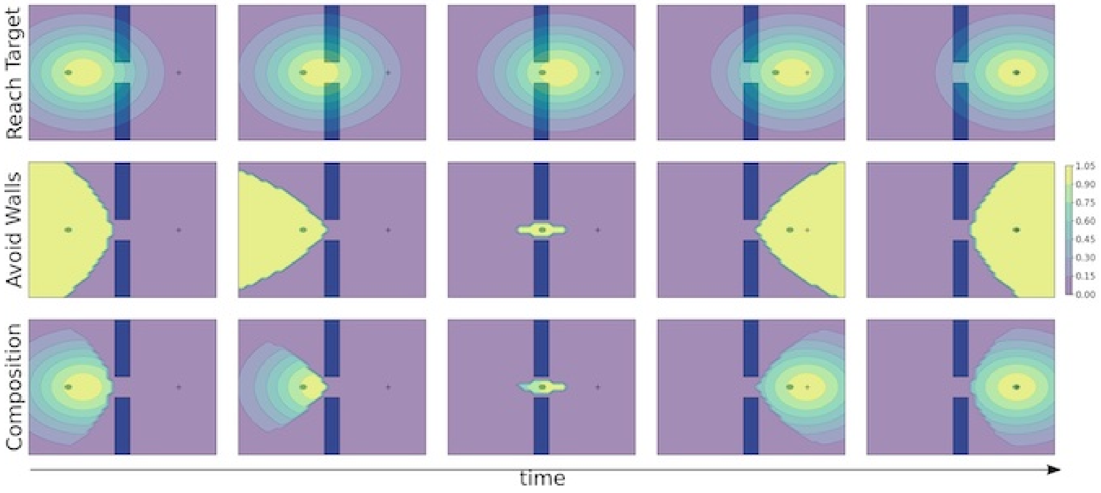


Figure 4.4.: A visual representation of the next state distribution $p(\boldsymbol{s}')$ running a set of designed policies. We visualize the distribution for different states in a 2D navigation task. In the top: next state distribution after applying a reaching target policy. In the middle: next state distribution after applying an obstacle avoidance policy. In the bottom: next state distribution after applying the composition of a reaching target and an obstacle avoidance policy.

of an RL agent. Additionally, we want to evaluate if adding an obstacle avoidance prior reduces the number of collisions in the training. The experiments are performed for three MDP's that vary in the reward function.

4.5.1. Visual 2D particle environment

In the first experimental section, we aim to provide a visual understanding of the proposed policy composition. We investigate the composition of a set of energy policies proposed in Appendix A.4 for a 2D navigation problem. We consider the toy environment in Figure 4.5 (a). We want to reach the target (cross) with the robot (blue circle) avoiding the walls (blue rectangles). To properly compute the obstacle avoidance, we represent the collision bodies for the walls and the robot with a set of spheres (Figure 4.5 (b)).

We model the composable energy policy with two component: a target reaching energy policy and a set of obstacle avoidance energy policies (one per each obstacle sphere in the wall). We model both energy policies with the proposed energy policies in Appendix A.4. The attractor energy is a quadratic function. For obstacle avoidance, we consider N energy policies. Each energy function is a binary function that penalizes the actions that move the robot below a certain distance threshold with respect to the obstacles.

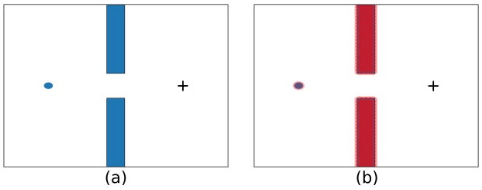


Figure 4.5.: 2D navigation task. (a) Environment. The robot is represented by a blue circle, the walls by the rectangles and the target by the cross. (b) spherical obstacle bodies for the robot and the walls.

We visualize the probability density functions for each policy component and for the composition of them in Figure 4.4. Visualizing directly the distribution in the acceleration space is not informative. To provide an intuitive visualization, we plot the probability density function for the next state $p(\boldsymbol{s}')$, given the robot is sampling an action from a certain energy policy π

$$p(\boldsymbol{s}') = \mathbb{E}_{\pi(\boldsymbol{a}|\boldsymbol{s})}[\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{a})], \tag{4.35}$$

with $\boldsymbol{f}$ being the linear dynamics in Equation (A.28). Given these policies have been designed as the maximum entropy policies maximizing the next state reward, the next state distribution is naturally, $p(\boldsymbol{s}') \propto \exp(r(\boldsymbol{s}'))$.

We observe that the reaching target policy defines a normal distribution with the mean in an interpolation between the target position and the current position. The most likely action is the one that moves the robot to the mean position. The sharpness of the distribution is defined by the metric defined in Appendix A.4. For the case of the obstacle avoidance policy, the next state distribution is a uniform distribution that sets the mass on a polytope defined by the shortest distances to the obstacles. This distribution will put zero mass to any action that moves the robot out of the polytope. For any action keeping the robot inside the polytope, the distribution remains constant.

The product of the two policies is a complex distribution that weights the influence of both. The obtained distribution is an attractor normal distribution truncated by the collision avoidance policy. We remark that rather than computing the maxima if we apply MCMC to sample from this distribution, the robot will never choose an action that collides against the obstacles and will sample actions that move towards the target with a higher probability.

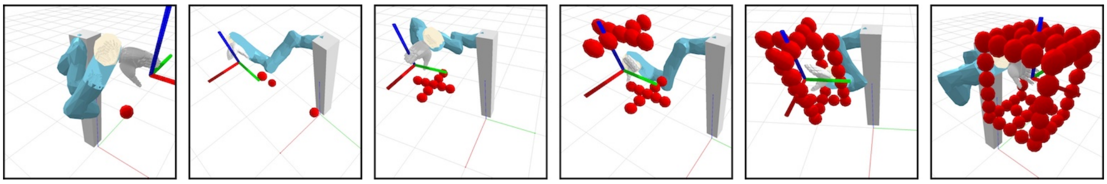


Figure 4.6.: Simulated Environments for the reaching through clutter environments experiment. From left to right: 1 obstacle, 3 obstacles, Cross, Double Cross, Cage I and Cage II.

Methods	1 Obstacle		3 Obstacles		Cross		Double Cross		Cage I		Cage II	
	Success	Collide	Success	Collide	Success	Collide	Success	Collide	Success	Collide	Success	Collide
Riemannian Motion Policies [191]	100/100	0/100	99/100	0/100	93/100	0/100	87/100	0/100	29/100	0/100	5/100	0/100
Artificial Potential Fields [105, 104]	100/100	0/100	98/100	0/100	91/100	0/100	46/100	0/100	2/100	0/100	0/100	0/100
Composable Energy Policies	**100/100**	**0/100**	**98/100**	**0/100**	**94/100**	**0/100**	**88/100**	**0/100**	**70/100**	**0/100**	**15/100**	**0/100**

Table 4.2.: Results for 3D GoTo + Obstacle Avoidance Task. First three rows are the results from [240]. We perform the same experiment with the robot hand included in row 4.

4.5.2. Reaching through clutter environments

In the following experimental section, we investigate the performance of CEP for local navigation with a robot manipulator. The experimental section is divided between a simulated experimental section and a real robot experiment. The simulated experiments have been performed to answer the following questions:

Q1: Does energy policy composition increase the probability of reaching the target in a cluttered environment with respect to deterministic composition methods?

Q2: In CEP, the best action is found by stochastic optimization algorithms. How many particles do we need for a 7 dof robot? How many optimization steps?

In the real robot platform, we investigate the performance of CEP under disturbances. The main question we aim to answer is:

Q3: Is CEP able to reactively adapt to unmodelled disturbances, such as human physical interaction, changes in the placing position, or changes in the picking position?

Methods	1 Obstacle		3 Obstacles		Cross		Double Cross		Cage I		Cage II	
	Success	Collide	Success	Collide	Success	Collide	Success	Collide	Success	Collide	Success	Collide
Riemannian Motion Policies [191]	100/100	0/100	89/100	0/100	71/100	0/100	63/100	0/100	11/100	0/100	0/100	0/100
Artificial Potential Fields [105, 104]	100/100	0/100	90/100	0/100	68/100	0/100	21/100	0/100	0/100	0/100	0/100	0/100
Composable Energy Policies	**100/100**	**0/100**	**95/100**	**0/100**	**84/100**	**0/100**	**72/100**	**0/100**	**30/100**	**0/100**	**5/100**	**0/100**

Table 4.3.: Results for 6D GoTo + Obstacle Avoidance Task. First three rows are the results from [240]. We perform the same experiment with the robot hand included in row 4.

Simulated reaching environments

In the following, we present the simulated experiment to reach a certain target avoiding the obstacles. We perform the experiments with a 7 dof Kuka-LWR robot in 6 environments. The environments have been designed to be increasing in difficulty. The first environment has a single obstacle body, while the last one has 62 obstacle bodies. We visualize the considered environments in Figure 4.6. The robot is initialized in a randomized joint configuration and the motion is generated by a set of local motion generators, without additional global path planning algorithms. The episode ends when the robot reaches the target, collides against any obstacle or a certain time is pass. In our experiment, the robot is initialized in 100 randomly chosen initial configurations. We consider a control frequency of 250Hz for this experiment and the episode length is 30 seconds.

Policy setup For this experiment, we consider three possible multi-objective reactive motion generators: RMP [191], APF [105, 104] and CEP. The three methods consider a set of policy components modeled in a set of task spaces. We compute the kinematics transformations in Equation (4.2) with pinocchio [24]. All the policies are local basic policies and there are no long-horizon planning components. The considered policy components are:

- A target reaching policy in the end-effector space;
- $P \times O$ obstacle avoidance policies in a set of cartesian task spaces;
- Joint limits avoidance policy in the configuration space;
- Joint velocity limit policy in the configuration space.

We consider just position reaching policy for the 3D experiment and a full-pose reaching policy for the 6D experiment. For the obstacle avoidance policies, we model a collision body for the manipulator (Figure 4.8). The collision body is composed of 35 spheres with

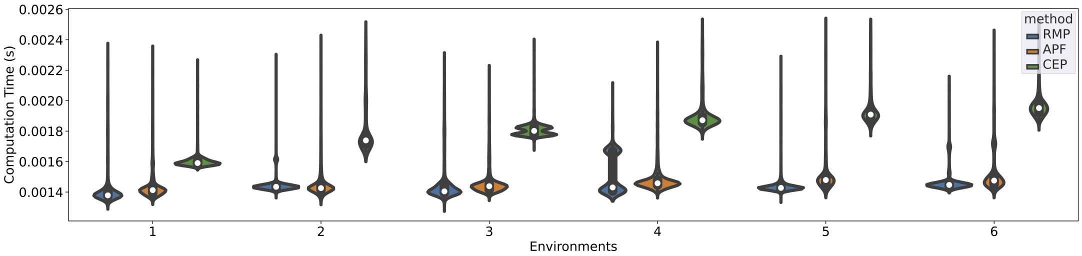


Figure 4.7.: Controller's computation time for the six simulated environments in Figure 4.6. Measured for RMP, APF and CEP.

different radii. We remark that for the most complex environment, we have 62 collision spheres, thus, we have in total $35 \times 62 = 2170$ obstacle avoidance policies. Nevertheless, all of them are computed in batch using tensor multiplication.

Comparative evaluation We initialize the experimental analysis evaluating the success rate and collisions in the 6 environments. We summarize the obtained results for the 3D go-to problem in Table 4.2. The easy environments are easily solved by all the methods. We observe a success rate of almost 100% for the first three environments in all the cases. This result shows that simple scenarios can be easily solved with local reactive controllers and it is not required to solve a global trajectory planning problem. In complex scenarios, CEP performs better than the baselines. We can obtain a 70% success rate in the first cage environment and 15% in the second cage. We hypothesize that this could be related to how the obstacle avoidance policies are modeled. In the chosen baselines, the obstacle avoidance policies apply a repulsive force in a certain robot's link to avoid the obstacles. In highly cluttered environments, where the robot needs to move through narrow passages, these repulsive forces will push the robot far from the obstacles. Then, the robot is not able to get close to the narrow passage and it gets stuck in the entrance. In contrast, in our method, the obstacle avoidance policy defines a uniform distribution for the set of valid actions. This policy is more conservative. Rather than pushing the robot away from the obstacles, our policy penalizes, with very low probability, any action that moves a certain robot link close to the obstacles. Thus, in front of a narrow passage, the obstacle avoidance policy will only inform about those actions that are not valid but will not apply any repulsive force. We suggest that in the most complex environments, integrating the output of a trajectory optimization method could improve the robot's performance.

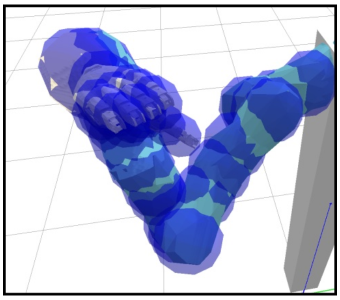
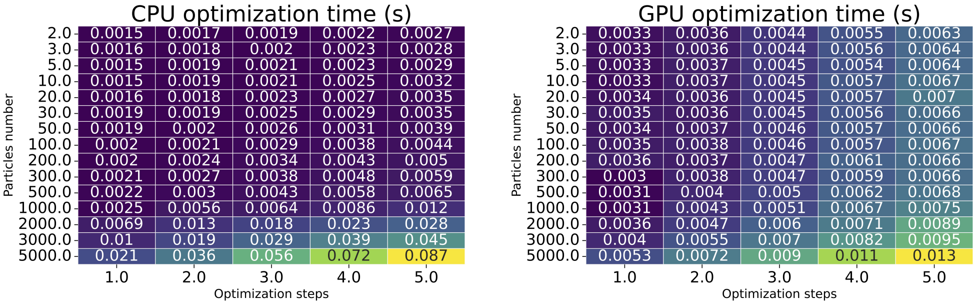


Figure 4.8.: Collision body for the robot manipulator. The collision body is composed of 35 spheres with different radius.

The performance worsens for the 6D go-to problem (Table 4.3). The orientation sets an important constraint in the possible final configurations and reduces the set of trajectories that solves the problem. CEP is able to perform relatively better than the baselines but it got less than 50% success rate in both cage environments, suggesting that in complex scenarios an additional global path planner should be integrated with CEP. It is important to remark, that both the deterministic baselines and our approach were able to properly impose the obstacle avoidance objective. We did not record any single collision in all the trials.

An important consideration for using CEP as reactive motion generators is its computational time. We compare the computation time for CEP with respect to RMP and APF. To properly compare them, we have considered the same amount of policies on the three cases and we used the same kinematics model. All the methods run in a *AMD Ryzen 9 3900* CPU. For CEP, we considered 50 particles and a single optimization step. We show the computational time in Figure 4.7. CEP is remarkably slower than RMP and APF. While previous methods consider an analytical solution for the optimal action, CEP requires solving an optimization problem. To do so, we require to evaluate a set of action particles and update a surrogate distribution Equation (4.17). In our implementation, RMP and APF computes the solution in $0.0015s$ in average, while CEP computes the solution in around $0.002s$. Even if it is slower, we run CEP to 500Hz which is enough for reactive motion generation. Additionally, we remark that our implementations are not optimized and are running in an interpreted language. Thus, we expect faster computation times if the code is optimized and written in a compiled language. An additional point is related to the different computation times for the different environments. Our method requires an average computation time of $0.0016s$ for the first environment and $0.002s$ for the last one. The difference in computation is based on the number of collision spheres. In our work, collision avoidance is evaluated by computing the projected acceleration of a set of obstacle spheres in the robot with respect to a set of obstacles in the environment. The robot is composed of 35 obstacle spheres, while the environment varies from 1 to 62. To be computationally efficient, we apply tensor multiplication and compute the projected accelerations in parallel. Given N possible acceleration candidates in the configuration space, the number of projected accelerations are $N \times O \times 35$, where O is the number of obstacles in the environment.



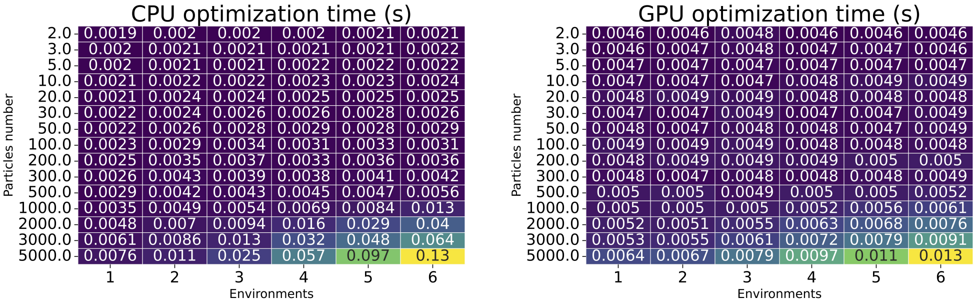
CPU optimization time (s)

	1.0	2.0	3.0	4.0	5.0
2.0	0.0015	0.0017	0.0019	0.0022	0.0027
3.0	0.0016	0.0018	0.002	0.0023	0.0028
5.0	0.0015	0.0019	0.0021	0.0023	0.0029
10.0	0.0015	0.0019	0.0021	0.0025	0.0032
20.0	0.0016	0.0018	0.0023	0.0027	0.0035
30.0	0.0019	0.0019	0.0025	0.0029	0.0035
50.0	0.0019	0.002	0.0026	0.0031	0.0039
100.0	0.002	0.0021	0.0029	0.0038	0.0044
200.0	0.002	0.0024	0.0034	0.0043	0.005
300.0	0.0021	0.0027	0.0038	0.0048	0.0059
500.0	0.0022	0.003	0.0043	0.0058	0.0065
1000.0	0.0025	0.0056	0.0064	0.0086	0.012
2000.0	0.0069	0.013	0.018	0.023	0.028
3000.0	0.01	0.019	0.029	0.039	0.045
5000.0	0.021	0.036	0.056	0.072	0.087

GPU optimization time (s)

	1.0	2.0	3.0	4.0	5.0
2.0	0.0033	0.0036	0.0044	0.0055	0.0063
3.0	0.0033	0.0036	0.0044	0.0056	0.0064
5.0	0.0033	0.0037	0.0045	0.0054	0.0064
10.0	0.0033	0.0037	0.0045	0.0057	0.0067
20.0	0.0034	0.0036	0.0045	0.0057	0.007
30.0	0.0035	0.0036	0.0045	0.0056	0.0066
50.0	0.0034	0.0037	0.0046	0.0057	0.0066
100.0	0.0035	0.0038	0.0046	0.0057	0.0067
200.0	0.0036	0.0037	0.0047	0.0061	0.0066
300.0	0.003	0.0038	0.0047	0.0059	0.0066
500.0	0.0031	0.004	0.005	0.0062	0.0068
1000.0	0.0031	0.0043	0.0051	0.0067	0.0075
2000.0	0.0036	0.0047	0.006	0.0071	0.0089
3000.0	0.004	0.0055	0.007	0.0082	0.0095
5000.0	0.0053	0.0072	0.009	0.011	0.013

Table 4.4.: CEP computation time for CPU and GPU. We consider the average computation time for 1-5 optimization steps.



CPU optimization time (s)

	1	2	3	4	5	6
2.0	0.0019	0.002	0.002	0.002	0.0021	0.0021
3.0	0.002	0.0021	0.0021	0.0021	0.0021	0.0022
5.0	0.002	0.0021	0.0021	0.0022	0.0022	0.0022
10.0	0.0021	0.0022	0.0022	0.0023	0.0023	0.0024
20.0	0.0021	0.0024	0.0024	0.0025	0.0025	0.0025
30.0	0.0022	0.0024	0.0026	0.0026	0.0028	0.0026
50.0	0.0022	0.0026	0.0028	0.0029	0.0028	0.0029
100.0	0.0023	0.0029	0.0034	0.0031	0.0033	0.0031
200.0	0.0025	0.0035	0.0037	0.0033	0.0036	0.0036
300.0	0.0026	0.0043	0.0039	0.0038	0.0041	0.0042
500.0	0.0029	0.0042	0.0043	0.0045	0.0047	0.0056
1000.0	0.0035	0.0049	0.0054	0.0069	0.0084	0.013
2000.0	0.0048	0.007	0.0094	0.016	0.029	0.04
3000.0	0.0061	0.0086	0.013	0.032	0.048	0.064
5000.0	0.0076	0.011	0.025	0.057	0.097	0.13

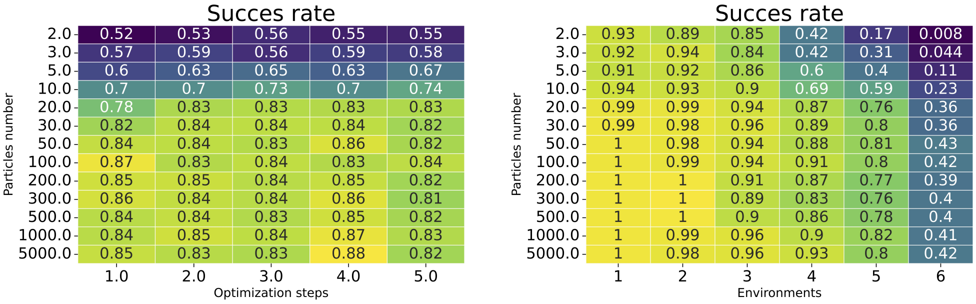
GPU optimization time (s)

	1	2	3	4	5	6
2.0	0.0046	0.0046	0.0048	0.0046	0.0046	0.0046
3.0	0.0046	0.0047	0.0048	0.0047	0.0047	0.0046
5.0	0.0047	0.0047	0.0047	0.0047	0.0047	0.0047
10.0	0.0047	0.0047	0.0047	0.0048	0.0049	0.0049
20.0	0.0048	0.0049	0.0049	0.0048	0.0048	0.0048
30.0	0.0047	0.0047	0.0049	0.0047	0.0047	0.0049
50.0	0.0048	0.0047	0.0048	0.0048	0.0047	0.0049
100.0	0.0049	0.0049	0.0049	0.0048	0.0048	0.0048
200.0	0.0048	0.0049	0.0049	0.0049	0.005	0.005
300.0	0.0048	0.0047	0.0048	0.0048	0.0048	0.0049
500.0	0.005	0.005	0.0049	0.005	0.005	0.0052
1000.0	0.005	0.005	0.005	0.0052	0.0056	0.0061
2000.0	0.0052	0.0051	0.0055	0.0063	0.0068	0.0076
3000.0	0.0053	0.0055	0.0061	0.0072	0.0079	0.0091
5000.0	0.0064	0.0067	0.0079	0.0097	0.011	0.013

Table 4.5.: CEP computation time for CPU and GPU. We show the variation of the mean computation time for the six environments

Given the number of collision spheres in the environment varies, the size of the tensor computing the projected accelerations also changes and thus, the required time for the tensor multiplication.

Conclusion 1 We have observed that the energy model flexibility provided by CEP improves the success rate with respect to previous methods. By choosing an appropriate energy policy, we can improve the cooperation between all the components and solve all the tasks jointly in a more successful way. Nevertheless, CEP requires solving an optimization problem by stochastic optimization methods, while previous methods consider an analytical solution. The stochastic optimization might reduce the computational efficiency of CEP and it will require more time to find a solution with respect to the analytic methods.



Succes rate

Particles number	1.0	2.0	3.0	4.0	5.0
2.0	0.52	0.53	0.56	0.55	0.55
3.0	0.57	0.59	0.56	0.59	0.58
5.0	0.6	0.63	0.65	0.63	0.67
10.0	0.7	0.7	0.73	0.7	0.74
20.0	0.78	0.83	0.83	0.83	0.83
30.0	0.82	0.84	0.84	0.84	0.82
50.0	0.84	0.84	0.83	0.86	0.82
100.0	0.87	0.83	0.84	0.83	0.84
200.0	0.85	0.85	0.84	0.85	0.82
300.0	0.86	0.84	0.84	0.86	0.81
500.0	0.84	0.84	0.83	0.85	0.82
1000.0	0.84	0.85	0.84	0.87	0.83
5000.0	0.85	0.83	0.83	0.88	0.82

Optimization steps

Succes rate

Particles number	1	2	3	4	5	6
2.0	0.93	0.89	0.85	0.42	0.17	0.008
3.0	0.92	0.94	0.84	0.42	0.31	0.044
5.0	0.91	0.92	0.86	0.6	0.4	0.11
10.0	0.94	0.93	0.9	0.69	0.59	0.23
20.0	0.99	0.99	0.94	0.87	0.76	0.36
30.0	0.99	0.98	0.96	0.89	0.8	0.36
50.0	1	0.98	0.94	0.88	0.81	0.43
100.0	1	0.99	0.94	0.91	0.8	0.42
200.0	1	1	0.91	0.87	0.77	0.39
300.0	1	1	0.89	0.83	0.76	0.4
500.0	1	1	0.9	0.86	0.78	0.4
1000.0	1	0.99	0.96	0.9	0.82	0.41
5000.0	1	0.98	0.96	0.93	0.8	0.42

Environments

Table 4.6.: CEP succes rate for reaching a target while avoiding obstacles. Left: Mean success rate for 1-5 optimization steps. Right: Mean success rate for the six environments.

Ablation study CEP finds the optimal action by stochastic optimization. For I optimization steps, we sample N possible actions from the sampling distribution, we evaluate the objective function for each sample, and update the sampling distributions (Algorithm 6). The number of particles and the number of optimization steps will directly influence the performance of the robot. Also, the required computation time will vary depending on the number of samples and optimization steps. In the following experiment, we investigate the number of optimization steps and particles required for the reaching tasks represented in Figure 4.6. Even if the obtained solutions are specific for the chosen experiments, this ablation study is a relevant tool to estimate the required optimization parameters for similar experiments.

The experiments are performed in a *AMD Ryzen 3900* CPU and a *Nvidia GeForce RTX 2800* GPU. We consider both to investigate which hardware is more convenient for our problem. We present the obtained computational performance in Table 4.4 and Table 4.5. With few samples (< 500) the CPU is faster computationally than the GPU. Nevertheless, when the number of particles augment, the GPU outperformed the optimization time of the CPU. We can also observe that the required computation time linearly grows with the number of optimization steps in both CPU and GPU cases. From Table 4.5, we can observe that the computation effort remains constant for all the environments as long as the number of particles is small. Nevertheless, for a high amount of particles the computation frequency decays from the first environment to the sixth environment. For example, using a GPU with 5000 particles, we have an average computation time of 0.0064s (156Hz) for the first environment and 0.013s (77Hz) for the sixth environment. This change in the computation is directly connected with our obstacle avoidance energy. As previously introduced, we compute a tensor of $N \times O \times 35$ for the obstacle avoidance. For the simplest scenario, we have 1 obstacles sphere and for the most complex scenario, we have 62 spheres. For the case of $N = 5000$, in the simplest environment, we deal with a tensor of length $1.7e5$, while in the most complex environment, we require to compute a tensor of length $1.e7$.

In the Table 4.6, we introduce the results for an ablation study on the success rate. We investigate the required amount of particles and optimization steps to reach the targets without colliding. For the experiment, we execute CEP 100 times on each environment. We perform the same experiment for a different number of particles and a different number of optimization steps. The robot is initialized in a random configuration and reactively navigates to reach the target. We show the mean success rate for all the environments in Table 4.6 (a) and the mean success rate for different number of optimization steps in Table 4.6 (b). For very few particles ($B = 2$) the robot performs poorly achieving a mean success rate of 0.54. Nevertheless, we can observe that it can solve properly the easiest environments and it has a success rate of 0.008 for the most complex one. We can also observe that for the cases in which few particles are used ($5 - 20$), the success rate improves if we use consider more optimization steps. After 50 particles, the success rate arrives in a plateau and an additional number of particles does not increase the success rate. We observe that while increasing optimization steps might have a direct impact with few particles (< 30) when considering a high amount of particles (> 30) a single optimization step is enough for solving the task at the maximum affordable success rate. We can also observe a clear pattern in the environments. While the simplest environments are solved with an almost $1.$ success rate, the performance decays up to a 0.4 success rate in the most complex environment. Due to the increase of obstacles, the robot gets stuck in local minima more often in complex environments with respect to simple ones. In these situations, a learned policy or a path planning algorithm could help the CEP, providing global guidance, while CEP solves the local reactive problem.

We remark that the variability in the success rate might be directly influenced by the stochasticity in the initial configuration. There are no predefined initial configurations, but rather the robot is initialized in arbitrary configurations.

Conclusion 2 From the Table 4.4 and Table 4.5, we conclude that for our experiments, if less than 500 particles are enough to solve the problem, we will choose the CPU while for more demanding tasks a GPU should be considered. From Table 4.6, we conclude that for the chosen experiments, 50 particles and one optimization step are enough to solve the tasks and we do not see further improvements when increasing the number of particles or the optimization steps. As we can observe, the computation times for 50 particles are sufficiently small to have control frequencies around $500Hz$. Nevertheless, we can

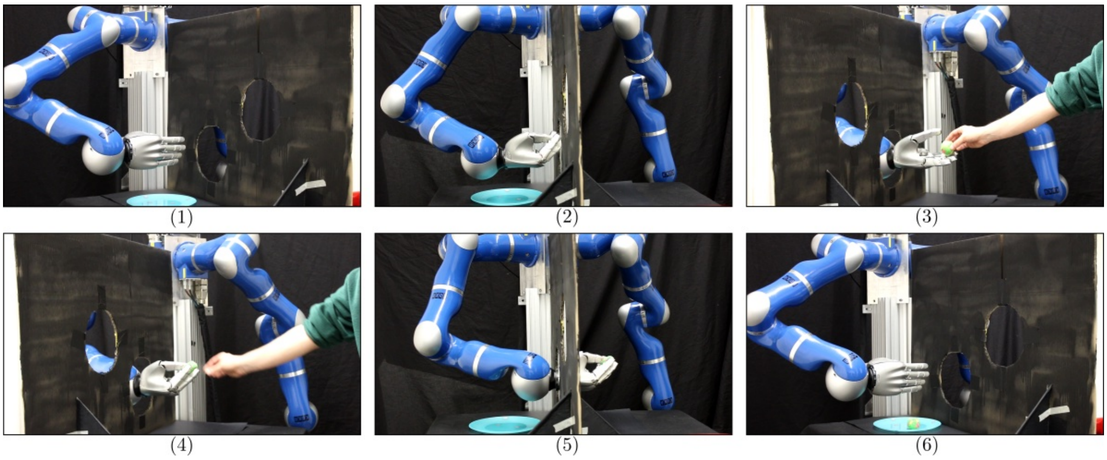


Figure 4.9.: A visual representation of the pick and place task in a real robot environment. The robot is initialized in the left side. It should reach to the other side through the holes to pick the object. Then, move back to the left side to place the object.

observe how the computation requirements grow the more complex the environment is. We can estimate that for environments that are even more cluttered than ours, we might benefit from learning Signed Distance Functions (SDF) [168] to reduce the computational requirements in the obstacle avoidance policy.

Pick-and-Place in cluttered environment

In the following experiment, we evaluate the performance of CEP in a real robot scenario. We consider the problem of picking and place in a cluttered environment. To pick the object, the robot is required to navigate its hand through a narrow hole, grasp the object and navigate out of the narrow hole to leave the object in a plate. We remark there is no global path planning or trajectory optimization and the robot reactively computes the desired accelerations with a local controller. We present a visualization of the task in Figure 4.9. We have modeled the obstacle wall by 67 obstacle spheres (Figure 4.11). With this experiment, we aim to investigate the performance of CEP in complex environments as real-world human-robot interaction environments. We additionally evaluate the performance of RMP as baseline.

To control the robot, we use a CEP with a similar architecture to the simulated experiments:

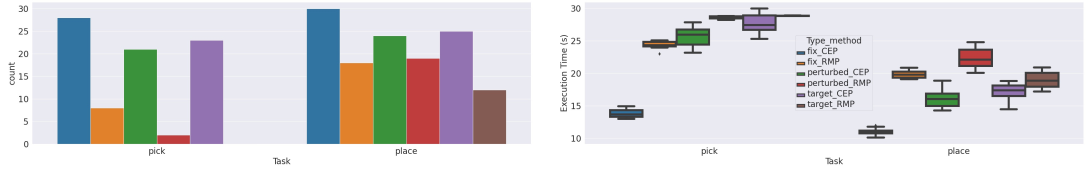


Figure 4.10.: Left: Number of successful picks and places. Right: boxplot showing the execution time to solve the pick and place tasks. We evaluate the performance for (i) fix targets and no disturbances, (ii) under human physical disturbances and, (iii) under target modifications.

- A target reaching policy in the end-effector space
- $P \times O$ obstacle avoidance policies in a set of cartesian task spaces
- Joint limits avoidance policy in the configuration space
- Joint velocity limit policy in the configuration space

The robot is controlled with 50 particles and a single optimization step.

To investigate the performance of CEP in the pick and place task, we measure the success rate and the execution time for picking and placing under 3 conditions. First, we assume there is a fixed target to pick and place and no human perturbing the robot. Second, the human applies physical perturbations to the robot and changes its position, and third, we track the human hand and dish to leave the object. Then, the picking and placing targets are changed online. We run the pick and place routine 30 times for every case. We present the results in Figure 4.10.

On average, the robot is able to solve the pick and placing task under the three conditions with more than 75% success rate. We can observe that the robot is performing slightly better when no human perturbations or target modifications happened. The failure cases are related to the robot getting stuck in an unrecoverable state. For some initial configurations, the robot might enter wrongly in the not correct hole. Given CEP is myopic, it lacks any notion on how to escape from the hole and it gets stuck. When the human interacts with the robot it might move the robot to an unrecoverable state more often and then, the robot is not able to recover and gets stuck. In the target modification case, we reactively change the picking position and the placing position. During the picking, we find the robot used to get stuck if the picking point is too far from the holes. Once the robot moves its arm inside the hole, the maneuverability decays. This might result in the robot getting in some configurations in which it does not know how to get out of the hole.

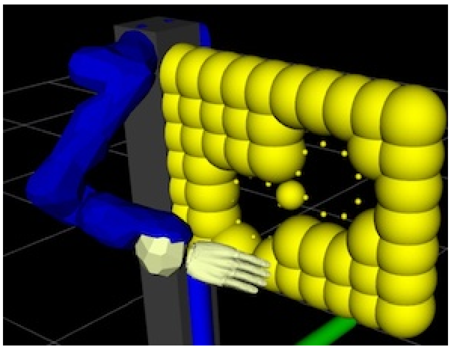


Figure 4.11.: A visual representation of the sphere-based collision body for the real robot experiments wall. The collision body is composed of 67 spheres with different radius.

On average, the performance is better for placing than for picking. In particular, when the placing target is modified, there is only the target reaching component having a big influence on the robot's motion. Nevertheless, when picking, the robot needs to trade-off between the obstacle avoidance component and the picking target reaching component. Due to this, the performance is better in the placing. We observe that CEP outperformed RMP in all situations. Similarly to simulated experiments, we consider that our proposed collision avoidance energy policy allows a better integration with the attractor policy in contrast with the repulsive collision avoidance policy from RMP. We consider that there are two easy fixes to improve the performance of the robot in the situations it gets stuck. First, we can easily combine longer horizon planning methods with CEP, to escape local minima. Additionally, we can learn specific energy policies that guide the robot through narrow passages. These energy policies will lead the robot's behavior when the robot is in a difficult passage, but won't influence the performance of the robot when is far from the narrow passages.

We can also observe the increase in execution time under human perturbances and target modifications for both picking and placing tasks. Due to the stochasticity, the human injects in the robot's motion, it requires additional time to solve the task. In the case of human perturbances, the injected noise is by physically stopping and moving the robot around. Additionally, the robot might be set in an uncomfortable configuration and it requires additional time to recover and solve the task. In the case of the target modifications, the additional execution time is usually due to the human lack of steadiness and sensor errors. The robot tries to grasp the object once a certain distance threshold to the object is passed. Given the sensor disturbances and the human's lack of steadiness, the robot might require additional time to reduce the threshold distance and pick the object. We can observe, that the execution times are bigger for the picking case rather than the placing task. Due to the lack of maneuverability in the picking task, the robot usually requires way more time to solve the task. Nevertheless, one is far from the obstacles, the robot's possible movements increases and it can solve the task faster.

Conclusion 3 We evaluated the validity of CEP as a reactive motion generator in a real system. We have observed that the robot is able to reactively adapt to unmodelled perturbances such as human physical disturbances or online target modifications and still solve the pick and place task while moving through a narrow hole. Nevertheless, we observe the locality of CEP in some configurations. The robot might get stuck in a local optimum trying to enter through the hole. Due to this, we suggest integrating learning components or path planning components to be able to consider longer horizon information and resolve the local minima easier.

4.5.3. Learning with structured policies

In this experimental section, we evaluate the performance of CEP as structured policy in a reinforcement learning problem. We perform the experiments to answer the following questions

Q1: Can we improve the learning performance of the reinforcement learning problem by integrating guiding priors through CEP?

Q2: Can we reduce the number of collisions while learning by integrating obstacle avoidance priors through CEP?

Learning how to hit a puck

We consider the problem of learning how to hit a puck and putting it in a certain target position (Figure 4.12). We consider this task a good experiment to investigate the benefits of integrating priors for learning. A desirable policy should learn to hit the puck without colliding against the table. Nevertheless, given that the puck is close to the table, it is hard to find a policy that weights properly both objectives. We investigate the benefit of CEP as a structured policy to deal with such situations. Additionally, if we consider the whole workspace of the robot, there are very few regions in the state-action space that make the robot move the puck. Most of the possible state-action pairs won't influence the position of the puck. Thus a prior guiding the robot to the puck might be very helpful to explore in more informative regions.

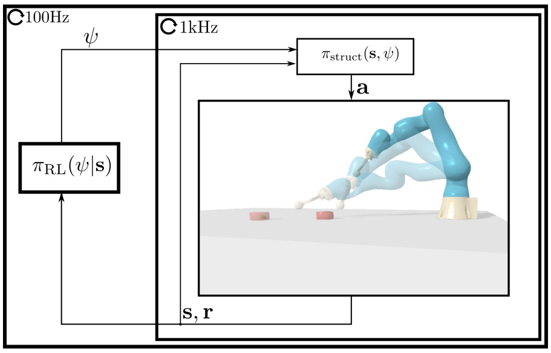

100Hz
1kHz
ψ
$\pi_{\text{struct}}(\mathbf{s}, \psi)$
$\mathbf{a}$
$\pi_{\text{RL}}(\psi|\mathbf{s})$
$\mathbf{s}, \mathbf{r}$

Figure 4.12.: A block diagram of a reinforcement learning problem with a structured policy. The RL agent π_{RL} and the structured policy π_{struct} might run to different control frequencies. Given the current state $\boldsymbol{s}$, the RL policy samples a parameter vector $\boldsymbol{\psi}$. This parameter vector is input in the structured policy and the control action is computed $\boldsymbol{a}$.

We use a 7 dof LBR-IIWA robot. A visual representation of the task can be found at Figure 4.12. The reinforcement learning agent π_{RL} receives as input the state $\boldsymbol{s}$. The state $\boldsymbol{s} \in \mathbb{R}^{18}$ is represented by the end-effector's cartesian position $\boldsymbol{x}_{\text{ee}}$, puck's position $\boldsymbol{x}_{\text{puck}}$, their relative position $r_{\text{p-ee}} = \boldsymbol{x}_{\text{ee}} - \boldsymbol{x}_{\text{puck}}$, the puck's velocity $\boldsymbol{v}_{\text{puck}}$, the end effector's velocity $\boldsymbol{v}_{\text{ee}}$ and the target position $\boldsymbol{x}_{\text{target}}$. The output of the reinforcement learning agent is the parameter $\boldsymbol{\psi}$. For our experiment, the output $\boldsymbol{\psi} \in \mathbb{R}^3$ represents the desired task space cartesian velocity in the end-effector. The robot is always initialized with the same joint configuration and each episode last 300 steps (*it lasts 300 steps for the reinforcement learning agent, but 3000 steps for the structured policy* $\pi_{\textit{struct}}$).

Policy Setup The structured policy receives the reinforcement learning agent's output, $\boldsymbol{\psi}$, and the state $\boldsymbol{s}$ and computes the desired configuration space acceleration as action $\boldsymbol{a} = \ddot{\boldsymbol{q}}$.

We consider two baselines as structured policies: direct operational space control [104] and residual operational space control [218, 93]. The structured policy for direct operational space control is modelled by

$$\pi_{\text{struct}}(\boldsymbol{s}, \boldsymbol{\psi}) = \boldsymbol{J}^{\dagger} \boldsymbol{K} (\boldsymbol{\psi} - \dot{\boldsymbol{q}}), \tag{4.36}$$

with $\boldsymbol{J}^{\dagger}$ the Jacobian pseudoinverse of the forward kinematics to the end-effector, $\dot{\boldsymbol{q}}$ the current robot joint configuration and $\boldsymbol{K}$ a damping gain. The controller defines a velocity error correction in the task space. Then, the desired task space acceleration is map to the configuration space by the Jacobian pseudoinverse. The residual operational space controller is modelled by

$$\begin{aligned}\pi_{\text{struct}}(\boldsymbol{s}, \boldsymbol{\psi}) &= \boldsymbol{J}^{\dagger}\boldsymbol{K}(\dot{\boldsymbol{q}}^{*} - \dot{\boldsymbol{q}}) \\ \dot{\boldsymbol{q}}^{*} &= \boldsymbol{\psi} + \pi_{g}(\boldsymbol{s}),\end{aligned} \tag{4.37}$$

with $\pi_g(\boldsymbol{s})$ the guiding policy. Residual task space control applies a similar approach to direct operational space control. Nevertheless, the desired task space velocity is defined by the linear sum of $\boldsymbol{\psi}$ and $\pi_g(\boldsymbol{s})$. $\pi_g(\boldsymbol{s})$ is a guiding policy. In our experiments, we model $\pi_g(\boldsymbol{s})$ as a CEP with an obstacle avoidance energy to the table and a puck attractor energy. Thus, the desired velocity is represented as the linear sum of the reinforcement learning action and the CEP action.

We compare these baselines with respect to using CEP as the structured policy (Section 4.4). Our model policy is modelled by

$$\pi_{\text{struct}}(\boldsymbol{s}, \boldsymbol{\psi}) = \arg\max_{\ddot{\boldsymbol{q}}} \log\left(\prod\nolimits_{k} \pi_k(\ddot{\boldsymbol{q}}|\boldsymbol{s}, \boldsymbol{\psi})\right). \tag{4.38}$$

In contrast with the residual policy, which linearly combines the output of the reinforcement learning agent and the CEP, in our case, we input the action of the reinforcement learning agent $\boldsymbol{\psi}$ as an additional parameter to condition the energy policies. Then, the optimal action is computed between the conditioned energy policies and the priors. The CEP is built by three energy policy components:

- A target (puck position) reaching energy in the end-effector;
- An obstacle avoidance energy in the end-effector (to avoid collisions against the table);
- $\boldsymbol{\psi}$ parameterized velocity tracking energy in the end-effector.

The chosen residual policy combines the output of the reinforcement learning agent $\boldsymbol{\psi}$ and the output of a defined CEP, π_g linearly. In contrast, the CEP policy takes as input parameter the reinforcement learning agent's output $\boldsymbol{\psi}$ and solves the maximization problem combining a $\boldsymbol{\psi}$ parameterized policy and two defined policies.

Both baselines have been widely applied as structured policies in reinforcement learning [89, 93, 123, 211, 58]. We choose an operational space control baseline to investigate the benefit of the target reaching bias terms in the CEP. Given both CEP and residual policy uses an attractor to the target, we aim to investigate if it provides an additional benefit with respect to not considering an attractor bias in the policy. We choose residual control to investigate the benefit of the obstacle avoidance prior in CEP. Even if both consider an obstacle avoidance prior, the prior and the reinforcement learning action are integrated differently. We aim to investigate the benefits and perks of imposing obstacle avoidance in our method concerning the residual control method.

Problem Setup To properly investigate the benefit of CEP, we extend the previous work in [240] with two additional MDPs. The three MDPs consider the same transition dynamics model, but they differ in the reward function. The three rewards contain a reward signal that defines the task to solve (move the puck to the target). The three of them differ in the inductive biases we additionally integrate into the reward function. We aim to study the influence of reward shaping with respect to the influence of structured policies to improve the learning performance.

The first reward is composed of the distance between the end-effector and the puck and distance between the puck and the target:

$$r_1 = -d_{\text{ee-puck}} - d_{\text{puck-target}}.$$

The distance between the end-effector and the puck is an inductive bias that helps the agent to find a policy that hits the puck. A similar inductive bias is integrated into the residual and CEP policies with the attractor policy.

The second reward function additionally considers a negative terminal cost if the robot hits the table, $r_T = -100$.

$$r_2 = -d_{\text{ee-puck}} - d_{\text{puck-target}} + r_T(\boldsymbol{s}_T).$$

The negative terminal cost is an inductive bias that pushes the robot to avoid the table. If the robot collides against the table, we stop the episode and add the terminal cost.

Finally, the last reward function (the one in [240]), considers only the reward distance between the puck and the target

$$r_3 = -d_{\text{puck-target}} + r_T(\boldsymbol{s}_T).$$

Eliminating the attractor to the puck, the direct operational space controller lacks any source of bias to approximate to the puck. We aim to study if the attractor inductive bias might have any influence on our learning performance. We claim that CEP can be used with any arbitrary reinforcement learning algorithm. To investigate its performance, we have conducted the experiments with several deep reinforcement learning algorithms (PPO [213], SAC [74], DDPG [130], TD3 [57]) implemented in MushroomRL [32].

Comparative evaluation We investigate the learning performance of the three structured policies in terms of the accumulated reward and accumulated collisions against the table per epoch. We present the obtained results in Figure 4.13.

We start evaluating the results for the first reward (Figure 4.13 top row). Observing the discounted reward, we can see that the three controllers can achieve a similar performance (direct operational space control is slightly worse). While CEP and residual policies initial return is close to the prior, the direct operational space control starts with a worse policy. Nevertheless, given that we have added a distance reward to the puck, the direct controller is able to get close to CEP and residual in a few episodes. The performance is pretty different if we observe the collisions. The collision avoidance prior allows CEP to explore without a single collision. In contrast, residual and direct controllers collide quite often while exploring. The residual controller has the obstacle avoidance prior encoded in the guiding policy. Nevertheless, we can observe that the prior is not strong enough and the robot collides quite often. In the third column, we present the performance of CEP for different deep reinforcement learning algorithms. As we can observe, the agent is able to solve the problem with any learning algorithm.

In the second experiment, we additionally add a terminal cost if the robot collides with the table. We present the results in Figure 4.13 middle row. We can observe that while CEP maintains a similar learning curve than in the first problem, the learning curve for residual learning and direct control slows down. In the current MDP, the robot not only needs to find a policy to hit the puck but also, avoid collisions against the table. We hypothesize that given CEP is able to impose the table collision properly, it will not explore in state-space regions that lead to a collision. Therefore, the learning agent is completely focused on the problem of hitting the puck. On the contrary, direct and residual learning policies are constantly hitting against the table. Due to this, the reinforcement learning agent requires to learn a policy that both avoids collisions and also hits the puck. This hypothesis match with the collisions plot. Both direct and residual policies have multiple collisions in the first episodes. Nevertheless, during the learning, they find policies that are able to avoid them. In the case of CEP, the obstacle avoidance prior is sufficiently strong to have an obstacle-free learning process. Similar to the first problem, all the reinforcement learning algorithms can solve the problem with similar performances.

Finally, in the third experiment, we eliminate from the reward function the prior that guides the robot close to the puck. In our previous experiments, the guiding prior is included in both the reward signal and the structured policies. We aim to investigate the learning performance when we impose the guiding attractor only in the structured policies. The obtained results are presented in Figure 4.13 bottom row. The performance of our method remains good even if the inductive bias is eliminated from the reward. The inductive bias in the policy is enough to find a good behavior to hit the puck. We can observe that also residual policy learning is performing well, similarly to the second environment. Nevertheless, the direct controller is not able to solve the problem and the performance decays from previous experiments. The reward function is highly sparse (the robot receives information only if it hits the puck) and the direct control lacks any guiding inductive bias to be close to the puck. In conclusion, a lack of guided exploration makes the controller not find proper behavior.

Conclusion 4 In this experiment we investigate the influence of integrating inductive biases in a robot reinforcement learning problem. We include two inductive biases: a collision-avoidance bias to encourage the robot not to collide against the table and an attractor bias that encourages the robot to get close to the puck. We investigate the differences between integrating the inductive biases directly in the reward or integrating them in the policy. Additionally, we investigate the differences linearly sum inductive biases (residual controller) or through an optimization problem (composable energy policies). A relevant conclusion is that in contrast with other structured policies, only CEP is able to guarantee a learning without a single collision. the collision avoidance bias in the residual controller does not impose sufficient constraints. On the other hand, the collision avoidance bias in the reward function requires the robot to collide against the table to receive the reward signal to learn not to collide. Thus, we conclude that the CEP approach is a relevant approach to guarantee safer exploration. In contrast, we observe that the attractor bias is properly integrated with the residual approach and also through the reward function. While CEP is able to also integrate this inductive bias, we do not see any major benefit in our approach with respect to others.

4.6. Related work

In the main part of this article, we focus on modeling reactive motion generators. Nevertheless, reactive motion generators have been widely explored for both robot local navigation and robot reinforcement learning. In this section, we want to briefly summarize the existing work on both topics.

Multi-objective reactive motion generation The problem of reactively generating motion for local robot navigation satisfies three conditions: (i) there are multiple objectives that must be jointly satisfied, (ii) usually, these objectives are defined in arbitrary task spaces and (iii) we should be able to compute the solution fast enough to have high control frequencies and be reactive. Two of the earliest solutions to this problem are proposed in [105, 104]. In [105], artificial potential fields method is proposed. This method provides a solution for combining multiple obstacle avoidance objectives as a combination of repulsive potential fields. In [104], the idea of operational space control is defined. Operational space control provides a method for mapping the desired policies defined in the task space to the configuration space in which the robot is controlled.

Inspired by these early efforts, in [191], Riemannian motion policies are introduced. In this work, the problem of properly combining policies defined in different task spaces is addressed. Riemannian motion policies focus on the concept of the metric to properly weight the contribution of each policy in the final action. Later works [28, 127] improve the riemannian motion policies in terms of computational efficiency. There has been a set of alternative works, dealing with different problems in Riemannian motion policies. In [215] pullback bundle dynamical systems are proposed. The work proposes a method to combine multiple policies defined in non-Euclidean spaces. Geometric Fabrics [190, 189, 258] propose to model the policy composition in terms of Finsler geometries. As shown in their work, modelling the problem in terms of Finsler geometries, they can easily guarantee stable behaviors in contrast to Riemannian motion policies.

All of the above methods assume the composition of a set of myopic controllers. In a different direction, some researchers have tried to study the composition of policies that are optimised to solve a longer horizon control problem. In [233], linearly-solvable Markov Decision Processes (LMDP) is presented. As shown in the work, a weighted sum of individually optimal policies was proven to be the optimal control problem solution for a reward function defined as weighted-sum of individual rewards and differing only in the terminal reward. In a similar vein [72, 231] proposes suming a set of optimal Q function. Additionally, in [72] the distance between the optimal Q function and the sum of Q functions is investigated.

Finally, a set of works propose solving the multi-objective reactive motion generation problem by numerical optimization. Dynamic Window Approach (DWA) [55, 246] solves the reactive motion generation for a 2D planar robot in a two steps optimization algorithm. First, the search space of possible actions is reduced given a set of constraints. Then, given an objective function, the optimal action is selected. Model predictive control (MPC) methods [59, 47, 37] consider a non-myopic trajectory optimization problem to find reactively the optimal action satisfying multiple objectives. To reduce the computational requirements, MPC methods initialize the optimization problem with the previously computer solution. These methods usually assume simplified kinematics and dynamics models and quadratic cost functions to be computationally efficient and be reactive.

Our work lays in a middle-ground. We consider a myopic (one-step ahead) optimization problem to find the optimal action similarly to artificial potential fields and Riemannian motion policies. Nevertheless, we do not assume there exist an analytic expression for our solution and rather we solve a numerical optimization problem in every control step as in model predictive control.

Structured policies in robot reinforcement learning Integrating inductive biases into the reinforcement learning problem has been shown to be effective in improving the learning performance of the reinforcement learning agents. These inductive biases are usually integrated into the problem through the reward function (*reward shaping*) or through the policy (*structured policy*). Structured policies modify the action space of the reinforcement learning agent. Rather than sampling actions that directly influence the robot such as torques, the reinforcement learning agent samples actions in a parameter space. These parameters are later inputted into the structured policy and the robot control signal is computed.

Structured policies have a long history in robot reinforcement learning. Operational space controllers [177, 123] transform the action space from the configuration space to the task space. It is shown that the robot is able to learn better policies if the reward function is also defined in the task space. A big set of works have considered applying reinforcement learning in the parameter space of movement primitives such as DMP or handcrafted primitives [155, 35, 176, 9, 34]. Considering a DMP as structured policy guarantees inductive biases such as stability or smoothness in the robot's behavior. Residual Policy Learning approaches [218, 93], model the structured policy by a linear sum of the reinforcement learning action and the output of a guiding policy. Assuming the guiding policy is properly modeled for the task, the reinforcement learning agent is expected to learn the residuals of the guiding policy and improve the performance of the guiding policy.

A different approach is proposed in [127]. Instead of splitting the policy between the reinforcement learning agent and the structured policy, they propose to model the reinforcement learning policy directly as a RMP. The action to apply in the robot is directly sampled from the agent, but the agent model is a parameterized RMP. They claim that both the policy leaves and the task maps can be parameterized and learned.

In contrast with previous works that assume an explicit structured policy; in our work, we propose to model the structured policy via an optimization function. The reinforcement learning agent samples a set of actions that parameterize the energy policies. Then, we apply an optimization problem to find the optimal action. As shown in the experiments, this optimization problem can easily impose hard constraints (through uniform distributions) and guarantee safer learning.

4.7. Discussion

CEP can be viewed as a generalization on RMP where the cost functions are not limited to be quadratic. This generalization provides additional flexibility to model the policy components and find better alternatives to represent the policy components. From a different perspective, CEP can be viewed as a particular case of MPC, in which the control horizon is fixed to a single step. Fixing the control horizon to a single step allows us to reduce the optimization variables and find control actions for a high-dimensional robot and a set of cost functions with a low computational budget. We show empirically, that with a proper choice of energies, one-step ahead optimization can still solve complex cluttered environments without the computational requirements of optimizing over longer horizons as in MPC. Nevertheless, it is important to remark that our approach is myopic and thus, we will never have guarantees of satisfying all the objectives in the long run. To have the guarantees of solving a long-horizon problem, we might instead rely on long-horizon planning methods. Additionally, we embrace a probabilistic interpretation of the multi-objective reactive motion generation problem. Framing the problem in a probabilistic view allow us to build connections between the literature in Bayesian inference [91, 195] or in energy based composition [78, 40] and the literature in reactive motion generation [104, 191]. We consider that this probabilistic interpretation is beneficial to integrate learning components in the reactive motion generation problem. From this view, we can built policy component as maximum likelihood distributions for a given dataset or maximum-entropy policies for a given reward.

Introduced in Section 4.3, RMP can be viewed as the solution of a myopic control as inference problem in which each component is modelled by a normal distribution. A normal distribution assumes that the action distribution is unimodal. There exist an optimal action (the mean) to satisfy a particular objective and the quality of the rest of the actions is measured given a Mahalanobis distance to the optimal one. While framing the problem in terms of normally distributed policy components have some benefits (there exist a close form solution for the optima); the expressivity might be limited for other components. We have shown experimentally, that uniformly distributed policies might be more benefitial to represent obstacle avoidance policies. Nevertheless, a drawback of considering non-quadratic policies is that we lack an analytical solution of our optimization problem and we require to use stochastic optimization algorithms to find the optimal action. Through a set of ablation studies, we have evaluated the computational requirements of our method and find out that with the current hardware, we can achieve control frequencies up to 500Hz with a non optimized code. We expect that with a highly optimized code and with more powerful hardware, we could achieve 1kHz control frequencies.

In our work, we have additionally evaluate the performance of CEP as a structured policy. In reinforcement learning, high dimensional state-action spaces with sparse rewards might require an excessive amount of samples to find a proper behavior. Additionally, the robot might have undesirable collisions while exploring. A common approach to deal with this problem is by exploring in the parameter space of a low-level controller. Through this abstraction we can impose all the desired inductive bias in the low-level controller and explore in manifold generated by the inductive biases. In contrast with most of the structured policies in the literature, in CEP, the structured policy is represented by an optimization problem over a set of parameterized implicit functions. The reinforcement learning agent samples a set of parameters that conditions the objective function to optimize. Through this abstration we can combine inductive bias costs and reinforcement learning conditioned costs and solve an optimization problem that aims to satisfy all the objectives. Through this approach, imposing collision avoidance constraints or guiding bias is simple as we only require to agregate an additional energy function to the objective. It also provides a modular learning as the reinforcement learning conditioned policy can be completely independent from the inductive bias policies.

4.8. Conclusion and future work

We have introduced a probabilistic approach for multi-objective reactive motion generation. We have shown theoretically the relations between CEP and RMP and observe that the increase in flexibility when modelling the policies could improve the composability performance with respect to RMP or APF methods. CEP is a general framework that allows the composition of multiple sources policies defined in arbitrary task spaces.

When integrated as a structured policy for reinforcement learning, we have shown that CEP provides a novel learning structure. The reinforcement learning agent samples a set of parameters that are integrated as conditioning elements of an objective function and the optimal action is computed by solving this low level optimization problem. This bi-level optimization problem allows the integration of safety constraints through uniformly distributed inductive biases or guiding inductive biases without changing the policy model, but simply including additional components to the objective function. We have shown experimentally that the implicit structured approach provides higher safety guarantees with respect to explicit structured policies.

A missing element in the current work is observing the performance of CEP with data-driven policy components. All the components we have considered are analitically computed policies or reinforcement learning based learned policies. We consider as future work, exploring the composition of multiple policies learned by imitation learning. Combining CEP architecture with imitation learning allows the composition of multiple demonstrations given in different task spaces. This would open the possibility of composing expert demonstrations from multiple sources and in different state-action spaces in a single policy. We expect this approach to increase the generalization and modularity properties in robot learning.

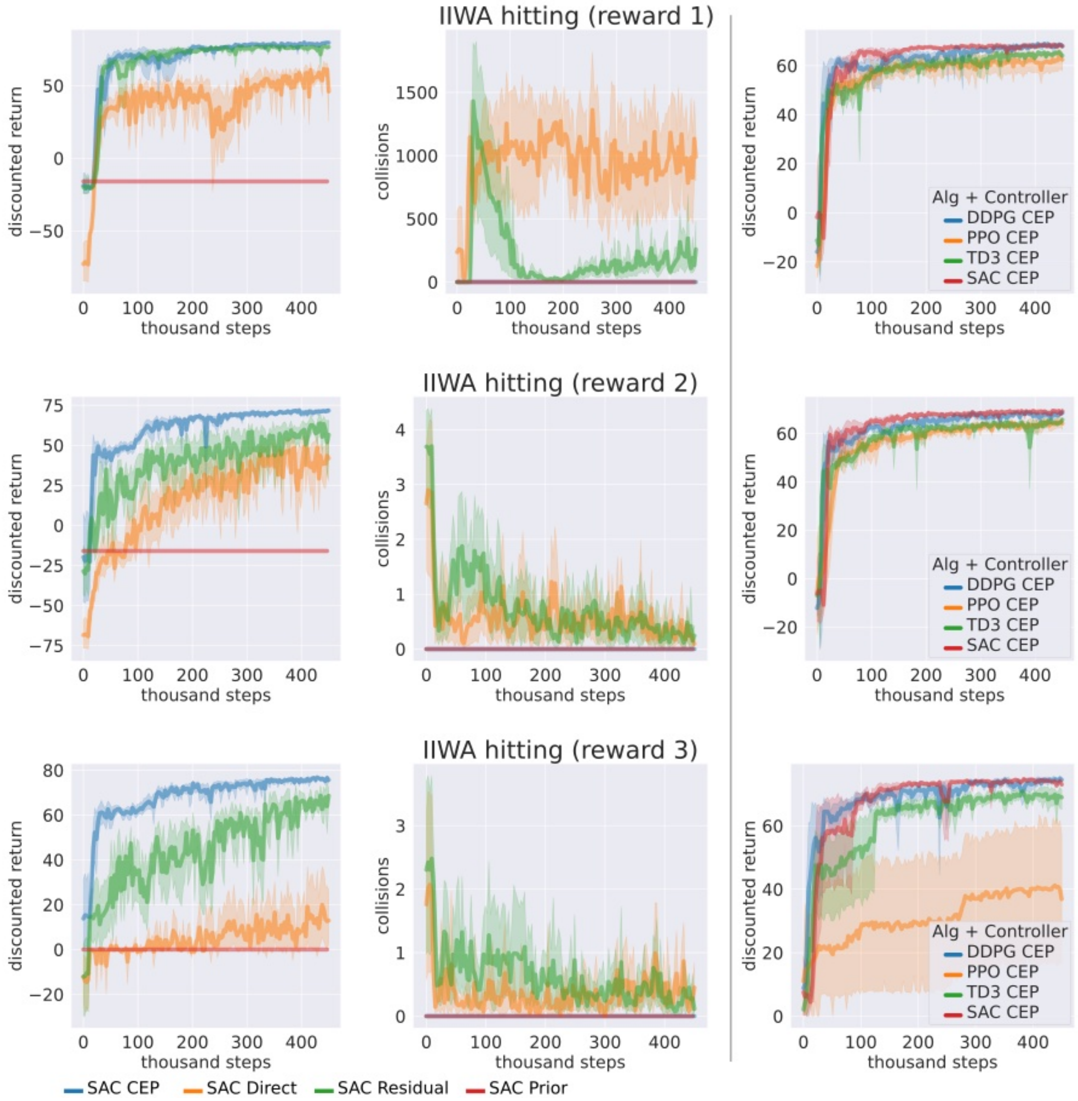


Figure 4.13.: Obtained results for the hitting a puck experiment. Column 1 and 2 present a comparison between different structured policies. Column 3 presents a comparison between different reinforcement learning algorithms.

5. Diffusion Models on SE(3) for Motion Planning

In this chapter, we introduce SE(3)-DiffusionFields, a novel model that represents diffusion models in the Lie group SE(3). The Lie group SE(3) is an important manifold for robotics, as it is the space in which we can represent 6D positions and orientations. In our work, we use our proposed model to learn a diffusion model to generate 6DoF grasp poses for arbitrary objects. As shown in the experimental section, our proposed model outperforms state-of-the-art generative models for grasp pose generation.

Due to the inherent implicit nature of the diffusion models, we can integrate diffusion models into optimization problems as cost gradients. Additionally, given the model is trained with denoising score matching, we have the guarantee that we have informative gradient in the whole space, being an useful signal for iterative optimization algorithms. In this work, we explore the problem of using diffusion models as cost or cost gradients into multi-objective optimization problems. We show that we can integrate the proposed grasp pose diffusion model as an additional objective into a motion planning problem, giving rise to a novel framework for joint grasp and motion optimization without needing to decouple grasp selection from trajectory generation. As shown in the experimental section, solving the problem of grasp and motion planning in a single optimization problem reduces the required number of samples to find a valid solution.

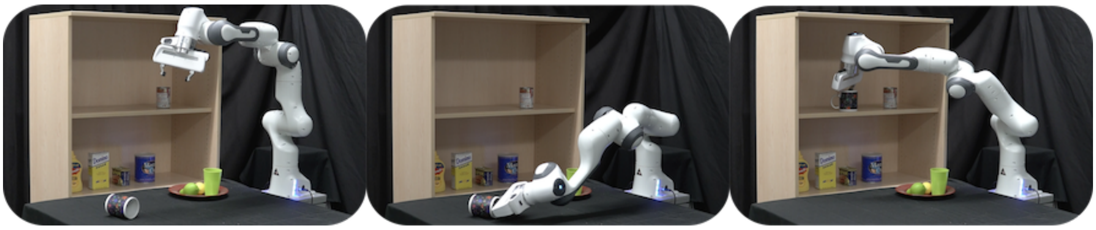


Figure 5.1.: Pick and place task in which the robot has to pick a mug and move it to the target pose (in the shelves) without colliding. We exploit diffusion models for jointly optimizing both grasp and motion and show the successful trajectory from left to right.

5.1. Introduction

Autonomous robot manipulation tasks usually involve complex actions requiring a set of sequential or recurring subtasks to be achieved while satisfying certain constraints, thus, casting robot manipulation into a multi-objective motion optimization problem [188, 97, 212]. Let us consider the pick-and-place task in Figure 5.1, for which the motion optimization should consider the possible set of grasping and placing poses, the trajectories' smoothness, collision avoidance with the environment, and the robot's joint limits. While some objectives are easy to model (e.g., joint limits, smoothness), others (e.g., collision avoidance, grasp pose selection) are more expensive to model and are therefore commonly approximated by learning-based approaches [184, 162, 152, 242, 219].

Data-driven models are usually integrated into motion optimization either as sampling functions (explicit generators) [152, 107], or cost functions (scalar fields) [243, 184]. When facing multi-objective optimization scenarios, the explicit generators do not allow a direct composition with other objectives, requiring two or even more separate phases during optimization [156]. Looking back at the example of Figure 5.1, a common practice is to learn a grasp generator as an explicit model, sample top-k grasps, and then find the trajectory that, initialized by a grasp candidate, solves the task with a minimum cost. Given the grasp sampling is decoupled from the trajectory planning, it might happen the sampled grasps to be unfeasible for the problem, leading to an unsolvable trajectory optimization problem. On the other hand, learned scalar fields represent task-specific costs that can be combined with other learned or heuristic cost functions to form a single objective function for a joint optimization process. However, these cost functions are often learned through *cross-entropy optimization* [152, 137] *or contrastive divergence* [50, 243], creating hard discriminative regions in the learned model that *lead to large plateaus in the* *learned field with zero or noisy slope regions* [7, 148], thereby making them unsuitable for pure gradient-based optimization. Thus, it is a common strategy to rely on task-specific samplers that first generate samples close to low-cost regions before optimizing [152, 137].

In this work, we propose learning *smooth* data-driven cost functions, drawing inspiration from state-of-the-art diffusion generative models [227, 138, 83, 16, 66]. By *smoothness*, we refer to the cost function exposing informative gradients in the entire space. We propose learning these smooth cost functions in the SE(3) robot's workspace, thus defining task-specific SE(3) cost functions. In particular, in this work, we show how to learn diffusion models for 6DoF grasping, leveraging open-source vastly annotated 6DoF grasp pose datasets like Acronym [46]. SE(3) diffusion models allow moving initially random samples to low-cost regions (regions of good grasping poses on objects) by evolving a gradient-based inverse diffusion process [226] (cf. Figure 5.2). SE(3) diffusion models come with two benefits. First, we get smooth cost functions in SE(3) that can be directly used in motion optimization. Second, they better cover and represent multimodal distributions, like in a 6DoF grasp generation scenario, leading to better and more sample efficient performance of the subsequent robot planning.

Consequently, we propose a joint grasp and motion optimization framework using the learned 6DoF grasp diffusion model as cost function and combining it with other differentiable costs (trajectory smoothness, collision avoidance, etc.). All costs combined (learned and hand-designed) form a single, smooth objective function that optimizing it enables the generation of good robot trajectories for complex robot manipulation tasks. This work shows how our framework enables facing grasp generation and classical trajectory optimization as a joint gradient-based optimization loop.

Our contributions are threefold: (**1**) we show how to **learn smooth cost functions in SE(3) as diffusion models**. While score-based generative modeling has been previously introduced for arbitrary Riemannian manifolds [16], we focus on the particular requirements for the Lie group SE(3). (**2**) **we use the SE(3) diffusion models to learn 6DoF grasp pose distributions as cost functions**. Our experiments show that our learned models generate more diverse and successful grasp poses w.r.t. state-of-the-art grasp generative models. Once the model is trained, (**3**) we introduce **a gradient-based optimization framework for jointly resolving grasp and motion generation**, in which we integrate our learned 6DoF grasp diffusion model with additional task-related cost terms. To properly integrate diffusion models in the motion optimization problem, we rewrite the optimization as an inverse diffusion process, similarly to [90]. In contrast with previous methods that decouple the grasp pose selection and the motion planning, our framework resolves the grasp and motion planning problem by iteratively improving the trajectory to jointly minimize the learned object-grasp cost term and the task-related costs. We remark that this joint optimization is only possible thanks to the smoothness of our learned diffusion model and using instead a grasp classifier, trained with cross-entropy loss, as cost won't resolve the problem due to its lack of smoothness. Our quantitative and qualitative results in simulation and the real-world robotic manipulation experiments suggest that our proposed method for learning costs as SE(3) diffusion models enables efficiently finding good grasp and motion solutions against baseline approaches and resolves complex pick-and-place tasks as in Figure 5.1.

5.2. Preliminaries

Diffusion Models. Unlike common deep generative models (VAE, generative adversarial networks (GAN)) that explicitly generate a sample from a noise signal, diffusion models learn to generate samples by iteratively moving noisy random samples towards a learned distribution [224, 227]. A common approach to train diffusion models is by *DSM* [249, 205]. To apply DSM [224, 225], we first perturb the data distribution $\rho_{\mathcal{D}}(\boldsymbol{x})$ with Gaussian noise on L noise scales $\mathcal{N}(\boldsymbol{0}, \sigma_k \boldsymbol{I})$ with $\sigma_1 < \sigma_2 < \cdots < \sigma_L$, to obtain a noise perturbed distribution $q_{\sigma_k}(\hat{\boldsymbol{x}}) = \int_{\boldsymbol{x}} \mathcal{N}(\hat{\boldsymbol{x}}|\boldsymbol{x}, \sigma_k \boldsymbol{I})\rho_{\mathcal{D}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$. To sample from the perturbed distribution, $q_{\sigma_k}(\hat{\boldsymbol{x}})$ we first sample from the data distribution $\boldsymbol{x} \sim \rho_{\mathcal{D}}(\boldsymbol{x})$ and then add white noise $\hat{\boldsymbol{x}} = \boldsymbol{x} + \epsilon$ with $\epsilon \sim \mathcal{N}(\boldsymbol{0}, \sigma_k \boldsymbol{I})$. Next, we estimate the score function of each noise perturbed distribution $\nabla_{\boldsymbol{x}} \log q_{\sigma_k}(\boldsymbol{x})$ by training a noise-conditioned vector field $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}, k)$, by score matching $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}, k) \approx \nabla_{\boldsymbol{x}} \log q_{\sigma_k}(\boldsymbol{x})$ for all $k = 1, \ldots, L$. The training objective of DSM [205] is

$$\mathcal{L}_{\text{dsm}} = \frac{1}{L}\sum_{k=0}^{L} \mathbb{E}_{\boldsymbol{x},\hat{\boldsymbol{x}}} \left[\left\| \boldsymbol{s}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}}, k) - \nabla_{\hat{\boldsymbol{x}}} \log \mathcal{N}(\hat{\boldsymbol{x}}|\boldsymbol{x}, \sigma_k^2 \boldsymbol{I}) \right\| \right], \tag{5.1}$$

with $\boldsymbol{x} \sim \rho_{\mathcal{D}}(\boldsymbol{x})$ and $\hat{\boldsymbol{x}} \sim \mathcal{N}(\boldsymbol{x}, \sigma_k \boldsymbol{I})$ To generate samples from the trained model, we apply Annealed Langevin MCMC [157]. We first draw an initial set of samples from a distribution $\boldsymbol{x}_L \sim \rho_L(\boldsymbol{x})$ and then, simulate an inverse Langevin diffusion process for L steps, from $k = L$ to $k = 1$

$$\boldsymbol{x}_{k-1} = \boldsymbol{x}_k + \frac{\alpha_k^2}{2}\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_k, k) + \alpha_k \boldsymbol{\epsilon}\,,\ \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{5.2}$$

with $\alpha_k > 0$ a step dependent coefficient. Overall, DSM Equation (5.1) learns models that output vectors pointing towards the samples of the training dataset $\rho_{\mathcal{D}}(\boldsymbol{x})$ [226].

SE(3) Lie group. The SE(3) Lie group is prevalent in robotics. A point $\boldsymbol{H} = \left[\begin{smallmatrix}\boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1\end{smallmatrix}\right] \in$ SE(3) represents the full pose (position and orientation) of an object or robot link with $\boldsymbol{R} \in$ SO(3) the rotation matrix and $\boldsymbol{t} \in \mathbb{R}^3$ the 3D position. A Lie group encompasses the concepts of group and smooth manifold in a unique body. Lie groups are smooth manifolds whose elements have to fulfil certain constraints. Moving along the constrained manifold is achieved by selecting any velocity withing the space tangent to the manifold at $\boldsymbol{H}$ (i.e., the so-called tangent space). The tangent space at the identity is called *Lie algebra* and noted $\mathfrak{se}(3)$. The Lie algebra has a non-trivial structure, but is isomorphic to the vector space $\mathbb{R}^6$ in which we can apply linear algebra. As in [223], we work in the vector space $\mathbb{R}^6$ instead of the Lie algebra $\mathfrak{se}(3)$. We can move the elements between the Lie group and the vector space with the logarithmic and exponential maps, Logmap : SE(3) → $\mathbb{R}^6$ and Expmap : $\mathbb{R}^6$ → SE(3) respectively [223]. A Gaussian distribution on Lie groups can be defined as

$$q(\boldsymbol{H}|\boldsymbol{H}_\mu, \boldsymbol{\Sigma}) \propto \exp\left(-0.5 \ \left\|\text{Logmap}(\boldsymbol{H}_\mu^{-1}\boldsymbol{H})\right\|^2_{\boldsymbol{\Sigma}^{-1}}\right), \tag{5.3}$$

with $\boldsymbol{H}_\mu \in SE(3)$ the mean and $\boldsymbol{\Sigma} \in \mathbb{R}^{6\times 6}$ the covariance matrix [30]. This special form is required as the distance between two Lie group elements is not represented in Euclidean space. Following the notation of [223], given a function f : SE(3) → $\mathbb{R}$, the derivative w.r.t. a SE(3) element, $Df(\boldsymbol{H})/D\boldsymbol{H} \in \mathbb{R}^6$ is a vector of dimension 6. We refer the reader to [223] and the Appendix in project site for an extended presentation of the SE(3) Lie group.

5.3. SE(3)-Diffusion Fields

In this section, we show how to adapt diffusion models to the Lie group SE(3) [223], as it is a crucial space for robot manipulation. The SE(3) space is not Euclidean, hence, multiple design choices need to be considered for adapting Euclidean diffusion models. In the following, we first explain the required modifications (Section 5.3.1). Then, we propose a neural network architecture for learning SE(3) diffusion models that represent 6DoF grasp pose distributions and show how we train it (Section 5.3.2). Finally, we show how to integrate the learned diffusion models into a grasp and motion optimization problem and show how to optimize it jointly considering the grasp and the motion (Section 5.4).

5.3.1. From Euclidean diffusion to diffusion in SE(3)

A diffusion model in SE(3) is a *vector field* that outputs a vector $\boldsymbol{v} \in \mathbb{R}^6$ for an arbitrary query point $\boldsymbol{H} \in$ SE(3), i.e., $\boldsymbol{v} = s_{\boldsymbol{\theta}}(\boldsymbol{H}, k)$ with a scalar conditioning variable k determining

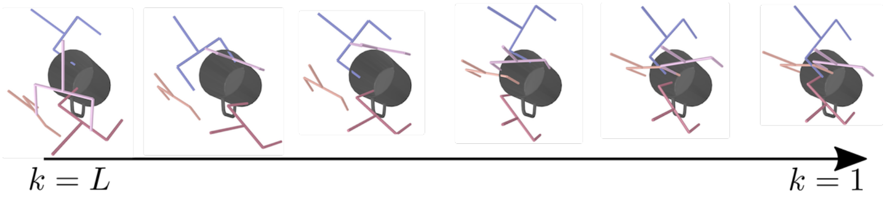


Figure 5.2.: Generating high quality SE(3) grasp poses by iteratively refining random initial samples (k=L) with an inverse Langevin diffusion process over SE(3) elements (Equation (5.6)).

the current noise scale [224].

Denoising Score Matching in SE(3). Similar to the Euclidean space version (cf. Section 5.2), DSM is applied in two phases. We first generate a perturbed data point in SE(3), i.e., sample from the Gaussian on Lie groups Equation (5.3), $\hat{\boldsymbol{H}} \sim q(\hat{\boldsymbol{H}}|\boldsymbol{H}, \sigma_k \boldsymbol{I})$ with mean $\boldsymbol{H} \in \rho_{\mathcal{D}}(\boldsymbol{H})$ and standard deviation σ_k for noise scale k. Practically, we sample from this distribution using a white noise vector $\boldsymbol{\epsilon} \in \mathbb{R}^6$,

$$\hat{\boldsymbol{H}} = \boldsymbol{H}\text{Expmap}(\boldsymbol{\epsilon}) \, , \, \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma_k^2 \boldsymbol{I}). \tag{5.4}$$

Following the idea of DSM, the model is trained to match the score of the perturbed training data distribution. Thus, DSM in SE(3) requires computing the derivatives of the perturbed distribution w.r.t. a Lie group element. Hence, the new DSM loss function on Lie groups equates to

$$\mathcal{L}_{\text{dsm}} = \frac{1}{L} \sum_{k=0}^{L} \mathbb{E}_{\boldsymbol{H},\hat{\boldsymbol{H}}} \left[\left\| \boldsymbol{s}_{\boldsymbol{\theta}}(\hat{\boldsymbol{H}}, k) - \frac{D \log q(\hat{\boldsymbol{H}}|\boldsymbol{H}, \sigma_k \boldsymbol{I})}{D\hat{\boldsymbol{H}}} \right\| \right], \tag{5.5}$$

with $\boldsymbol{H} \sim \rho_{\mathcal{D}}(\boldsymbol{H})$ and $\hat{\boldsymbol{H}} \sim q(\hat{\boldsymbol{H}}|\boldsymbol{H}, \sigma_k \boldsymbol{I})$. Note that, as introduced in Section 5.2, the derivatives w.r.t. a SE(3) element $\hat{\boldsymbol{H}}$ outputs a vector on $\mathbb{R}^6$. In practice, we compute this derivative by automatic differentiation using Theseus [178] library along with PyTorch.

Sampling with Langevin MCMC in SE(3). Evolving the inverse Langevin diffusion process for SE(3) elements (cf. Figure 5.2 for visualization) requires adapting the previously presented Euclidean Langevin MCMC approach Equation (5.2). In particular, we have to ensure staying on the SE(3) manifold throughout the inverse diffusion process. Thus, we adapt the inverse diffusion in SE(3) as

$$\boldsymbol{H}_{k-1} = \text{Expmap}\left(\frac{\alpha_k^2}{2} \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{H}_k, k) + \alpha_k \boldsymbol{\epsilon} \right) \boldsymbol{H}_k, \tag{5.6}$$

with $\boldsymbol{\epsilon} \in \mathbb{R}^6$ sampled from $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and the step dependent coefficient $\alpha_k > 0$. By iteratively applying Equation (5.6), we move a set of randomly sampled SE(3) poses to

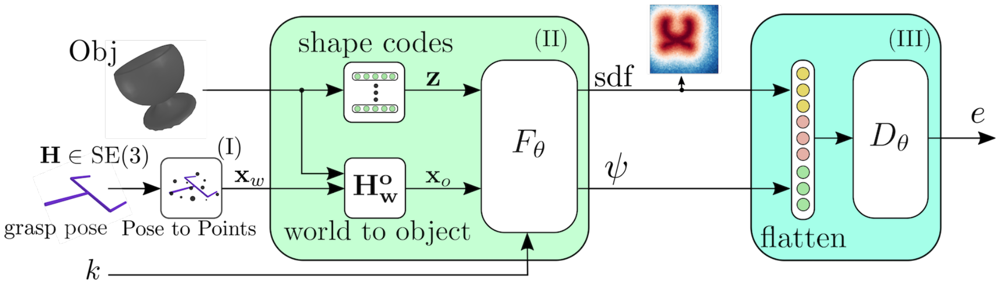


Figure 5.3.: SE(3)-DiF's architecture for learning 6D grasp pose distributions. We train the model to jointly learn the objects' sdf and to minimize the denoising loss. Given grasp pose $\boldsymbol{H}\in\mathrm{SE}(3)$ we transform it to a set of 3D points $\boldsymbol{x}_w\in\mathbb{R}^{N\times3}$ (**I**). Next, we transform the points into the object's local frame, using the object's pose $\boldsymbol{H}_w^o$. Given the resulting points $\boldsymbol{x}_o$ and the object's shape code $\boldsymbol{z}$ we apply the feature encoder F_θ (**II**) to obtain a object and grasp-related features (sdf, ψ)$\in$ $\mathbb{R}^{N\times(\psi+1)}$. Finally, (**III**) we flatten the features and compute the energy e through the decoder D_θ. We provide a point-cloud-based implementation in our code repository: https://github.com/TheCamusean/grasp_diffusion

the data distribution $\rho_D(\boldsymbol{H})$ (See Figure 5.2).
From the score function to energy model. While most of the works in learning diffusion models learn a vector field representing the score $\boldsymbol{s_\theta}$, in our work, we learn a scalar field that represents the energy of the distribution $E_{\boldsymbol{\theta}}$. In contrast with learning a score function, learning an EBM allow us evaluating the quality of the generated samples and compose it with other cost functions for multi-objective motion optimization. To learn an EBM with denoising score matching, we model our score function $\boldsymbol{s_\theta}(\boldsymbol{H},k) = -DE_{\boldsymbol{\theta}}(\boldsymbol{H},k)/D\boldsymbol{H}$, as the derivative of the EBM $E_{\boldsymbol{\theta}}$.

5.3.2. Architecture & training of Grasp SE(3)-DiffusionFields

Even though we can represent any data-driven cost in SE(3) with SE(3)-DiF, in this work, we focus on cost functions that capture 6DoF grasp pose distributions conditioned on the object we aim to grasp. In this work, we assume to have access to the object pose, a reasonable assumption thanks to the impressive results in 6DoF object pose estimation and segmentation [253]. We defer studying the perception aspect of encoding point clouds into object pose and shape as in [152, 92] for a future work. We illustrate the architecture for our grasp SE(3)-DiF model in Figure 5.3 and the training pipeline in Algorithm 7. The proposed model maps an object (represented by its id and pose) and a 6DoF grasp pose $\boldsymbol{H} \in \mathrm{SE}(3)$ to an energy $e \in \mathbb{R}$, that measures the grasp quality for the particular object.

We train the model to jointly match the Signed Distance Field (SDF) of the object we aim to grasp and predict the grasp energy level by the DSM loss Equation (5.5). Learning jointly the SDF of the object and the grasp pose improves the quality of the grasp generation [92, 219]. During the training, we assume the object's id m and pose $\boldsymbol{H}_w^o \in \mathrm{SE}(3)$ are available, and we retrieve a learnable object shape code $\boldsymbol{z}_m$ given the index m as in [168]. For training the SDF loss, we apply a supervised learning pipeline. Given a dataset of 3D points $\boldsymbol{x}_w \in \mathbb{R}^3$ and $\mathrm{sdf} \in \mathbb{R}$ for a particular object m, $\mathcal{D}_{sdf}^m : (\boldsymbol{x}_w, \mathrm{sdf})$, we first map the points to the object's reference frame $\boldsymbol{x}_o = \boldsymbol{H}_w^o \boldsymbol{x}_w$ and then predict the SDF given the feature encoder $F_{\boldsymbol{\theta}}$ (See Algorithm 7).

As previously introduced in Equation (5.5), to apply the DSM loss, we compute the energy $e \in \mathbb{R}$ over the grasp poses $\hat{\boldsymbol{H}}$. These grasp poses have been previously obtained by perturbing grasp poses from the dataset $\boldsymbol{H} \in \rho_{\mathcal{D}}(\boldsymbol{H})$ with a noise level k Equation (5.4). In our problem, we consider $\rho_{\mathcal{D}}(\boldsymbol{H})$ to be a distribution of successful grasp poses for a particular object, and learn the energy to approximate the log-probability of this distribution under noise. We compute the energy e given a grasp pose $\hat{\boldsymbol{H}}$ in three steps. **(I)** We transform the grasp pose to a fixed set of N 3D-points around the gripper $\boldsymbol{x}_g \in \mathbb{R}^{N\times 3}$ in the world frame $\boldsymbol{x}_w = \boldsymbol{H}\boldsymbol{x}_g$. We thereby express the grasp pose through a set of 3D points' positions, similar to [219]. Then, we move the points to the object's local frame, $\boldsymbol{x}_{o_m} = \boldsymbol{H}_w^{o_m} \boldsymbol{x}_w$. **(II)** We apply the feature encoding network $F_{\boldsymbol{\theta}}$ which is also conditioned on $\boldsymbol{z}_m$ and k to inform about the object shape and noise level, respectively. The encoding network outputs both the SDF predictions for the query points, $\mathrm{sdf} \in \mathbb{R}^{N\times 1}$, and a set of additional features $\psi \in \mathbb{R}^{N\times\psi}$. Thus, the feature encoder's output is of size $N \times (1+\psi)$. **(III)** We flatten the features and pass them through the decoder $D_{\boldsymbol{\theta}}$ to obtain the scalar energy value e. Given the energy, we compute the DSM loss Equation (5.5). During training, we jointly learn the objects' latent codes $\boldsymbol{z}_m$, and the parameters $\boldsymbol{\theta}$ of the feature encoder $F_{\boldsymbol{\theta}}$ and decoder $D_{\boldsymbol{\theta}}$.

5.4. Grasp and motion optimization with diffusion models

Given a trajectory $\boldsymbol{\tau} : \{\boldsymbol{q}_t\}_{t=1}^T$, consisting of T waypoints, with $\boldsymbol{q}_t \in \mathbb{R}^{d_q}$ the robot's joint positions at time instant t; in motion optimization, we aim to find the minimum cost trajectory $\boldsymbol{\tau}^* = \arg\min_{\boldsymbol{\tau}} \mathcal{J}(\boldsymbol{\tau}) = \arg\min_{\boldsymbol{\tau}} \sum_j \omega_j c_j(\boldsymbol{\tau})$, where the objective function $\mathcal{J}$ is a weighted sum of costs c_j, with weights $\omega_j > 0$. Herein, we integrate the learned SE(3)-DiF for grasp generation as one cost term of the objective function. It is, thus, combined with other heuristic costs, e.g., collision avoidance or trajectory smoothness. Optimizing over

Algorithm 7: Grasp SE(3)-DiF Training

Given : $\boldsymbol{\theta}_0$: initial params for $\boldsymbol{z}$, $F_{\boldsymbol{\theta}}$, $D_{\boldsymbol{\theta}}$;
Datasets: $\mathcal{D}_o : \{m, \boldsymbol{H}_w^o\}$, object ids and poses, $\mathcal{D}_{sdf}^m : \{\boldsymbol{x}, \text{sdf}\}$, 3D positions $\boldsymbol{x}$ and sdf for object m, $\mathcal{D}_g^m : \{\boldsymbol{H}\}$ succesful grasp poses for object m;

```
for $s \leftarrow 0$ to $S-1$ do
    $k, \sigma_k \leftarrow [0, \dots, L]$;                                   // sample noise scale
    $m, \boldsymbol{H}_w^o \in \mathcal{D}_o$;                               // sample objects ids and poses
    $\boldsymbol{z} = \text{shape codes}(m)$;                                 // get shape codes
    SDF train
    $\boldsymbol{x}, \text{sdf} \in \mathcal{D}_{sdf}^m$;                      // get 3D points and sdf for obj. $m$
    $\hat{sdf}, \_ = F_{\boldsymbol{\theta}}(\boldsymbol{H}_w^o \boldsymbol{x}, \boldsymbol{z}, k)$;   // get predicted sdf
    $L_{\text{sdf}} = \mathcal{L}_{\text{mse}}(\hat{sdf}, sdf)$;              // compute sdf error
    Grasp diffusion train
    $\boldsymbol{H} \sim \mathcal{D}_g^m$;                                    // Sample success grasp poses for obj. $m$
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma_k \boldsymbol{I})$;   // sample white noise on $k$ scale
    $\hat{\boldsymbol{H}} = \boldsymbol{H}\text{Expmap}(\boldsymbol{\epsilon})$;   // perturb grasp pose eq. (5.4)
    $\boldsymbol{x}_n^o = \hat{\boldsymbol{H}} \boldsymbol{x}_n$;              // Transform to N 3d points (see Figure 5.3)
    $\hat{sdf}_n, \boldsymbol{\psi}_n = F_{\boldsymbol{\theta}}(\boldsymbol{x}_n^o, \boldsymbol{z}_b, k)$;   // get features
    $\Psi = \text{Flatten}(\hat{sdf}_n, \boldsymbol{\psi}_n)$;               // Flatten the features
    $e = D_{\boldsymbol{\theta}}(\Psi)$;                                      // compute energy
    $L_{\text{dsm}} = \mathcal{L}_{\text{dsm}}(e, \hat{\boldsymbol{H}}, \boldsymbol{H}, \sigma_k)$;   // Compute dsm loss eq. (5.5)
    Parameter update
    $L = L_{\text{dsm}} + L_{\text{sdf}}$;                                    // Sum losses
    $\boldsymbol{\theta}_{s+1} = \boldsymbol{\theta}_s - \alpha \nabla_{\boldsymbol{\theta}} L$;   // Update parameters
return $\boldsymbol{\theta}*$;
```

the whole set of costs enables obtaining optimal trajectories jointly taking into account grasping, as well as motion-related objectives. This differs from classic grasp and motion planning approaches in which the grasp pose sampling and trajectory planning are treated separately [113], by first sampling the grasp pose, and, then, searching for a trajectory that satisfies the selected grasp. In classic approaches, given the grasp sampling is decoupled from the trajectory planning, it might happen the sampled grasps to be unfeasible for the problem, leading to an unsolvable trajectory planning problem. We hypothesize that jointly optimizing over both the grasp pose and the trajectory allows us to be more sample efficient w.r.t. decoupled approaches.

Given that the learned function is in SE(3) while the optimization is w.r.t. the robot's joint space, we redefine the cost as $c(\boldsymbol{q}_t, k) = E_{\boldsymbol{\theta}}(\phi_{ee}(\boldsymbol{q}_t), k)$, with the forward kinematics $\phi_{ee} : \mathbb{R}^{d_q} \rightarrow \text{SE}(3)$ mapping from robot configuration to the robot's end-effectors task space. To obtain minimum cost trajectories, *we frame the motion generation problem as an inverse diffusion process*. Using a planning-as-inference view [18, 126, 240, 90], we define a desired target distribution as $q(\boldsymbol{\tau}|k) \propto \exp(-\mathcal{J}(\boldsymbol{\tau}, k))$. This allows us to set an inverse Langevin diffusion process that evolves a set of random initial particles drawn from a distribution $\boldsymbol{\tau}_L \sim p_L(\boldsymbol{\tau})$ towards the target distribution $q(\boldsymbol{\tau}|k)$

$$\boldsymbol{\tau}_{k-1} = \boldsymbol{\tau}_k + 0.5\ \alpha_k^2 \nabla_{\boldsymbol{\tau}_k} \log q(\boldsymbol{\tau}|k) + \alpha_k \boldsymbol{\epsilon}\,,\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{5.7}$$

with step dependent coefficient $\alpha_k > 0$, noise level moving from $k = L$ to $k = 1$, and one particle corresponding to an entire trajectory. If we evolve the particles by this inverse diffusion process for sufficient steps, the particles at $k = 1$, $\boldsymbol{\tau}_1$ can be considered as particles sampled from $q(\boldsymbol{\tau}|k = 1)$. To obtain the optimal trajectory, we evaluate the samples on $\mathcal{J}(\boldsymbol{\tau}, 1)$ and pick the one with the lowest cost.

5.5. Experimental Evaluation

The experimental section is divided in three parts. First, we evaluate our trained model for 6DoF grasp pose generation (Section 5.5.1). We train a SE(3)-DiF as a 6DoF grasp pose generative model using the Acronym dataset [46]. This simulation-based dataset contains successful 6DoF grasp poses for a variety of objects from ShapeNet [26]. We focus on the collection of successful grasp poses for 90 different mugs (approximately 90K 6DoF grasp poses). We provide a model trained in a larger dataset and conditioned on point cloud in the project page. We obtain the mugs' meshes from ShapeNet, and train the model as described in Algorithm 7. We generate a set of grasp poses from the learned models and evaluate on successful grasping and diversity. Second, we evaluate the quality of our trained model when used as an additional cost term for grasp and motion optimization (Section 5.5.2). We compare the performance of solving a grasp and motion optimization problem jointly (using the learned model as cost function), w.r.t. the state-of-the-art approaches that decouple the grasp selection and motion planning, or heuristically combine them. Finally, we validate the performance of our method in a set of real robot experiments (Section 5.5.3).

5.5.1. Evaluation of 6DoF grasp pose generation

We evaluate grasp poses generated from our trained grasp SE(3)-DiF model in terms of the success rate, and the EMD between the generated grasps and the training data distribution. We consider 90 different mugs and evaluate 200 generated grasps per mug. We evaluate the grasp success on Nvidia Isaac Gym [144]. The EMD measures the divergence between

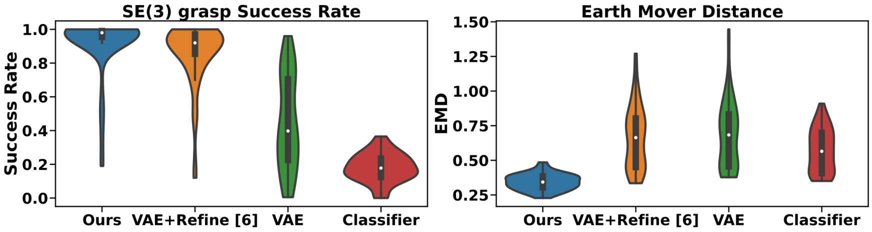
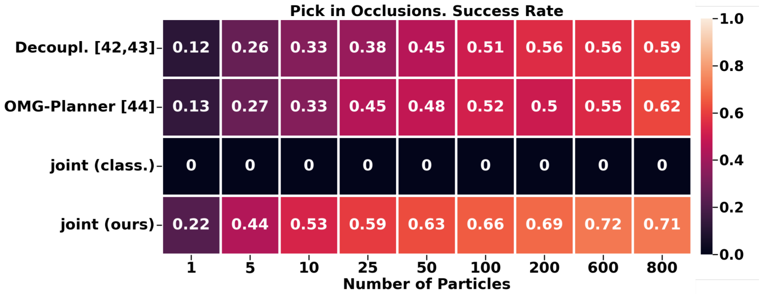


Figure 5.4.: 6D grasp pose generation experiment. Left: Success rate evaluation. Right: Earth Mover Distance (EMD) evaluation metrics (lower is better).

two empirical probability distributions [230], providing a metric on how similar the generated samples are to the training dataset. To eliminate any other influence, we only consider the gripper and assume that we can set it to any arbitrary pose. We generate 6DoF grasp poses from SE(3)-DiF by an inverse diffusion process, following Equation (5.6).

We compare against three baselines. First, based on [152, 228], we consider generating grasp poses by first sampling from a decoder of a trained VAE and subsequently running MCMC over a trained classifier for pose refinement (VAE+Refine). Second, we consider sampling from the VAE (without any further refinement). Third, we consider running MCMC over the classifier starting from random initial pose [170]. In this experiment, we assume the object's pose and id/shape to be known, and purely focus on evaluating the models' generative capabilities. For ensuring a fair comparison, all the baselines consider a shape code z_m to encode the object information as presented in Figure 5.3. We add a pointcloud-conditioned experiment in the Appendix.

We present the results in Figure 5.4. In terms of success rate, SE(3)-DiF outperforms VAE+Refine slightly (especially yielding lower variance), and VAE or classifier on their own significantly. The VAE alone generates noisy grasp poses that are often in collision with the mug. In the case of classifier only, the success rate is low. We hypothesize that this might be related with the classifier's gradient, as specifically in regions far from good samples, the field has a large plateau with close to zero slopes [7]. This leads to not being able to improve the initial samples. Considering grasp diversity, i.e., EMD metric (lower is better), SE(3)-DiF outperforms all baselines significantly. A reason for the difference, might be that VAE+Refine overfits to specific overrepresented modes of the data distribution. In contrast, SE(3)-DiF's samples capture the data distribution more properly. We, therefore, conclude that SE(3)-DiF is indeed generating high-quality and diverse grasp poses. We add an extended presentation of the experiment in the Appendix in our project site.



	1	5	10	25	50	100	200	600	800
Decoupl. [42,43]	0.12	0.26	0.33	0.38	0.45	0.51	0.56	0.56	0.59
OMG-Planner [44]	0.13	0.27	0.33	0.45	0.48	0.52	0.5	0.55	0.62
joint (class.)	0	0	0	0	0	0	0	0	0
joint (ours)	0.22	0.44	0.53	0.59	0.63	0.66	0.69	0.72	0.71

Figure 5.5.: Evaluation Pick in occlusion. We measure the success rate of 4 different methods based on different number of initializations.

5.5.2. Performance on grasp and motion optimization

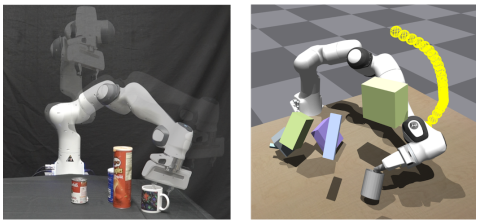


Figure 5.6.: Simulated and real robot environments for picking amidst clutter.

We evaluate the performance of our learned grasp SE(3)-DiF as a cost term into multi-objective grasp and motion optimization problems. We consider the task of picking amidst clutter (see Figure 5.6) and measure the success rate on solving it. The success is measured based on the robot being able to grasp the object at the end of the execution. In the Appendix in our project site, we provide additional details on the chosen cost functions for the task. As introduced in Section 5.4, we generate the trajectories by integrating our learned grasp SE(3)-DiF as an additional cost function to the motion optimization objective function. Then, given a set of initial trajectory samples, obtained from a Gaussian distribution with a block diagonal matrix as in [97], we apply gradient descent methods Equation (5.7) to iteratively improve the trajectories on the objective function. We evaluate the success rate of the trajectory optimization given a different number of initial samples. As gradient-based trajectory optimization methods are inherently local optimization methods, multiple initializations might lead to better results. We consider three baselines (see Figure 5.5). **Decoupl.**: we adopt the common routing to solve grasp and motion optimization problems in a decoupled way [183, 142, 156]. We first sample a set of 6DoF grasp poses from a generative model and then plan a trajectory that satisfies the selected grasp pose with CHOMP [188]. Second, we consider the **OMG-Planner** [250], that applies an online grasp selection and planning approach. Finally, **joint (class.)**: we consider applying a joint optimization as in our approach, but using a 6DoF grasp classifier as cost function rather than a grasp SE(3)-DiF.

The results in Figure 5.5 present a clear benefit from the joint optimization w.r.t. the decoupled approach and the OMG-Planner. In particular, our proposed joint optimization only requires 25 particles to match the success rate of the decoupled approach with 800 particles. The reason for this significant gap in efficiency is that the decoupled approach generates SE(3) grasp poses that are not feasible given the environment constraints, such as clutter or joint limits. However, when optimizing jointly, we can find trajectories that satisfy all the costs by iteratively improving entire trajectories w.r.t. all objectives. We also observe the importance of using grasp SE(3)-DiF as cost term instead of a grasp classifier. The classifier model lacks proper gradient information to inform how to move the trajectories to grasp the object due to its lack of smoothness in the whole space. Thus, the motion optimization problem is unable to find solutions.

5.5.3. Grasp and motion optimization on real robots

We conducted a thorough real-world evaluation of our joint grasp and motion optimization framework driven by our 6DoF grasp diffusion model, using it as an additional cost function, similarly to our simulated robot manipulation tasks. Figure 5.1 depicts a sequence of a real-world pick-mug and place-on-shelf scenario. Overall, the experiments aim at assessing the method's capabilities in realistic conditions that include, i) non-perfect state information, as the mugs pose is retrieved from an external system (Optitrack) which induces small calibration errors, ii) variations in the mug's shape, as we use a mug that is slightly different from the one we specify for SE(3)-DiF, and iii) real-world trajectory execution. For optimization, we initialize 800 particles (trajectories), and only execute the one with lowest cost.

In the simplest testing scenario, where the robot has to pick up a mug from various poses in a scene without any clutter, we achieve **100%** (20 successes / 20 trials) pickup-success. We also find that our method transfers well to the more difficult scenarios of picking up mugs that are initially placed upside down with **90%** (18/20) success, picking in occluded scenes with **95%** (19/20) success, and having to pick and place the mug in a desired pose inside the shelf of Figure 5.1 with **100%** (20/20) success. Our real-world results underline the effectiveness of our joint optimization approach. Videos of the experiments also showcase that our method still comes up with very versatile solutions[1].

[1] Videos in `https://sites.google.com/view/se3dif`

Note that we attribute the increased real-world performance w.r.t. the simulated one to the simpler designed experimental scene, i.e., in simulation we considered flying obstacles that were not realizable in the real scene (Figure 5.6). Nevertheless, our results confirm that our proposed approach is highly performant in real settings, without suffering sim2real discrepancies.

Limitations In our experiments, we focused on evaluating our diffusion model's performance in grasp generation, besides full trajectory optimization, assuming full object state knowledge, without relying on complex perception systems. Potential sim2real gaps w.r.t. the real environment could potentially arise from imperfect perception, and hand-designed cost terms that may not capture well the relevant task description in more complex scenarios. Moreover, a limitation comes with increasing number of cost terms, as it becomes more difficult to weight them.

5.6. Related Work

Diffusion models in Robotics Diffusion models have appeared in robotics in various tasks, from text-conditioned scene rearrangement [98, 133], decision-making [90, 2, 29, 25, 251, 257] and controllable traffic generation[271]. We additionally highlight earlier works like [169], where a diffusion process is integrated into a motion planning problem.
6D grasp generation. 6D grasp pose generation is solved with a myriad of methods from classifiers to explicit samplers. [135, 170, 129] sample candidate grasps and score them with learned classifiers. [261] predicts grasping outcomes using a geometry-aware representation. Contrary to methods classifying grasps, generative models can be trained to generate grasp poses from data [152] but might require additional sample refinement. While the generator in [228] considers possible collisions in the scene, [75] proposes to learn a grasp distribution over the object's manifold. [92] uses scene representation learning to learn grasp qualities and explicitly predict 3D rotations. Recently, [254] proposed learning a 6 DoF SDF to represent grasp pose generation as a smooth cost function and optimize on top of it.
Integrated grasp and motion planning. Due to the interdependence of the selected grasp pose with the robot motion, multiple efforts have tried to integrate both variables into a single planning problem [39, 11, 245, 250, 58]. In [39, 11], goal sets representing grasp poses are integrated as constraints in a motion optimization problem. In [245, 54], Rapidly-exploring Random Trees [119] is combined with a TCP attractor to bias the tree towards good grasps.

5.7. Conclusions

We proposed SE(3)-DiffusionFields (SE(3)-DiF) for learning task-space, data-driven cost functions to enable robotic motion generation through joint gradient-based optimization over a set of combined cost functions. At the core of SE(3)-DiFs is a diffusion model that provides informative gradients across the entire space and enables data generation through an inverse Langevin dynamics diffusion process. Besides having demonstrated that SE(3)-DiF generates diverse and high-quality 6DoF grasp poses, we also drew a connection between motion generation and inverse diffusion. Thus, we presented a joint gradient-based grasp and motion optimization framework, which outperforms traditional decoupled optimization approaches. Our extensive experimental evaluations reveal the superior performance of the proposed method w.r.t. efficiency, adaptiveness, and success rates. In the future, we want to explore diffusion models for reactive motion control and the composition of multiple diffusion models to solve complex manipulation tasks in which multiple hard-to-model objectives might arise.

6. Conclusion

This thesis investigates the problem of learning deep generative models for robot motion generation. The main theme of this thesis is that the use of deep generative models in robotics requires the consideration of the inherent properties of the robotic systems in the design of the models. A careful choice of the model will boost the performance and generalization of the learned model. In this regard, we have proposed novel models to integrate stability in flow-based policies, studied the composable and geometric properties of EBM to represent multi-objective modular policies, and introduced novel algorithms to learn Diffusion Models in SE(3) and show their application for both grasps pose generation and motion planning.

6.1. Summary of Contributions

The contributions of this thesis are four, each presented in a separate chapter from Chapter 2 to Chapter 5.

In Chapter 2, we explore the links between deep generative models and robot motion generation. Although deep generative models have been used for behavioral cloning, learning motion primitives, or learning cost functions and rewards in inverse reinforcement learning/optimal control, the relationship between different generative models and motion generation methods has not been explicit. In this chapter, we aimed to provide an overview of the different methods that have used deep generative models to learn robot motion and to relate different types of models to different types of tasks. For example, if we want to learn a high-frequency control policy, we might want to learn a generative model such as a NFlow or a GAN, if we want to represent a cost function, we should learn a EBM, while if we want to learn the gradient of costs, we might be interested in learning a DDPM. We expect that this work will help the research community in selecting training algorithms and models when faced with different motion generation problems.

Then in three separate chapters, we presented three distinct methods that explore different robotics-relevant architectural properties in deep generative models.
In Chapter 3, we explored the problem of learning globally stable motion policies. We introduced ImitationFlows , a model that extends NFlow to represent globally stable dynamical systems, and showed that this model can be applied in robotics as a reactive motion policy. Previous work in the field of motion primitives had explored the problem of representing globally stable policies [103, 192, 158, 175]. However, most previous work has considered simple models, which limits their expressiveness. We have shown, through an experimental evaluation, the benefit of increasing expressivity to represent more complex dynamic systems. In addition, we extended ImitationFlows to the Lie group SE(3). This extension allowed us to represent stable policies in the end-effector space. We found this particularly useful and saw a clear benefit in generation, as many robotic tasks are inherently represented in this space rather than the configuration space.
In Chapter 4, we studied the problem of representing multi-objective motion policies. We introduced Composable Energy Policies , a novel way to generate robot motion by composing a set of EBM. Previous approaches [191, 28, 21] proposed to represent each motion generation module with deterministic models. This could limit policy composition when different modules generate conflicting motion behaviors. In our work, we extended modular motion generation approaches to energy-based policies. We showed that this probabilistic view improves the composition of multiple policies and provides a framework for integrating learned generative models for multi-objective motion generation.
In Chapter 5, we addressed the problem of learning cost functions for motion planning. We proposed SE(3)-DiffusionFields , a novel approach to represent cost functions as Diffusion Models. In our work, we showed how to extend Diffusion Models to the Lie group SE(3), which allows learning Diffusion Models in a robotics-relevant space. This allows the representation of distributions on end-effector grasp poses. We then showed that Diffusion Models can be used as cost functions in motion planning problems, allowing the integration of data-driven models with well-established robot motion planning algorithms. Finally, we demonstrated the benefits of integrating these learned models into robot pick-and-place tasks.

6.2. Open Challenges and Future Work

During the time in which this Ph.D. has been pursued (2019-2023), the field of Deep Generative Models has exploded.
Large Language Models (LLM) have emerged as powerful text generation models [20,

236] and have found many applications in a wide variety of situations. In the field of Robotics, LLMs have been used for learning text-conditioned policies [216, 217, 68]. Given a text command and an observation of the scene, the robot outputs a distribution of desirable actions to satisfy the text command. Also, inspired by LLMs, several works in robotics have started to explore learning a foundation model that can solve multiple robot tasks fed with massive amounts of demonstrations [19, 143, 196].
Recent advances in **Diffusion Models** [227, 80, 81] allow high-dimensional density models to be learned and high-quality samples to be generated. These models have been particularly successful in text-based image generation [204, 269], but similar models have begun to be integrated into robotics problems [244, 90, 29, 133, 220]. We have recently observed the application of Diffusion Models to learn policies that can solve more than 60 different household tasks, and it is expected that the number of behaviors will grow over time.

Following these trends, the foreseeable future for deep generative models in robot motion generation is to learn a **foundational model of robot motion behaviors**. The model should have a number of features:

1. **Multi-Skill Model**. A foundational behavior model should generate desirable motions for a wide set of tasks. The robot should be able to generate motions to open bottles, clean a table, or open doors to name a few possible desirable tasks.

2. **Generalizable Behaviors**. If the behavior of opening doors is learned, the model should be able to represent this behavior for arbitrary doors, with arbitrary handles and in arbitrary environments.

3. **Language Conditioned Models**. The model should be conditioned in natural language to allow easy interaction between the user and the robot. In addition, the robot should understand different language commands as long as the robot is able to perform the desired skill.

What are the open challenges to reach this future?

6.2.1. Out-of-distribution generation

An important factor for generalization is for the robot to generate desirable movements beyond the demonstrated region. Even if we can increase the data, how can we guarantee that the robot will adapt its behavior to different object arrangements or novel tools?

Along with this thesis, we have explored the possibility of integrating architectural elements into our motion generators to induce desirable behaviors beyond the demonstrations. We have studied the representation of motion behaviors in the Lie group SE(3), in order to represent the motion in the space corresponding to the tasks to be solved, as opposed to the configuration space. We have studied stability to induce safe behaviors beyond the demonstrations, and we have studied composability as a way to modularise the motion generation problem into small individual components that can be plugged together to solve complex tasks.

In the future, we will continue to explore architectural elements to improve the generalization capabilities of the motion generators. Several directions are relevant

Goal-conditioned skill primitives Following the intuition of our work, we strongly believe that complex robot motion should be built as a composition of multiple skill modules. We believe that complex robot behaviors can be generated by concatenating skill primitives. These skill primitives should be simple enough to be reusable in multiple tasks, yet complex enough to provide a useful abstraction for longer horizon tasks. In the future, we aim to explore useful skill representations. What should be the goal conditioning for the skill primitives? In what space should we represent the goal? Should it be geometric or semantic?

Motion-scene reference frame Most robot tasks require interaction with elements in the scene. Opening a door requires defining the robot's motion with respect to the door, and screwing requires defining the motion with respect to both the screw and the screwdriver. We believe that an important element for generalization is the correct choice of the motion reference frame. By representing the door opening behavior in the door reference frame and the screwing motion in the screw reference frame, we can easily adapt the robot's behavior to new positions and orientations. In the future, we want to explore how these reference frames can be extracted from visual data. Can we model our motion generators to adapt naturally by extracting the reference frame of motion from scene observations?

6.2.2. Scaling-up data

Another challenge is data. The success of deep generative models in both text and image generation depends largely on the amount of data available. If we want to have a foundational model of movement behaviors, we need to increase the amount of data available.

One possible direction is through **teleoperation**. Providing demonstrations of desired behaviors directly in the robot allows easy deployment of the learned behaviors since the demonstrations are directly embodied. However, scaling the teleoperation data would require a farm of robot teachers to generate desired behaviors in the robot.

A possible alternative direction is **human videos**. The Internet is full of videos of humans performing various activities. We can find demonstrations of humans doing all sorts of activities and interacting with all sorts of tools and objects. Can we use the data in these videos to train our robots?

Adapting Motion Data to Robot Policies One of the limitations of the video data is the difference in embodiment. While the demonstrations are performed by a human, we aim to implement the demonstrations in a robot. Also, the recorded data only considers kinematic information (positions and velocities), while for proper interaction with the scene, the robot is expected to apply forces and torques to the different tools and objects in the scene.

In the future, we want to explore how to bridge this gap. We will explore the combination of deep generative models with reinforcement learning to tackle this problem in a similar way to [79, 173, 172]. Similar to SE(3)-DiffusionFields, we aim to learn Diffusion Models as reward functions and use them in reinforcement learning to find physically plausible policies that maximize the probability of our learned generative model.

A. Appendix

A.1. ImitationFlows Stability evaluation

In the following, we prove the asymptotic stability in probability of the learned system, under the assumption of a stable latent dynamic.

Lyapunov Stability Lyapunov stability studies the stability guarantees of a dynamical system. To do so, we evaluate if a Lyapunov potential candidate $V(\boldsymbol{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ and its time derivative $\dot{V}(\boldsymbol{x})$ satisfies a set of conditions. For the case of stochastic dynamics, the conditions are evaluated over the expected value $\mathbb{E}\left[\dot{V}(\boldsymbol{x})\right]$ [145].

Given the stochastic dynamics in Equation (3.1), the time derivative of the Lyapunov function, $V(\boldsymbol{x}, t)$, is expressed by the following differential equation

$$\begin{aligned} \mathrm{d}V(\boldsymbol{x}) &= LV(\boldsymbol{x})\mathrm{d}t + V_{\boldsymbol{x}}(\boldsymbol{x})\boldsymbol{\sigma}(\boldsymbol{x})\mathrm{d}\boldsymbol{W}_t \\ LV(\boldsymbol{x}) &= V_{\boldsymbol{x}}\boldsymbol{g}(\boldsymbol{x}) + \frac{1}{2}\mathrm{Tr}(\boldsymbol{\sigma}^{\intercal}(\boldsymbol{x}))V_{\boldsymbol{x}\boldsymbol{x}}\boldsymbol{\sigma}(\boldsymbol{x}), \end{aligned} \tag{A.1}$$

with $V_{\boldsymbol{x}} = \partial V(\boldsymbol{x})/\partial \boldsymbol{x}$ and $V_{\boldsymbol{x}\boldsymbol{x}} = \partial^2 V(\boldsymbol{x})/\partial \boldsymbol{x}^2$.
The expected value of $\dot{V}$ is

$$\mathbb{E}\left[\dot{V}(\boldsymbol{x})\right] = LV(\boldsymbol{x}). \tag{A.2}$$

Given a strictly increasing functions μ_1, μ_2, μ_3 such that

$$\mu_1(|\boldsymbol{x}|) \leqslant V(\boldsymbol{x}) \leqslant \mu_2(|\boldsymbol{x}|), \tag{A.3a}$$

$$LV(\boldsymbol{x}) \leqslant -\mu_3(|\boldsymbol{x}|)\, \forall \boldsymbol{x} \in \mathbb{R}^d. \tag{A.3b}$$

Then, the trivial solution for the SDE is stochastically asymptotically stable [145].

Lemma 1 *For any diffeomorphic transformation $\boldsymbol{x}_t = \Phi(\boldsymbol{z}_t)$, if the dynamics of $\boldsymbol{z}(t)$ are stochastically asymptotically stable, then the dynamics of $\boldsymbol{x}_t$ are also stochastically asymptotically stable.*

We follow a derivation similar to the one proposed in [159] for Lyapunov stability analysis in Ordinary Differential Equation (ODE). Let $U(\cdot)$ be a Lyapunov function for the latent dynamics $\boldsymbol{z}$. We define the following Lyapunov candidate for the base space $\boldsymbol{x}$ dynamics:

$$V(\boldsymbol{x}) = U(\Phi^{-1}(\boldsymbol{x})).$$

From this definition it follows that

$$U(\boldsymbol{z}) = V(\Phi(\boldsymbol{z})). \tag{A.4}$$

From the equality in Equation (A.4) and by the fact that U is a valid Lyapunov candidate we get that the condition in Equation (A.3a) is also satisfied.

To satisfy the condition in Equation (A.3b), we compute the Lyapunov function LV Equation (A.1). Considering Equation (A.1) and Equation (3.10)

$$LV(\boldsymbol{x}) = V_{\boldsymbol{x}}\boldsymbol{J}(\boldsymbol{x})\boldsymbol{A}\Phi^{-1}(\boldsymbol{x}) + \frac{1}{2}\mathrm{Tr}(\boldsymbol{\sigma}^\intercal \boldsymbol{J}^\intercal(\boldsymbol{x}) V_{\boldsymbol{xx}} \boldsymbol{J}(\boldsymbol{x})\boldsymbol{\sigma}). \tag{A.5}$$

Moreover, we can rewrite $V_{\boldsymbol{x}}$ and $V_{\boldsymbol{xx}}$ in terms of $U_{\boldsymbol{z}}$ and $U_{\boldsymbol{zz}}$.

$$V_{\boldsymbol{x}}(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}} V(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{z}} U(\boldsymbol{z}) \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = U_{\boldsymbol{z}}(\boldsymbol{z})\boldsymbol{J}^{-1}(\boldsymbol{x}), \tag{A.6}$$

where $U_{\boldsymbol{z}}(\cdot) : \mathbb{R}^d \to \mathbb{R}^{1\times d}$.For the case of $V_{\boldsymbol{xx}}$

$$\begin{aligned} V_{\boldsymbol{xx}}(\boldsymbol{x}) = \frac{\partial^2}{\partial \boldsymbol{x}^2} V(s) &= \frac{\partial}{\partial \boldsymbol{x}^\intercal}\left(\frac{\partial}{\partial \boldsymbol{x}} V(\boldsymbol{x})\right) = \\ &= \frac{\partial}{\partial \boldsymbol{x}^\intercal}\left(U_{\boldsymbol{z}}(\boldsymbol{z})\boldsymbol{J}^{-1}(\boldsymbol{x})\right) \\ &= \frac{\partial \boldsymbol{z}^\intercal}{\partial \boldsymbol{x}^\intercal}\frac{\partial}{\partial \boldsymbol{z}^\intercal}\left(U_{\boldsymbol{z}}(\boldsymbol{z})J^{-1}(\boldsymbol{x})\right) \\ &= \boldsymbol{J}^{-\intercal}(\boldsymbol{x})U_{\boldsymbol{zz}}(\boldsymbol{z})\boldsymbol{J}^{-1}(\boldsymbol{x}). \end{aligned} \tag{A.7}$$

Introducing Equation (A.6) and Equation (A.7) in Equation (A.5)

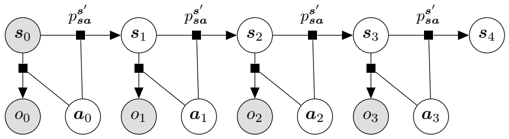
$$LV(\boldsymbol{x}) = U_{\boldsymbol{z}}\boldsymbol{A}\boldsymbol{z} + \frac{1}{2}\mathrm{Tr}(\boldsymbol{\sigma}^\intercal U_{\boldsymbol{zz}}\boldsymbol{\sigma}) = LU(\boldsymbol{z}). \tag{A.8}$$

By hypothesis $LU(\boldsymbol{z})$ satisfies the condition in Equation (A.3b), therefore the condition is also satisfied by $LV(\boldsymbol{x})$.

A.2. A Control as Inference view for Composable Energy Policies

In the following section, we want to highlight the connections between composable energy policies and control as inference to evaluate the optimality guarantees of composing energies in a multi-objective optimal control problem.

Figure A.1.: Graphical model for the Optimal Control problem. $\boldsymbol{s}_t$ denoted the state, $\boldsymbol{a}_t$ denotes the action and o_t is an additional variable representing the optimality of the state and action for a given reward.



We frame optimal control as a Bayesian inference problem [195, 126] over the sequence of actions $\boldsymbol{a}_{0:T}$. The optimal control problem is visualized as a graphical model in Figure A.1. The problem is formulated introducing an auxiliary variable $o_{0:T}$ that represents the optimality of $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$ under a certain reward function, $p(o_t|\boldsymbol{s}_t, \boldsymbol{a}_t) \propto \exp(r(\boldsymbol{s}_t, \boldsymbol{a}_t))$. Given a certain prior distribution $q(\boldsymbol{a}|\boldsymbol{s}_0)$ and given $\boldsymbol{s}_0$ is known, the inference problem is

$$p(A_0^T|\boldsymbol{s}_0, O_0^T) = \frac{p(O_0^T|A_0^T, \boldsymbol{s}_0)q(A_0^T|\boldsymbol{s}_0)}{p(O_0^T|\boldsymbol{s}_0)} \tag{A.9}$$

with

$$p(O_0^T|A_0^T, \boldsymbol{s}_0) = \int_{\boldsymbol{s}_{1:T}} p(O_0^T|S_0^T, A_0^T)p(S_1^T|A_0^T, \boldsymbol{s}_0)dS_1^T \tag{A.10}$$

where $A_0^T : \{\boldsymbol{a}_0, \dots, \boldsymbol{a}_T\}$, $O_0^T : \{o_0, \dots, o_T\}$ and $S_0^T : \{\boldsymbol{s}_0, \dots, \boldsymbol{s}_T\}$. The are two main directions to solve the posterior in Equation (A.9). First, methods that frame the problem as an Hidden Markov Model (HMM) and solve it in an Expectation-Maximization approach. Second, methods that compute the posterior in the trajectory level A_0^T. The first, are computationally demanding as they require several forward and backward message passing to compute the posterior. The second, needs to solve the problem in the trajectory level and thus the dimension of the variables grows linearly with the trajectory length, T.

In our work, we consider solving a one-step-ahead optimal control problem. Rather than solving an optimization problem for a sequence of actions A_0^T, we solve the problem for a single step $\boldsymbol{a}_0$.

We can reframe the control as inference problem as a one-step ahead control problem

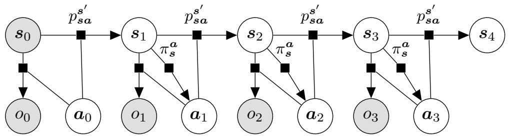
$$p(\boldsymbol{a}_0|\boldsymbol{s}_0, O_0^T) = \frac{p(O_0^T|\boldsymbol{a}_0, \boldsymbol{s}_0)q(\boldsymbol{a}_0|\boldsymbol{s}_0)}{p(O_0^T|\boldsymbol{s}_0)} \tag{A.11}$$

with

$$\begin{aligned} & p(O_0^T|\boldsymbol{a}_0, \boldsymbol{s}_0) = \\ & \int_{S_{1:T}} \int_{A_{1:T}} p(O_0^T|S_0^T, A_0^T)p(S_1^T|\boldsymbol{a}_0, \boldsymbol{s}_0)\pi(A_1^T|S_1^T)dS_1^T dA_1^T. \end{aligned} \tag{A.12}$$

In contrast with the trajectory optimization problem that finds the posterior for the whole trajectory A_0^T, in one-step-ahead control as inference problem, we aim to find the posterior for only the instant next control action, $\boldsymbol{a}_0$. Computing the posterior only for $\boldsymbol{a}_0$ requires the likelihood to be defined as the marginal of not only the state trajectory S_1^T, but also the action trajectory A_1^T. The graphical model for one-step-ahead control as inference is presented in Figure A.2. In one-step ahead control as inference, we introduce an additional distribution π that provides us the probability of A_1^T given S_1^T. This additional policy π is interpreted as the policy the agent will apply in the future. In this context, we are looking for the action the maximizes the cumulative reward in the long horizon trajectory running the policy π. Equation (A.12) can be rewritten as the expectation over $p(O_0^T|S_0^T, A_0^T)$

Figure A.2.: Graphical model for one-step ahead optimal control problem. In this approach, the actions A_1^T are dependant on S_1^T given a policy π.



$$\begin{aligned} \mathbb{E}_{S_0^T, A_0^T \sim p^\pi(S_0^T, A_0^T|\boldsymbol{s}_0, \boldsymbol{a}_0)} \left[p(O_0^T|S_0^T, A_0^T)\right] \propto \\ \mathbb{E}_{S_0^T, A_0^T \sim p^\pi(S_0^T, A_0^T|\boldsymbol{s}_0, \boldsymbol{a}_0)} \left[\exp(\sum_{t=0}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t))\right] = \\ \exp(Q_r^\pi(\boldsymbol{s}_0, \boldsymbol{a}_0)) \end{aligned} \tag{A.13}$$

From what follows, the likelihood for our inference problem is proportional to the $\exp(Q)$ defined over a certain policy π. Given a π, the posterior for our inference problem is given by

$$p(\boldsymbol{a}_0|\boldsymbol{s}_0, O_0^T) \propto \exp(Q_r^\pi(\boldsymbol{s}_0, \boldsymbol{a}_0))q(\boldsymbol{a}_0|\boldsymbol{s}_0). \tag{A.14}$$

The provided Q function depends on π and then, the quality of our reactive motion generator to solve a long horizon optimal control problem directly depends on the quality of π. In the optimal case, for Q^*, the one-step ahead control problem follows the optimal trajectory, even if the optimization is done locally. Nevertheless, in CEP we aim to study the obtained policy in a multi-objective framework. Given we have a set of Q functions, each being optimal for a particular reward; is the sum of the Q functions still optimal for the sum of the rewards?

A.2.1. Optimality Guarantees

In CEP, we propose to model the Q function as the sum of a set of optimal Q_k^* functions. Instead, we are aware that $Q_\Sigma = \frac{1}{K}\sum_{k=0}^K Q_k^*$ is not the optimal function Q^* for the sum of the rewards $r = \frac{1}{K}\sum_{k=1}^K r_k$. The closer Q_Σ is from the optimal Q^*, the closer the product of experts policy would be from the optimal policy. In this section, we study, given a certain reward $r = \frac{1}{2}(r_1 + r_2)$, how much the sum of the individual components Q_Σ diverge from the optimal Q^*. In [126] is shown, that the optimal Q function for the control as inference problem can be computed by recursively solving the soft-value iteration [272]

$$\begin{aligned} Q(\boldsymbol{s}, \boldsymbol{a}) &= r(\boldsymbol{s}, \boldsymbol{a}) + \mathbb{E}_{p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})}\left[V(\boldsymbol{s}')\right] \\ V(\boldsymbol{s}) &= \log\left(\int_{\mathcal{A}} \exp(Q(\boldsymbol{a}, \boldsymbol{s}))\mathrm{d}\boldsymbol{a}\right). \end{aligned} \tag{A.15}$$

We assume a finite horizon control as inference problem and evaluate how much Q_Σ diverges from $Q*$ when increasing the control horizon

For $t = T$, the optimal Q function for the sum of the rewards is

$$Q^{*T} = \frac{1}{2}(r_1 + r_2). \tag{A.16}$$

The individual optimal Q functions for each reward are

$$Q_1^{*T} = r_1\,,\; Q_2^{*T} = r_2. \tag{A.17}$$

Then, the sum of the Q components is given by

$$Q_{\Sigma}^{T} = \frac{1}{2}(Q_1^{*T} + Q_2^{*T}). \tag{A.18}$$

If we compute the distance between the optimal Q^{*T} and the sum of Q's, Q_{Σ}^{T}

$$\Delta Q^{T} = Q^{*T} - Q_{\Sigma}^{T} = 0. \tag{A.19}$$

From Equation (A.18), for $T = 1$, the sum of the optimal components Q_{Σ} is equal to the optimal Q.

For longer temporal horizons, the optimal Q is computed by recursively solving the soft Bellman update backward in time. For computing $t = T - 1$,

$$\begin{aligned} Q^{*T-1}(\boldsymbol{s}, \boldsymbol{a}) &= r(\boldsymbol{s}, \boldsymbol{a}) + \mathbb{E}_{p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})}\left[V^{*T}(\boldsymbol{s}')\right] \\ V^{*T}(\boldsymbol{s}) &= \log \int_{\mathcal{A}} \exp(Q^{*T}(\boldsymbol{a}, \boldsymbol{s})). \end{aligned} \tag{A.20}$$

we do a soft Bellman update. The difference between Q^{*T-1} and Q_{Σ}^{T-1}

$$\Delta Q^{T-1} = Q^{*T-1} - Q_{\Sigma}^{T-1} = \mathbb{E}[V^{*T} - V_{\Sigma}^{T}] \tag{A.21}$$

with

$$V_{\Sigma}^{T} = \frac{1}{2}(V_1^{*T} + V_2^{*T}). \tag{A.22}$$

The distance in the value function can be represented as

$$\begin{aligned} V^{*T} - V_{\Sigma}^{T} &= \log \frac{\int_{\mathcal{A}} \exp(Q^{*T})}{\int_{\mathcal{A}}(\exp(Q_1^{*T}) \int_{\mathcal{A}} \exp(Q_2^{*T}))^{\frac{1}{2}}} \\ &= \log \frac{\int_{\mathcal{A}} \exp(\frac{1}{2}Q_1^{*T}) \exp(\frac{1}{2}Q_2^{*T})}{(\int_{\mathcal{A}} \exp(Q_1^{*T}) \int_{\mathcal{A}} \exp(Q_2^{*T}))^{\frac{1}{2}}}. \end{aligned} \tag{A.23}$$

Then,

$$
\begin{aligned}
Q^{*T-1} &= Q_{\Sigma}^{T-1} + \Delta Q^{T-1} \\
&= Q_{\Sigma}^{T-1} + \\
\mathbb{E}_{p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})} & \left[\log \frac{\int_{\mathcal{A}} \exp(\frac{1}{2} Q_1^{*T}) \exp(\frac{1}{2} Q_2^{*T})}{(\int_{\mathcal{A}} \exp(Q_1^{*T}) \int_{\mathcal{A}} \exp(Q_2^{*T}))^{\frac{1}{2}}} \right]
\end{aligned} \tag{A.24}
$$

From Equation (A.21) and Equation (A.23), we can obtain the recurrence relation for the distance error

$$
\begin{aligned}
&\Delta Q^{t-1} = \\
&\mathbb{E}_{p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})} \left[\log \frac{\int_{\mathcal{A}} \exp(\frac{1}{2} Q_1^{*t}) \exp(\frac{1}{2} Q_2^{*t}) \exp(\Delta Q^{t})}{(\int_{\mathcal{A}} \exp(Q_1^{*t}) \int_{\mathcal{A}} \exp(Q_2^{*t}))^{\frac{1}{2}}} \right].
\end{aligned} \tag{A.25}
$$

From Equation (A.23), we can see that if $Q_1^* = Q_2^*$, then $\Delta Q = 0$ and the more they differ, the bigger the distance error to the optima Q^*. From the obtained results, we can obtain some conclusions.

Similar theoretical studies have been already developed [72, 247, 233]. In Optimal Control, composable optimality guarantees were proven for linear dynamics, by the LMDP approach. In [233, 33], a weighted policy sum was proven to be optimal for the sum of the rewards, as long as the rewards differ only in the terminal reward. In [72, 231], the optimality of the composition is studied in a maximum entropy reinforcement learning problem.

A.3. Experiments

We present additional details for the experiments in Chapter 4.

A.3.1. Reaching through a cluttered environment

The modular components (reach target, obstacle avoidance and joint limits avoidance) for both baselines APF and RMP were modeled based on [105] and [28] respectively. The CEP without hand was modelled with the energy policy components introduced in [240] and

Energy Modules	Parameters		
Reach Target	$\boldsymbol{K}_p = 20.$	$\boldsymbol{K}_v = 30.$	$\alpha = 10.$
Obstacle Avoidance	$\gamma = 0.2$	$\alpha = 4.$	$\beta = 0.1$
Joint Limits avoidance	$\gamma = 0.3$	$\alpha = 4.$	$\beta = 0.1$

Table A.1.: Component parameters for Composable Energy Policies in reaching through cluttered environment [240].

with the parameters In this work, we have extended the experiments with an additional evaluation with the robot hand. We considered the energy policies in Appendix A.4 to model the energy policies. We considered a target reaching policy, a set of obstacle avoidance policies, joint limits avoidance policy and a joint velocity limits policy. We show in Table A.2 the parameters we consider for the experiments. The reaching target policy is parameterized by the metric $\boldsymbol{\Lambda}$ that frames the relevance of this component and α that defines the maximum Euclidean distance of the truncated target position. The obstacle avoidance energy policy is parameterized by the desired minimum distance between the surfaces of the body sphere and the obstacle sphere. The joint velocity limits is parameterized by the metric defining the importance of this component. As we can observe, in contrast with [242], in the current approach, we have (i) less tuning parameters for each energy component and (ii) there is an intuition in the meaning of each parameter.

Energy Modules	Parameters
Reach target	$\boldsymbol{\Lambda} = \boldsymbol{I}$, $\alpha = 0.05$
Obstacle avoidance	$\alpha = r_{\text{obstacle}} + r_{\text{body}}$ + 0.01
Joint velocity limits	$\boldsymbol{\Lambda} = 0.1\boldsymbol{I}$

Table A.2.: Component parameters for Composable Energy Policies in reaching through cluttered environment with hand. r_{obstacle} and r_{body} represent respectively, the radius of the collision spheres in the obstacle and the robot body.

A.3.2. Learning to hit a puck

In the reinforcement learning experiment, we consider the same hyperparameters for CEP, residual operational space control and direct operational space control. Additionally, we keep the same hyperparameters for the three reward functions. We consider the reinforcement learning algorithms implemented in Mushroom-RL [32]. In the following, we introduce a table with the hyperparameters for PPO, SAC, DDPG and TD3.

Hyperparameters	
policy net	18-128-128-3
policy batch	64
policy learn rate	3e-4
critic learn rate	3e-4
critic net	18-128-128-1
critic batch	256
critic learn rate	3e-4
n_steps_per_fit	600
discount factor	.99
eps_ppo	0.1

Table A.3.: **PPO** hyperparameters for hitting the puck

Hyperparameters	
policy net mean	18-128-128-3
policy net sigma	18-128-128-3
policy batch	64
policy learn rate	3e-4
critic learn rate	3e-4
critic net	18-128-128-1
critic batch	256
critic learn rate	3e-4
n_steps_per_fit	1
discount factor	.99
target entropy	-6
warmup transitions	10000
max replay size	200000

Table A.4.: **SAC** hyperparameters for hitting the puck

A.4. A practical overview of energy policies

While the CEP framework can be used for arbitrary agents; in our work, we focus on the problem of generating motion for robot manipulators. The robot state in the configuration space $\boldsymbol{s}^q$ is represented by the robot's position $\boldsymbol{q}$, velocity $\dot{\boldsymbol{q}}$ and environment information $\boldsymbol{c}$ (obstacles position, obstacles shape, target pose). The action in the configuration space $\boldsymbol{a}^q$ is the robot's joint acceleration $\ddot{\boldsymbol{q}}$.

The energy policies are defined in a set of task spaces. We model the map from the

Hyperparameters	
policy net	18-128-128-3
policy batch	64
policy learn rate	3e-4
critic learn rate	3e-4
critic net	18-128-128-1
critic batch	256
critic learn rate	3e-4
n_steps_per_fit	1
discount factor	.99
warmup transitions	10000
max replay size	200000

Table A.5.: **DDPG** and **TD3** hyperparameters for hitting the puck

configuration space to the different task spaces $\boldsymbol{f}_q^{x_k}$ by the robot's kinematics

$$\begin{aligned}\boldsymbol{x}_k &= \phi_q^{x_k}(\boldsymbol{q})\\ \dot{\boldsymbol{x}}_k &= \boldsymbol{J}^k(\boldsymbol{q})\dot{\boldsymbol{q}}\\ \ddot{\boldsymbol{x}}_k &= \boldsymbol{J}^k(\boldsymbol{q})\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}^k(\boldsymbol{q})\dot{\boldsymbol{q}} \approx \boldsymbol{J}^k(\boldsymbol{q})\ddot{\boldsymbol{q}},\end{aligned} \tag{A.26}$$

with $\phi_q^{x_k}$ the forward kinematics to a given k task space and $\boldsymbol{J}^k(\boldsymbol{q}) = \partial \boldsymbol{x}_k / \partial \boldsymbol{q}$, the Jacobian for the given forward kinematics.

In CEP, we provide a framework to compose policies from different motion generation paradigms such as optimal control, imitation learning, movement primitives, reinforcement learning, or handcrafted policies. In this section, we introduce a practical overview of the different sources from which a policy component could be computed. In Appendix A.4.1 we introduce a set of analytically computed energy policies to represent a set of basic local behaviors. Then, in Appendix A.4.2, we introduce a set of possible methods to learn energy policies from data. Finally, in Appendix A.4.3, we briefly introduced a set of methods to obtain energy policies from optimal control or reinforcement learning.

A.4.1. Basic local reactive energies

In a previous work [240], we proposed handcrafted models to represent the local reactive energies. Handcrafted policies might lead to difficult parameter tuning when using them and lack an intuition of the objective they are trying to maximize. In this work, we propose to represent the energies as value functions maximizing a particular reward $r : \mathcal{S} \rightarrow \mathbb{R}$. In multiple problems, defining the behavior of the robot to solve a particular tasks with a reward function might be more intuitive and easier than defining it directly as a policy. If the task is easier to define in the state space rather than the action space, a reward function provides us with a natural form to describe the desired behavior. For example, in the case of collision avoidance, it is easier to represent the desired behavior in terms of the robot's position rather than with respect to the robot's acceleration. Modeling the energy of the policies in terms of the value function has been widely studied in the maximum entropy reinforcement learning community [272, 73]. Under some assumptions, the value functions can be computed analytically. This approach allows the practitioner to understand the objective the policy is trying to maximize and provides additional intuition to tune the parameters.

The energies are represented by the optimal advantage function A for a one-step control horizon problem

$$A(\boldsymbol{a}|\boldsymbol{s}) = r(\boldsymbol{s}) + \mathbb{E}_{\rho(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})}\left[V(\boldsymbol{s}')\right] - V(\boldsymbol{s}), \tag{A.27}$$

with $V(\boldsymbol{s}) = r(\boldsymbol{s})$ and $\rho(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})$ the transition dynamics. We can observe, that for the particular case of one-step control horizon with state dependant rewards, the advantage function is simply the expected value function given the transition dynamics.

Defining the state $\boldsymbol{s} = (\boldsymbol{x}, \dot{\boldsymbol{x}})$ by the position and the velocity and the action $\boldsymbol{a} = \ddot{\boldsymbol{x}}$ by the acceleration, we model the transition dynamics with the explicit Euler discretization of a linear dynamic system

$$\begin{aligned} \boldsymbol{x}_{t+1} &= \boldsymbol{x}_t + \dot{\boldsymbol{x}}_t \Delta t + \frac{1}{2}\Delta t^2 \ddot{\boldsymbol{x}}_t \\ \dot{\boldsymbol{x}}_{t+1} &= \dot{\boldsymbol{x}}_t + \Delta t \ddot{\boldsymbol{x}}_t, \end{aligned} \tag{A.28}$$

with Δt being the step size. In the following we show that for some particular reward function, we can analytically derive the optimal advantage function that we exploit as the energy of our policy.

Target position Consider the problem of reaching a certain target position. The reward function to solve the problem can be modelled by the negative Mahalanobis distance to the target position $\boldsymbol{x}_g$

$$r(\boldsymbol{x}) = -||\boldsymbol{x} - \boldsymbol{x}_g||^2_{\boldsymbol{\Lambda}}. \tag{A.29}$$

The advantage function for the one-step ahead optimal control problem with state dependant reward is represented by

$$A(\boldsymbol{a}|\boldsymbol{s}) = \mathbb{E}_{\rho(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})}[r(\boldsymbol{s}')]. \tag{A.30}$$

Given the dynamics in Equation (A.28) are deterministic, the optimal advantage function can be computed analytically, by applying a change of variables in the reward Equation (A.29)

$$A(\ddot{\boldsymbol{x}}|\boldsymbol{x},\dot{\boldsymbol{x}}) = -||\ddot{\boldsymbol{x}} - \ddot{\boldsymbol{x}}_g(\boldsymbol{x},\dot{\boldsymbol{x}})||^2_{\boldsymbol{\Lambda}^{\ddot{x}}}, \tag{A.31}$$

with

$$\begin{aligned}\ddot{\boldsymbol{x}}_g(\boldsymbol{x},\dot{\boldsymbol{x}}) &= \frac{2}{\Delta t^2}(\boldsymbol{x}_g - \boldsymbol{x} - \Delta t\dot{\boldsymbol{x}})\\ \boldsymbol{\Lambda}^{\ddot{x}} &= \frac{\Delta t^4}{4}\boldsymbol{\Lambda}.\end{aligned} \tag{A.32}$$

The maximum acceleration of the advantage function is the one that moves a point in Δt to the $\boldsymbol{x}_g$. In our work we want to control the robot in fast control rates ($< 0.01s$). When the robot is far from the target, reaching the target in that small Δt will require the robot to achieve very high accelerations. To avoid it, we model the new target position $\hat{\boldsymbol{x}}_g$ by

$$\hat{\boldsymbol{x}}_g = \boldsymbol{x} + \frac{\max(||\boldsymbol{x}_g - \boldsymbol{x}||, \alpha)}{||\boldsymbol{x}_g - \boldsymbol{x}||}(\boldsymbol{x}_g - \boldsymbol{x}). \tag{A.33}$$

The following equation projects the target position $\boldsymbol{x}_g$ to a ball centered in $\boldsymbol{x}$ and with a radius α. This way, the maximum Euclidean distance between the desired target and the current position is limited to α.

Target orientation We can apply a similar approach for reaching the desired orientation. In our work, we consider the orientation is represented in a Lie group $\boldsymbol{R} \in SO(3)$. Modeling a distance metric as the reward function is hard in the Lie Group given it is not a Euclidean space [223]. To properly model the reward function, we first map the rotation to the Lie algebra $\mathfrak{so}(3)$ centered in the target orientation $\boldsymbol{R}_g \in SO(3)$ (*We transform the rotation matrix to the axis-angle representation*)

$$\boldsymbol{\theta} = \text{LogMap}_{\boldsymbol{R}_g}(\boldsymbol{R}), \tag{A.34}$$

with LogMap being the logarithmic map that moves a point in the Lie Group to the Lie algebra. The Lie algebra is an Euclidean space in which we can apply calculus. Given the

Lie algebra is centered at the target $\boldsymbol{R}_g$, the desired target position in the Lie algebra is $\boldsymbol{\theta}_g = \mathbf{0}$; the origin. Thus, we can model the reward in $\mathfrak{so}(3)$ by

$$r(\boldsymbol{\theta}) = -||\boldsymbol{\theta}||^2_{\boldsymbol{\Lambda}}. \tag{A.35}$$

The reward function will maximize when $\boldsymbol{\theta} = \mathbf{0}$ and quadratically reduces with respect to the Euclidean distance in the Lie algebra.

To compute the optimal advantage function, we first transform the rotation velocity and accelerations from the world frame (*We compute the world frame velocity and acceleration in Equation* (4.2)) to the target orientation frame $\boldsymbol{R}_g$

$$\begin{aligned} \boldsymbol{\omega}' &= \boldsymbol{R}_g^{-1}\boldsymbol{\omega} \\ \dot{\boldsymbol{\omega}}' &= \boldsymbol{R}_g^{-1}\dot{\boldsymbol{\omega}}. \end{aligned} \tag{A.36}$$

This map is known as the adjoint operation in the Lie Group theory [223]. The linear dynamics in the Lie algebra are

$$\begin{aligned} \boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k + \boldsymbol{\omega}'_k\Delta t + \frac{1}{2}\Delta t^2\dot{\boldsymbol{\omega}}'_k \\ \boldsymbol{\omega}'_{k+1} &= \boldsymbol{\omega}'_k + \Delta t\dot{\boldsymbol{\omega}}'_k. \end{aligned} \tag{A.37}$$

Once everything is represented in the Lie algebra centered at $\boldsymbol{R}_g$, we can similarly to Equation (A.31) compute the advantage function

$$A(\dot{\boldsymbol{\omega}}'|\boldsymbol{\theta},\boldsymbol{\omega}') = -||\dot{\boldsymbol{\omega}}' - \dot{\boldsymbol{\omega}}_g(\boldsymbol{\theta},\boldsymbol{\omega}')||^2_{\boldsymbol{\Lambda}^{\dot{\omega}'}}, \tag{A.38}$$

with

$$\begin{aligned} \dot{\boldsymbol{\omega}}_g(\boldsymbol{\theta},\boldsymbol{\omega}') &= \frac{2}{\Delta t^2}(\boldsymbol{\theta} - \Delta t\boldsymbol{\omega}') \\ \boldsymbol{\Lambda}^{\dot{\omega}'} &= \frac{\Delta t^4}{4}\boldsymbol{\Lambda}. \end{aligned} \tag{A.39}$$

Similarly to Equation (A.31), the acceleration maximizing the advantage is the one that sets the rotation to $\boldsymbol{R}_g$ in Δt. To bound the acceleration in the rotation we can also bound the target as in Equation (A.33).

Obstacle avoidance We represent obstacle avoidance energy in the unidimensional space represented by the vector between a cartesian robot position $\boldsymbol{x}$ in a certain task space and the cartesian obstacle position $\boldsymbol{x}_o$. We compute this energy for every combination of a set of task space points P and a set of obstacles O. The total obstacle avoidance energy components are $P \times O$.

We first compute the distance to the obstacles and the vector pointing to the obstacle

$$\begin{aligned} d_o &= ||\boldsymbol{x} - \boldsymbol{x}_o|| \\ \hat{\boldsymbol{v}}_o &= (\boldsymbol{x} - \boldsymbol{x}_o)/d_o, \end{aligned} \tag{A.40}$$

with d_o being the distance and $\hat{\boldsymbol{v}}_o$ the vector pointing to the obstacle.

We define the obstacle avoidance reward function by

$$r(d_o) = \begin{cases} 0 & \text{if} \quad d_o > \alpha \\ -\infty & \text{if} \quad d_o \leqslant \alpha \end{cases}, \tag{A.41}$$

with α being a parameter that represents the minimum allowed distance to the obstacle. The proposed reward function allows the robot to be in any position except those that approximate to the obstacle to a distance below α.

To represent the advantage function, we first compute the velocity and acceleration projected in the vector $\hat{\boldsymbol{v}}_o$

$$\begin{aligned} \dot{x}_p &= \dot{\boldsymbol{x}} \cdot \hat{\boldsymbol{v}}_o \\ \ddot{x}_p &= \ddot{\boldsymbol{x}} \cdot \hat{\boldsymbol{v}}_o. \end{aligned} \tag{A.42}$$

Given the dynamics in Equation (A.28), the dynamics in the projected space are

$$\begin{aligned} d_{ok+1} &= d_{ok} + \dot{x}_{pk}\Delta t + \frac{1}{2}\Delta t^2 \ddot{x}_{pk} \\ \dot{x}_{pk+1} &= \dot{x}_{pk} + \Delta t \ddot{x}_{pk}. \end{aligned} \tag{A.43}$$

Then, we represent the advantage function in the uni-dimensional space represented by the vector between the current task space point and the obstacle

$$A(\ddot{x}_p|\dot{x}_p, d_o) = \begin{cases} 0 & \text{if} \quad \ddot{x}_p > \alpha^{\ddot{x}_p}(\dot{x}_p, d_o) \\ -\infty & \text{if} \quad \ddot{x}_p \leqslant \alpha^{\ddot{x}_p}(\dot{x}_p, d_o) \end{cases}, \tag{A.44}$$

with

$$\alpha^{\ddot{x}_p}(\dot{x}_p, d_o) = \frac{2}{\Delta t^2}(\alpha - d_o - \dot{x}_p \Delta t). \tag{A.45}$$

Using the advantage function in Equation (A.44) as the energy of a policy represents a uniformly distributed policy

$$\pi(\ddot{x}_p | \dot{x}_p, d_o) = \mathcal{U}(\alpha^{\ddot{x}_p}(\dot{x}_p, d_o), \infty) \propto \exp(A(\ddot{x}_p | \dot{x}_p, d_o)). \tag{A.46}$$

Joint limits avoidance Similarly to the collision avoidance energy, we apply a binary reward to bound the joint limits. We define by $\underline{\boldsymbol{q}}$ and $\bar{\boldsymbol{q}}$ the minimum and maximum joints. We represent the reward by

$$r(\boldsymbol{q}) = \begin{cases} 0 & \text{if} \quad \boldsymbol{q} > \underline{\boldsymbol{q}} \text{ and } \boldsymbol{q} < \bar{\boldsymbol{q}} \\ -\infty & \text{otherwise} \end{cases}. \tag{A.47}$$

Given the reward in Equation (A.47), the advantage function is

$$A(\ddot{\boldsymbol{q}} | \dot{\boldsymbol{q}}, \boldsymbol{q}) = \begin{cases} 0 & \text{if} \quad \ddot{\boldsymbol{q}} > \underline{\ddot{\boldsymbol{q}}} \text{ and } \ddot{\boldsymbol{q}} < \bar{\ddot{\boldsymbol{q}}} \\ -\infty & \text{otherwise} \end{cases}, \tag{A.48}$$

with

$$\begin{aligned} \underline{\ddot{\boldsymbol{q}}}(\dot{\boldsymbol{q}}, \boldsymbol{q}) &= \frac{2}{\Delta t^2}(\underline{\boldsymbol{q}} - \boldsymbol{q} - \dot{\boldsymbol{q}} \Delta t) \\ \bar{\ddot{\boldsymbol{q}}}(\dot{\boldsymbol{q}}, \boldsymbol{q}) &= \frac{2}{\Delta t^2}(\bar{\boldsymbol{q}} - \boldsymbol{q} - \dot{\boldsymbol{q}} \Delta t). \end{aligned} \tag{A.49}$$

Joint velocity control Due to the myopic behavior of CEP, if the robot moves too fast, it might not be able to adapt fast enough to avoid collisions. Thus, we are interested in constraining the velocity the robot can achieve. We define a reward for the configuration space velocity

$$r(\dot{\boldsymbol{q}}) = -||\dot{\boldsymbol{q}}||^2_{\boldsymbol{\Lambda}}. \tag{A.50}$$

Given the reward in Equation (A.50), the advantage function is

$$A(\ddot{\boldsymbol{q}} | \boldsymbol{q}, \dot{\boldsymbol{q}}) = -||\ddot{\boldsymbol{q}} - \ddot{\boldsymbol{q}}_g(\dot{\boldsymbol{q}})||^2_{\boldsymbol{\Lambda}^{\ddot{q}}}, \tag{A.51}$$

with

$$\begin{aligned}\ddot{\boldsymbol{q}}_g(\dot{\boldsymbol{q}}) &= \dot{\boldsymbol{q}}/\Delta t \\ \boldsymbol{\Lambda}^{\ddot{q}} &= \Delta t^2 \boldsymbol{\Lambda}.\end{aligned} \tag{A.52}$$

All these control energies are purely local. As we have shown, the proposed energies try to maximize a one-step-ahead control horizon reward. While they perform well for local navigation with obstacles, some tasks require a longer horizon look ahead to properly solve the task. In these situations, our myopic policies might fail. Nevertheless, these ”smarter” policies could be obtained from data, given some expert demonstrations are provided or by applying long horizon optimal control or reinforcement learning and fitting a value function that solves a long horizon problem. Then, we could integrate these policies as an additional component of our CEP.

A.4.2. Learning energy policies from data

A common approach to learn policies from data is by behavioural cloning. Given a set of state-action pairs demonstrations $\mathcal{D} : \{\boldsymbol{s}_i, \boldsymbol{a}_i\}_{i=0:N}$, the policy is learned by a conditioned maximum likelihood estimation

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}} \left[\log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\theta})\right]. \tag{A.53}$$

While the most common case assumes a conditioned Gaussian distribution as a policy model π, several works consider more expressive policy models. In [242], a normalizing flow is used to model the policy distribution, while in [52] an EBM is proposed to model the policy and trained by contrastive divergence [78].

Alternatively, we can build complex energy policies from simply learned distributions

Mixture-of-Expert energies A possible option to represent multi-modal policy distributions is to build a mixture of energies policies. Given a set of already given energies $E_0, \dots, E_k$, we can compute the mixture of energies

$$E_M(\boldsymbol{a}, \boldsymbol{s}) = \log \sum_k w_k(\boldsymbol{s}) \exp(E_k(\boldsymbol{a}, \boldsymbol{s})), \tag{A.54}$$

with w_k the weighting term. For the particular case in which the energy E is quadratic, the energy policy in Equation (A.54) is the energy of a Gaussian mixture model.

Negated energy policy A more conservative approach to defining policies is by negative energies. Given a certain policy distribution, we might want our algorithm not to follow that policy without properly specifying what should be the desired path to follow. Given that the policy is modeled by energies of a Boltzman distribution $\pi(\boldsymbol{a}|\boldsymbol{s}) \propto \exp(E(\boldsymbol{a},\boldsymbol{s}))$, the negated policy is straightforwardly computed by negating the energy, $\pi_{\text{not}}(\boldsymbol{a}|\boldsymbol{s}) \propto \exp(-E(\boldsymbol{a},\boldsymbol{s}))$. This policy will inform about the action the robot should not do, rather than what to do.

A.4.3. Q-function in optimal control and reinforcement learning

The previously proposed advantage functions only solve a one-step-ahead control problem. Nevertheless, for several problems, we require to solve a longer horizon optimization. CEP enables integration of longer horizon value functions as energy components. We can integrate value functions learned by reinforcement learning [74] or optimal control [139].Alternatively, we could compute the distribution for an optimal trajectory distribution by particles as in Stein Variational MPC [116] and exploit this multi-modal distribution as a guiding policy. These learned models can be afterward integrated with additional energy policies to deal with specific parts that were not covered in the RL or optimal control problem.

B. Supplementary Material

B.1. Conference Papers

1. Hansel, K., **Urain, J.**, Peters, J., & Chalvatzaki, G. (2023). Hierarchical policy blending as inference for reactive robot control. IEEE International Conference on Robotics and Automation (ICRA)

2. **Urain, J.**, Funk, N., Peters, J., & Chalvatzaki, G. (2023). SE(3)-DiffusionFields: Learning smooth cost functions for joint grasp and motion optimization through diffusion. IEEE International Conference on Robotics and Automation (ICRA)

3. **Urain, J.**, Le, A. T., Lambert, A., Chalvatzaki, G., Boots, B., & Peters, J. (2022). Learning implicit priors for motion optimization. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

4. **Urain, J.**, Li, A., Liu, P., D'Eramo, C., & Peters, J. (2021). Composable Energy Policies for Reactive Motion Generation and Reinforcement Learning. Robotics: Science & Systems (R:SS)

5. **Urain, J.**, Ginesi, M., Tateo, D., & Peters, J. (2020). Imitationflow: Learning deep stable stochastic dynamic systems by normalizing flows. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

6. **Urain, J.**, & Peters, J. (2019). Generalized multiple correlation coefficient as a similarity measurement between trajectories. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

B.2. Journal Articles

1. **Urain, J.**, Li, A., Liu, P., D'Eramo, C., & Peters, J. (2023). Composable energy policies for reactive motion generation and reinforcement learning. The International Journal of Robotics Research (IJRR)

2. **Urain, J.**, Tateo, D., & Peters, J. (2022). Learning stable vector fields on Lie groups. IEEE Robotics and Automation Letters (RA-L)

3. Funk, N., Schaff, C., Madan, R., Yoneda, T., **Urain, J.**, Watson, J., ... & Peters, J. (2021). Benchmarking structured policies and policy optimization for real-world dexterous object manipulation. IEEE Robotics and Automation Letters (RA-L)

4. Iriondo, A., Lazkano, E., Susperregi, L., **Urain, J.**, Fernandez, A., & Molina, J. (2019). Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning. Applied Sciences

5. Lanini, J., Razavi, H., **Urain, J.**, & Ijspeert, A. (2018). Human intention detection as a multiclass classification problem: Application in physical human–robot interaction while walking. IEEE Robotics and Automation Letters (RA-L)

B.3. Preprints

1. **Urain, J.**, Chalvatzaki, G., & Peters, J. (2023) Deep Generative Models for Motion Planning and Control: Tutorial & Survey

2. Bauer, S., Widmaier, F., Wüthrich, M., Funk, N., **Urain, J.**, Peters, J., ... & Schölkopf, B. (2021). A robot cluster for reproducible research in dexterous manipulation. arXiv preprint arXiv:2109.10957.

B.4. Workshop Papers

1. **Urain, J.**, Funk N., Peters J., & Chalvatzaki G. (2023). Learning Diffusion Models in SE(3) for 6DoF Grasp Pose Generation. Geometric Representations: The Roles of Screw Theory, Lie algebra, & Geometric Algebra in ICRA

2. Carvalho, J., Baierl, M., **Urain, J.**, & Peters, J. (2022). Conditioned Score-Based Models for Learning Collision-Free Trajectory Generation. Workshop on Score-Based Methods in NeurIPS

3. Bauer, S., Wüthrich, M., Widmaier, F., Buchholz, A., Stark, S., Goyal, A., ... & Schölkopf, B. (2022). Real robot challenge: A robotics competition in the cloud. In NeurIPS 2021 Competitions and Demonstrations Track

4. **Urain J.**, Ren, T., Tateo, D., & Peters, J. (2021). Structured Policy Representation: Imposing Stability in arbitrarily conditioned dynamic systems. Robot Learning Workshop in NeurIPS.

C. Curriculum Vitae

Julen Urain. Robotics & Machine Learning Research Scientist
My research interests are in the interplay of generative modeling, geometry, optimization, robotics, planning & control.

Education

2019-2024 **PhD. in Computer Science**. Advisor: Jan Peters. Technische Universität Darmstadt - TUDA

2017 **M.Sc. Thesis.** Advisor: Auke Ijspeert. École Polytechnique Fédérale de Lausanne - EPFL. *GPA – 5.5/6*

2015-2017 **M.Sc. in Automatic Control and Robotics**. Universitat Politècnica de Catalunya - UPC. *GPA – 8.71/10 , Top 3%*

2011-2015 **B.Sc. in Electronical Engineering**. Advisor: Josu Jugo. Universidad del Pais Vasco - UPV. *GPA – 7.3/10*

Research and Work Experience

2022-2023 **Research Intern**. NVIDIA, Robotics Lab. Research in Robot Learning

2019-2023 **Scientific Researcher Staff**. IAS - TU Darmstadt. Research in Robot Learning, Generative Models, Geometry, Optimization, Deep Learning

2017-2018 **Robotics Researcher**. IK4 Research Alliance - Tekniker. Applied Research in Robot Learning.

2017 **Research Intern**. Volkswagen DataLab. In connection with the Deep Learning and Robotics Challenge.

Honors and Awards

2023	**Best paper award in Geometric Representations Workshop at ICRA 2023**. Award earned for the work on SE(3)-Diffusion Models for 6DoF Grasp Generative Models
2023	**R:SS Pioneers**. Selected as a 30 member strong-cohort of top early robotics researchers (%22 acceptance)
2020	**Dexterous Manipulation Real Robot Challenge**. 3rd place in the Max Planck Institute (MPI) Real Robot Dexterous Manipulation Challenge
2017	**Deep Learning and Robotic Challenge**. 1st place of the jury in the VW:DataLab Deep Learning and Robotic Challenge
2017	**MSc. Graduated top of class**. Top 3% in the MSc. in Automatic Control and Robotics at UPC
2015	**Hilbert-Bernays Fellowship**. in relation with Hilbert-Bernays Summer School on Logic and Computation

Funded Projects

2023 **Smart Assistant for Image-guided Needle Insertion**. Hessian.AI

- Role: Project and Technical Leader for TU Darmstadt. PI: Jan Peters

2019-2022 **Safe and effective human robot cooperation towards a better competiveness on current automation lack manufacturing processes(SHAREWORK)**. EU Project - HORIZON 2020

- Role: Project and Technical Leader for TU Darmstadt. PI: Jan Peters

2018-2019 **Flexible, safe and dependable robotic part handling in industrial environments (PICK-PLACE)**. EU Project - HORIZON 2020

- Role: Research Scientist for Tekniker. PI: Iñaki Maurtua

Invited Talks

2023	**An introduction to Energy Based Models and Diffusion Models.** International Workshop of Intelligent Autonomous Learning Systems 2023
2023	**Robot Motion Generative Models.** Dyson Robot Learning Lab
2023	**Robot Motion Generative Models.** The Robot Learning Lab at Imperial College

Teaching Experience

2020-2022	**Robot Learning.** TU Darmstadt. Teaching Assistant
2020-2021	**Robotics Integrated Projects.** TU Darmstadt. Teaching Assistant

Professional Service and Volunteering

Reviewing

Conferences
International Conference on Intelligent Robots (IROS), Conference on Robot Learning (CORL), International Conference on Robotics and Automation (ICRA), Artificial intelligence and Statitistics Conference (AISTATS)

Journals
Robotics and Automation Letters (RA-L), The International Journal of Robotics Research (IJRR)

Other

MOOC on Robot Learning
Design and prepare a MOOC on Robot Learning for the KI-campus platform

Open-Source Software and Datasets

SE(3) DiffusionFields for Grasp and Motion Planning

- Diffusion Models in SE(3) for training 6DoF Grasp Generative Models.
- https://github.com/robotgradient/grasp_diffusion

Stable Vector Fields on Lie Groups

- A method to learn data-driven globally stable dynamics in in Lie Groups to represent task-space robot policies.
- https://github.com/robotgradient/LieFlows

Mentoring and Supervision

2022 **Mark Baierl**. Score-Based Generative Models as Trajectory Priors for Motion Planning. Master Thesis

2022 **Jascha Hellwig**. Residual Reinforcement Learning with Stable Priors. Master Thesis

2021 **Yifei Wang**. Bimanual Control and Learning with Composable Energy Policies. Master Thesis

2021 **Jiawei Huang**. Multi-Objective Reactive Motion Planning in Mobile Manipulators. Master Thesis

2021 **Hanyu Sun**. Can we improve time-series classification with Inverse Reinforcement Learning?. Master Thesis

2021 **Lanmiao Liu**. Detection and Prediction of Human Gestures by Probabilistic Modelling. Master Thesis

2020 **Zhenhui Zhou**. Approximated Policy Search in Black-Box Optimization. Master Thesis

Glossary

APF Artificial Potential Fields. 52–56, 58, 59, 66, 74–76, 92, 123, 149

CD Contrastive Divergence. 21, 23

CE Cross-Entropy. 21, 22

CEP Composable Energy Policies. 5, 50, 56, 59, 62, 65, 66, 70, 73–83, 85–88, 91–93, 121, 123–126, 131–133, 149, 151

CHOMP Covariant Hamiltonian Optimization for Motion Planning. 10, 14, 15

CV Computer Vision. 3

DDPM Denoising Diffusion Probabilistic Models. 17, 18, 24, 26, 111

DMP Dynamic Movement Primitives. 35, 68, 69, 90

DSM Denoising Score Matching. 25, 26, 98, 100, 102

DWA Dynamic Window Approach. 89

EBM Energy Based Models. 3, 5, 7, 13, 14, 17, 21–25, 101, 111, 112

EMD Earth Mover Distance. 104, 105, 150

GAN Generative Adversarial Networks. 16–20, 24, 111

HMM Hidden Markov Model. 119

iFlows ImitationFlows. 5, 29, 33–35

IL Imitation Learning. 28

INN Invertible Neural Networks. 32, 42, 45–49, 148

IOC Inverse Optimal Control. 5, 21, 23

IRL Inverse Reinforcement Learning. 5, 21, 23

KMP Kernelized Movement Primitives. 35

LLM Large Language Models. 112, 113

LMDP linearly-solvable Markov Decision Processes. 89, 123

MAP Maximum a Posteriori. 14

MCMC Markov Chain Monte Carlo. 15, 17, 18, 21, 24, 98, 100, 105, 146

MLE Maximun Likelihood Estimation. 32–34

MP Movement Primitive. 28, 35

MPC Model Predictive Control. 5, 7, 52, 91

MSE Mean Squared Error. 44, 45, 148

NCE Noise Contrastive Estimation. 21, 24

NCSN Noise Conditioned Score Network. 17, 18, 24, 26

NFlow Normalizing Flows. 3, 5, 17–20, 31, 32, 111, 112

NLP Natural Language Processing. 3

ODE Ordinary Differential Equation. 118

ProMP Probabilistic Movement Primitives. 12, 35

RL Reinforcement Learning. 13, 21, 23, 24, 54, 68, 69, 71

RMP Riemannian Motion Policies. 5, 7, 50, 52–56, 58, 59, 62, 65, 66, 74–76, 80, 82, 90–92, 123, 149

RRT Rapidly-exploring Random Trees. 10, 20

SDE Stochastic Differential Equation. 30, 117

SDF Signed Distance Field. 102

SE(3)-DiF SE(3)-DiffusionFields. 5, 101–107, 150

SEDS Stable Estimator of Dynamical Systems. 28

STOMP Stochastic Trajectory Optimization for Motion Planning. 10, 11, 14

SVF Stable Vector Fields. 27, 35, 37, 40, 48, 49, 148

TP-GMM Task Parameterized GMM. 35

VAE Variational Autoencoders. 16–21, 24, 98, 105

List of Figures

List of Tables

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *International Conference on Machine learning*. 2004.

[2] Anurag Ajay et al. “Is Conditional Generative Modeling all you need for Decision Making?” In: *International Conference on Learning Representations*. 2022.

[3] Elie Aljalbout et al. “Learning vision-based reactive policies for obstacle avoidance”. In: *Conference on Robot Learning*. 2021.

[4] Barrett Ames, Jeremy Morgan, and George Konidaris. “Ikflow: Generating diverse inverse kinematics solutions”. In: *IEEE Robotics and Automation Letters* (2022).

[5] Christophe Andrieu and Johannes Thoms. “A tutorial on adaptive MCMC”. In: *Statistics and computing* (2008).

[6] Brenna D Argall et al. “A survey of robot learning from demonstration”. In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[7] Martin Arjovsky and Leon Bottou. “Towards Principled Methods for Training Generative Adversarial Networks”. In: *International Conference on Learning Representations*. 2016.

[8] Hagai Attias. “Planning by probabilistic inference”. In: *International Workshop on Artificial Intelligence and Statistics*. PMLR. 2003, pp. 9–16.

[9] Shikhar Bahl et al. “Neural dynamic policies for end-to-end sensorimotor learning”. In: *Advances in Neural Information Processing Systems* (2020).

[10] Hadi Beik-Mohammadi et al. “Reactive motion generation on learned riemannian manifolds”. In: *Proceedings of Robotics: Science and Systems (R:SS)* (2022).

[11] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. “Task space regions: A framework for pose-constrained manipulation planning”. In: *The International Journal of Robotics Research* 30.12 (2011), pp. 1435–1460.

[12] Michael Betancourt. “A conceptual introduction to Hamiltonian Monte Carlo”. In: *arXiv preprint arXiv:1701.02434* (2017).

[13] Mohak Bhardwaj et al. “STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation”. In: *Conference on Robot Learning*. 2022.

[14] Aude Billard et al. *Survey: Robot programming by demonstration*. Tech. rep. Springrer, 2008.

[15] Damian Boborzi et al. “Learning normalizing flow policies based on highway demonstrations”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 22–29.

[16] Valentin De Bortoli et al. “Riemannian Score-Based Generative Modelling”. In: *Advances in Neural Information Processing Systems*. 2022.

[17] Zdravko I Botev et al. “The cross-entropy method for optimization”. In: *Handbook of statistics*. Vol. 31. Elsevier, 2013, pp. 35–59.

[18] Matthew Botvinick and Marc Toussaint. “Planning as inference”. In: *Trends in cognitive sciences* (2012).

[19] Anthony Brohan et al. “RT-1: Robotics transformer for real-world control at scale”. In: *arXiv preprint arXiv:2212.06817* (2022).

[20] Tom Brown et al. “Language models are few-shot learners”. In: *Advances in Neural Information Processing Systems* (2020).

[21] Andrew Bylard, Riccardo Bonalli, and Marco Pavone. “Composable geometric motion policies using multi-task pullback bundle dynamical systems”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 7464–7470.

[22] Arunkumar Byravan et al. “Space-time functional gradient optimization for motion planning”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 6499–6506.

[23] Sylvain Calinon. “A tutorial on task-parameterized movement learning and retrieval”. In: *Intelligent service robotics* 9.1 (2016), pp. 1–29.

[24] Justin Carpentier et al. “The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives”. In: *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2019, pp. 614–619.

[25] Joao Carvalho et al. “Motion Planning Diffusion: Learning and Planning of Robot Motions with Diffusion Models”. In: *arXiv preprint arXiv:2308.01557* (2023).

[26] Angel X Chang et al. “Shapenet: An information-rich 3D model repository”. In: *arXiv preprint arXiv:1512.03012* (2015).

[27] Ricky TQ Chen et al. “Neural ordinary differential equations”. In: *Advances in Neural Information Processing Systems*. 2018.

[28] Ching-An Cheng et al. “RMPflow: A computational graph for automatic motion policy generation”. In: *International Workshop on the Algorithmic Foundations of Robotics*. 2018.

[29] Cheng Chi et al. “Diffusion policy: Visuomotor policy learning via action diffusion”. In: *Proceedings of Robotics: Science and Systems (R:SS)* (2023).

[30] Gregory Chirikjian and Marin Kobilarov. “Gaussian approximation of non-linear measurement models on Lie groups”. In: *IEEE Conference on Decision and Control*. 2014.

[31] Kurtland Chua et al. “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. In: *Advances in Neural Information Processing Systems* (2018).

[32] Carlo D’Eramo et al. “MushroomRL: Simplifying Reinforcement Learning Research”. In: *Journal of Machine Learning Research* 22.131 (2021), pp. 1–5.

[33] Marco Da Silva, Frédo Durand, and Jovan Popović. “Linear Bellman combination for control of character animation”. In: *ACM SIGGRAPH* (2009).

[34] Murtaza Dalal, Deepak Pathak, and Russ R Salakhutdinov. “Accelerating Robotic Reinforcement Learning via Parameterized Action Primitives”. In: *Advances in Neural Information Processing Systems* (2021).

[35] Christian Daniel et al. “Hierarchical relative entropy policy search”. In: *Journal of Machine Learning Research* 17 (2016), pp. 1–50.

[36] Pieter-Tjerk De Boer et al. “A tutorial on the cross-entropy method”. In: *Annals of operations research* 134.1 (2005), pp. 19–67.

[37] Jan Dentler et al. “A real-time model predictive position control with collision avoidance for commercial low-cost quadrotors”. In: *2016 IEEE conference on control applications (CCA)*. IEEE. 2016, pp. 519–525.

[38] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using Real NVP”. In: *International Conference in Learning Representations*. 2017.

[39] Anca D Dragan, Geoffrey J Gordon, and Siddhartha S Srinivasa. "Learning from experience in manipulation planning: Setting the right goals". In: *Robotics Research: The 15th International Symposium ISRR*. Springer. 2017, pp. 309–326.

[40] Yilun Du, Shuang Li, and Igor Mordatch. "Compositional Visual Generation and Inference with Energy Based Models". In: *Advances in Neural Information Processing Systems*. 2020.

[41] Yilun Du, Toru Lin, and Igor Mordatch. "Model-Based Planning with Energy-Based Models". In: *Conference on Robot Learning*. 2020.

[42] Yilun Du and Igor Mordatch. "Implicit generation and modeling with Energy Based Models". In: *Advances in Neural Information Processing Systems* (2019).

[43] Yilun Du et al. "Learning universal policies via text-guided video generation". In: *arXiv preprint arXiv:2302.00111* (2023).

[44] Yilun Du et al. "Video Language Planning". In: *arXiv preprint arXiv:2310.10625* (2023).

[45] Chris Dyer. "Notes on noise contrastive estimation and negative sampling". In: *arXiv preprint arXiv:1410.8251* (2014).

[46] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. "Acronym: A large-scale grasp dataset based on simulation". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6222–6227.

[47] Tom Erez et al. "An integrated system for real-time model predictive control of humanoid robots". In: *2013 13th IEEE-RAS International conference on humanoid robots (Humanoids)*. IEEE. 2013, pp. 292–299.

[48] L. Falorsi et al. "Reparameterizing distributions on Lie groups". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2019).

[49] Fletcher Fan et al. "Learning stable Koopman embeddings". In: *2022 American Control Conference (ACC)*. IEEE. 2022, pp. 2742–2747.

[50] Chelsea Finn, Sergey Levine, and Pieter Abbeel. "Guided Cost Learning: Deep inverse optimal control via policy optimization". In: *International Conference on Machine Learning*. 2016.

[51] Chelsea Finn et al. "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models". In: *arXiv preprint arXiv:1611.03852* (2016).

[52] Pete Florence et al. "Implicit behavioral cloning". In: *Conference on Robot Learning*. 2022.

[53] Pete Florence et al. “Implicit behavioral cloning”. In: *Conference on Robot Learning*. 2022.

[54] Joan Fontanals et al. “Integrated grasp and motion planning using independent contact regions”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 887–893.

[55] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. “The dynamic window approach to collision avoidance”. In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33.

[56] Justin Fu, Katie Luo, and Sergey Levine. “Learning Robust Rewards with Adverserial Inverse Reinforcement Learning”. In: *International Conference on Learning Representations*. 2018.

[57] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *International Conference on Machine Learning*. 2018.

[58] Niklas Funk et al. “Benchmarking structured policies and policy optimization for real-world dexterous object manipulation”. In: *IEEE Robotics and Automation Letters* 7.1 (2021), pp. 478–485.

[59] Carlos E Garcia, David M Prett, and Manfred Morari. “Model Predictive Control: Theory and practice—A survey”. In: *Automatica* 25.3 (1989), pp. 335–348.

[60] Shuzhi Sam Ge and Yun J Cui. “Dynamic motion planning for mobile robots using potential field method”. In: *Autonomous robots* 13.3 (2002), pp. 207–222.

[61] Mevlana C Gemici, Danilo Rezende, and Shakir Mohamed. “Normalizing flows on Riemannian manifolds”. In: *arXiv preprint arXiv:1611.02304* (2016).

[62] Yiran Geng et al. “RLAfford: End-to-End Affordance Learning for Robotic Manipulation”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 5880–5886.

[63] Theophile Gervet et al. “Act3D: Infinite resolution action detection transformer for robotic manipulation”. In: *arXiv preprint arXiv:2306.17817* (2023).

[64] Josiah Willard Gibbs. *Elementary principles in statistical mechanics: developed with especial reference to the rational foundations of thermodynamics*. C. Scribner’s sons, 1902.

[65] Nikolaos Gkanatsios et al. “Energy-based Models are Zero-Shot Planners for Compositional Scene Rearrangement”. In: *RSS 2023 Workshop on Learning for Task and Motion Planning*. 2023.

[66] Dwaraknath Gnaneshwar et al. “Score-based generative models for molecule generation”. In: *arXiv preprint arXiv:2203.04698* (2022).

[67] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems*. 2014.

[68] Ankit Goyal et al. “RVT: Robotic View Transformer for 3D Object Manipulation”. In: *arXiv preprint arXiv:2306.14896* (2023).

[69] Will Grathwohl et al. “FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models”. In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=rJxgknCcK7`.

[70] Michael Gutmann and Aapo Hyvärinen. “Noise-contrastive estimation: A new estimation principle for unnormalized statistical models”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 297–304.

[71] Huy Ha, Pete Florence, and Shuran Song. “Scaling Up and Distilling Down: Language-Guided Robot Skill Acquisition”. In: *Conference on Robot Learning*. 2023.

[72] Tuomas Haarnoja et al. “Composable deep reinforcement learning for robotic manipulation”. In: *IEEE International Conference on Robotics and Automation*. IEEE. 2018, pp. 6244–6251.

[73] Tuomas Haarnoja et al. “Reinforcement learning with deep Energy-Based Policies”. In: *International Conference on Machine Learning*. 2017.

[74] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International Conference on Machine Learning*. 2018.

[75] Janik Hager et al. “GraspME-Grasp Manifold Estimator”. In: *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*. IEEE. 2021, pp. 626–632.

[76] Micha Hersch et al. “Dynamical system modulation for robot learning via kinesthetic demonstrations”. In: *IEEE Transactions on Robotics* 24.6 (2008), pp. 1463–1467.

[77] Carolina Higuera, Byron Boots, and Mustafa Mukadam. “Learning to Read Braille: Bridging the Tactile Reality Gap with Diffusion Models”. In: *arXiv preprint arXiv:2304.01182* (2023).

[78] Geoffrey E Hinton. “Training products of experts by minimizing contrastive divergence”. In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[79] Jonathan Ho and Stefano Ermon. “Generative Adversarial Imitation Learning”. In: *Advances in Neural Information Processing Systems*. 2016.

[80] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in Neural Information Processing Systems* (2020).

[81] Jonathan Ho and Tim Salimans. “Classifier-Free Diffusion Guidance”. In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. 2021.

[82] Taylor Howell et al. “Predictive sampling: Real-time behaviour synthesis with mujoco”. In: *arXiv preprint arXiv:2212.00541* (2022).

[83] Chin-Wei Huang et al. “Riemannian diffusion models”. In: *Advances in Neural Information Processing Systems* (2022).

[84] Siyuan Huang et al. “Diffusion-based generation, optimization, and planning in 3d scenes”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 16750–16761.

[85] Yanlong Huang et al. “Kernelized Movement Primitives”. In: *The International Journal of Robotics Research* 38.7 (2019), pp. 833–852.

[86] Yanlong Huang et al. “Toward orientation learning and adaptation in cartesian space”. In: *IEEE Transactions on Robotics* (2020).

[87] Brian Ichter, James Harrison, and Marco Pavone. “Learning sampling distributions for robot motion planning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7087–7094.

[88] Auke Jan Ijspeert. “Central pattern generators for locomotion control in animals and robots: a review”. In: *Neural networks* 4 (2008).

[89] Ander Iriondo et al. “Pick and place operations in logistics using a mobile manipulator controlled with deep reinforcement learning”. In: *Applied Sciences* 9.2 (2019), p. 348.

[90] Michael Janner et al. “Planning with Diffusion for Flexible Behavior Synthesis”. In: *International Conference on Machine Learning*. 2022.

[91] Edwin T Jaynes. “Information theory and statistical mechanics”. In: *Physical review* 106.4 (1957), p. 620.

[92] Zhenyu Jiang et al. “Synergies between affordance and geometry: 6-dof grasp detection via implicit representations”. In: *Proceedings of Robotics: Science and Systems (R:SS)* (2021).

[93] Tobias Johannink et al. “Residual reinforcement learning for robot control”. In: *International Conference on Robotics and Automation*. IEEE. 2019, pp. 6023–6029.

[94] Leslie Pack Kaelbling and Tomás Lozano-Pérez. “Hierarchical task and motion planning in the now”. In: *IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 1470–1477.

[95] Leslie Pack Kaelbling and Tomás Lozano-Pérez. “Integrated task and motion planning in belief space”. In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1194–1227.

[96] Mrinal Kalakrishnan et al. “Learning objective functions for manipulation”. In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1331–1336.

[97] Mrinal Kalakrishnan et al. “STOMP: Stochastic trajectory optimization for motion planning”. In: *IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4569–4574.

[98] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. “Dall-e-bot: Introducing web-scale diffusion models to robotics”. In: *IEEE Robotics and Automation Letters* (2023).

[99] Daniel Kappler et al. “Real-time perception meets reactive motion generation”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1864–1871.

[100] Lydia E Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[101] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of NAACL-HLT*. 2019.

[102] S Mohammad Khansari-Zadeh and Aude Billard. “Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions”. In: *Robotics and Autonomous Systems* (2014).

[103] S Mohammad Khansari-Zadeh and Aude Billard. “Learning stable nonlinear dynamical systems with gaussian mixture models”. In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957.

[104] Oussama Khatib. “A unified approach for motion and force control of robot manipulators: The operational space formulation”. In: *IEEE Journal on Robotics and Automation* 3.1 (1987), pp. 43–53.

[105] Oussama Khatib. “Real-time obstacle avoidance for manipulators and mobile robots”. In: *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.

[106] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *Conference on Learning Representations, ICLR 2014*. Ed. by Yoshua Bengio and Yann LeCun. 2014.

[107] Dorothea Koert et al. “Demonstration based trajectory optimization for generalizable robot motions”. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2016, pp. 515–522.

[108] J Zico Kolter and Gaurav Manek. “Learning stable deep dynamics models”. In: *Advances in Neural Information Processing Systems*. 2019.

[109] Leonidas Koutras and Zoe Doulgeri. “A correct formulation for the orientation dynamic movement primitives for robot control in the cartesian space”. In: *Conference on Robot Learning*. 2020.

[110] Rahul G Krishnan, Uri Shalit, and David Sontag. “Structured inference networks for nonlinear state space models”. In: *AAAI conference on artificial intelligence*. 2017.

[111] James J Kuffner and Steven M LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2000.

[112] Thibaut Kulak, Joao Silvério, and Sylvain Calinon. “Fourier movement primitives: an approach for learning rhythmic robot skills from demonstrations”. In: *Proceedings of Robotics: Science and Systems (R:SS)*. 2020.

[113] Fabien Lagriffoul et al. “Efficiently combining task and motion planning using geometric constraints”. In: *The International Journal of Robotics Research* 33.14 (2014), pp. 1726–1747.

[114] Tin Lai and Fabio Ramos. “Plannerflows: Learning motion samplers with normalising flows”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021.

[115] Tin Lai et al. “Parallelised diffeomorphic sampling-based motion planning”. In: *Conference on Robot Learning*. 2022.

[116] Alexander Lambert et al. “Stein Variational Model Predictive Control”. In: *Conference on Robot Learning*. 2021.

[117] Jean-Claude Latombe. *Robot motion planning*. Kluwer Academic Publishers, Boston, 1991.

[118] Steven M LaValle. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[119] Steven M LaValle et al. “Rapidly-exploring Random Trees: A new tool for path planning”. In: (1998).

[120] A.T. Le et al. “Accelerating Motion Planning via Optimal Transport”. In: *Advances in Neural Information Processing Systems*. 2023.

[121] Yann LeCun et al. “A tutorial on energy-based learning”. In: *Predicting structured data* 1.0 (2006).

[122] John M. Lee. “Introduction to Smooth Manifolds”. In: *Springer-Verlag*. 2006.

[123] Michelle A Lee et al. “Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8943–8950.

[124] Teguh Santoso Lembono et al. “Generative adversarial network to learn valid distributions of robot configurations for inverse kinematics and constrained motion planning”. In: *CoRR, abs/2011.05717* (2020).

[125] Andre Lemme et al. “Neurally Imprinted Stable Vector Fields”. In: *European Symposium on Artificial Neural Networks*. 2013.

[126] Sergey Levine. “Reinforcement learning and control as probabilistic inference: Tutorial and review”. In: *arXiv preprint arXiv:1805.00909* (2018).

[127] Anqi Li et al. “RMP2: A Structured Composable Policy Class for Robot Learning”. In: *Robotics Science and Systems (R:SS)* (2021).

[128] Ge Li et al. “ProDMP: A Unified Perspective on Dynamic and Probabilistic Movement Primitives”. In: *IEEE Robotics and Automation Letters* (2023).

[129] Hongzhuo Liang et al. “PointnetGPD: Detecting grasp configurations from point sets”. In: *International Conference on Robotics and Automation*. 2019.

[130] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[131] Puze Liu et al. “Regularized Deep Signed Distance Fields for Reactive Motion Generation”. In: *IEEE International Conference on Intelligent Robots and Systems* (2022).

[132] Puze Liu et al. “Robot reinforcement learning on the constraint manifold”. In: *Conference on Robot Learning*. 2022.

[133] Weiyu Liu et al. “StructDiffusion: Object-centric diffusion for semantic rearrangement of novel objects”. In: *Proceedings of Robotics: Science and Systems (R:SS)* (2023).

[134] Aaron Lou et al. “Neural manifold ordinary differential equations”. In: *Advances in Neural Information Processing Systems* (2020).

[135] Xibai Lou, Yang Yang, and Changhyun Choi. “Collision-aware target-driven object grasping in constrained environments”. In: *IEEE International Conference on Robotics and Automation*. 2021.

[136] Tobias Löw et al. “PROMPT: Probabilistic Motion Primitives based Trajectory Planning.” In: *Proceedings of Robotics: Science and Systems (R:SS)*. 2021.

[137] Qingkai Lu et al. “Planning multi-fingered grasps as probabilistic inference in a learned deep network”. In: *Robotics Research*. Springer, 2020, pp. 455–472.

[138] Calvin Luo. “Understanding diffusion models: A unified perspective”. In: *arXiv preprint arXiv:2208.11970* (2022).

[139] Michael Lutter et al. “Value Iteration in Continuous Actions, States and Time”. In: *International Conference on Machine Learning*. 2021.

[140] Corey Lynch et al. “Learning latent plans from play”. In: *Conference on robot learning*. 2020.

[141] Xiaobai Ma, Jayesh K Gupta, and Mykel J Kochenderfer. “Normalizing flow policies for multi-agent systems”. In: *International Conference on Decision and Game Theory for Security*. Springer. 2020, pp. 277–296.

[142] Jeffrey Mahler et al. “Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics”. In: *Proceedings of Robotics: Science and Systems (R:SS)* (2017).

[143] Arjun Majumdar et al. “Where are we in the search for an Artificial Visual Cortex for Embodied Intelligence?” In: *Workshop on Reincarnating Reinforcement Learning at ICLR*. 2023.

[144] Viktor Makoviychuk et al. “Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning”. In: *Advances in Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021.

[145] Xuerong Mao. *Stochastic differential equations and applications*. Elsevier, 2007.

[146] Emile Mathieu and Maximilian Nickel. “Riemannian continuous normalizing flows”. In: *Advances in Neural Information Processing Systems* (2020).

[147] Takahiro Miki et al. “Learning robust perceptive locomotion for quadrupedal robots in the wild”. In: *Science Robotics* 7.62 (2022), eabk2822.

[148] Takeru Miyato et al. “Spectral Normalization for Generative Adversarial Networks”. In: *International Conference on Learning Representations*. 2018.

[149] Kaichun Mo et al. “Where2act: From pixels to actions for articulated 3D objects”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 6813–6823.

[150] Mehdi Mohammadi, Ala Al-Fuqaha, and Jun-Seok Oh. “Path planning in support of smart mobility applications using generative adversarial networks”. In: *IEEE International Conference on Internet of Things*. 2018.

[151] Manfred Morari and Jay H Lee. “Model predictive control: past, present and future”. In: *Computers & Chemical Engineering* 23.4-5 (1999), pp. 667–682.

[152] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. “6-dof graspnet: Variational grasp generation for object manipulation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 2901–2910.

[153] Mustafa Mukadam, Xinyan Yan, and Byron Boots. “Gaussian process motion planning”. In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 9–15.

[154] Mustafa Mukadam et al. “Continuous-time Gaussian process motion planning via probabilistic inference”. In: *The International Journal of Robotics Research* 37.11 (2018), pp. 1319–1340.

[155] Katharina Mülling et al. “Learning to select and generalize striking movements in robot table tennis”. In: *The International Journal of Robotics Research* 32.3 (2013), pp. 263–279.

[156] Adithyavairavan Murali et al. “6-dof grasping for target-driven object manipulation in clutter”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 6232–6238.

[157] Radford M Neal et al. “MCMC using Hamiltonian dynamics”. In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.

[158] Klaus Neumann, Andre Lemme, and Jochen J Steil. “Neural learning of stable dynamical systems based on data-driven Lyapunov candidates”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 1216–1222.

[159] Klaus Neumann and Jochen J Steil. “Learning robot motions with stable dynamical systems under diffeomorphic transformations”. In: *Robotics and Autonomous Systems* 70 (2015), pp. 1–15.

[160] Toshiyuki Ohtsuka. “A continuation/GMRES method for fast computation of nonlinear receding horizon control”. In: *Automatica* 40.4 (2004), pp. 563–574.

[161] Joaquim Ortiz-Haro et al. "Structured deep generative models for sampling on constraint manifolds in sequential manipulation". In: *Conference on Robot Learning*. 2022.

[162] Takayuki Osa. "Motion planning by learning the solution manifold in trajectory optimization". In: *The International Journal of Robotics Research* (2022).

[163] Takayuki Osa et al. "An algorithmic perspective on imitation learning". In: *Foundations and Trends® in Robotics* 7.1-2 (2018), pp. 1–179.

[164] George Papamakarios, Theo Pavlakou, and Iain Murray. "Masked Autoregressive Flow for density estimation". In: *Advances in Neural Information Processing Systems*. 2017.

[165] George Papamakarios et al. "Normalizing flows for probabilistic modeling and inference". In: *arXiv preprint arXiv:1912.02762* (2019).

[166] Alexandros Paraschos et al. "Probabilistic Movement Primitives". In: *Advances in Neural Information Processing Systems*. 2013.

[167] Alexandros Paraschos et al. "Using probabilistic movement primitives in robotics". In: *Autonomous Robots* 42.3 (2018), pp. 529–551.

[168] Jeong Joon Park et al. "DeepSDF: Learning continuous signed distance functions for shape representation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 165–174.

[169] Wooram Park et al. "Diffusion-based motion planning for a nonholonomic flexible needle model". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE. 2005, pp. 4600–4605.

[170] Andreas ten Pas et al. "Grasp pose detection in point clouds". In: *The International Journal of Robotics Research* 36.13-14 (2017), pp. 1455–1473.

[171] P. Pastor et al. "Online movement adaptation based on previous sensor experiences". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011.

[172] Xue Bin Peng et al. "AMP: Adversarial motion priors for stylized physics-based character control". In: *ACM Transactions on Graphics (ToG)* (2021).

[173] Xue Bin Peng et al. "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills". In: *ACM Transactions On Graphics (TOG)* 37.4 (2018), pp. 1–14.

[174] Xue Bin Peng et al. “MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies”. In: *Advances in Neural Information Processing Systems* (2019).

[175] Nicolas Perrin and Philipp Schlehuber-Caissier. “Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems”. In: *Systems & Control Letters* 96 (2016), pp. 51–59.

[176] Karl Pertsch, Youngwoon Lee, and Joseph Lim. “Accelerating Reinforcement Learning with Learned Skill Priors”. In: *Conference on Robot Learning*. 2021.

[177] Jan Peters and Stefan Schaal. “Reinforcement learning by reward-weighted regression for operational space control”. In: *International Conference on Machine Learning*. 2007.

[178] Luis Pineda et al. “Theseus: A library for differentiable nonlinear optimization”. In: *Advances in Neural Information Processing Systems* (2022).

[179] Ph Poignet and Maxime Gautier. “Nonlinear model predictive control of a robot manipulator”. In: *6th International workshop on advanced motion control. Proceedings (Cat. No. 00TH8494)*. IEEE. 2000, pp. 401–406.

[180] Dean A Pomerleau. “Alvinn: An autonomous land vehicle in a neural network”. In: *Advances in Neural Information Processing Systems* (1988).

[181] Ben Poole et al. “DreamFusion: Text-to-3D using 2D Diffusion”. In: *International Conference on Learning Representations*. 2022.

[182] Sean Quinlan and Oussama Khatib. “Elastic bands: Connecting path planning and control”. In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*. IEEE. 1993, pp. 802–807.

[183] Krisnawan Rahardja and Akio Kosaka. “Vision-based bin-picking: Recognition and localization of multiple complex objects using simple visual cues”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems.* 1996.

[184] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. “RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion.” In: *Proceedings of Robotics: Science and Systems (R:SS)*. 2018.

[185] Aditya Ramesh et al. “Hierarchical text-conditional image generation with clip latents”. In: *arXiv preprint arXiv:2204.06125* (2022).

[186] Krishan Rana et al. “Residual skill policies: Learning an adaptable skill-based action space for reinforcement learning for robotics”. In: *Conference on Robot Learning*. 2023.

[187] Muhammad Asif Rana et al. “Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems”. In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 630–639.

[188] Nathan Ratliff et al. “CHOMP: Gradient optimization techniques for efficient motion planning”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 489–494.

[189] Nathan D Ratliff et al. “Generalized nonlinear and finsler geometry for robotics”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 10206–10212.

[190] Nathan D Ratliff et al. “Optimization fabrics”. In: *arXiv preprint arXiv:2008.02399* (2020).

[191] Nathan D Ratliff et al. “Riemannian motion policies”. In: *arXiv preprint arXiv:1801.02854* (2018).

[192] Harish Ravichandar and Ashwin Dani. “Learning contracting nonlinear dynamics from human demonstration for robot motion planning”. In: *ASME, Dynamic Systems and Control Conference*. 2015.

[193] Harish Ravichandar et al. “Recent advances in robot learning from demonstration”. In: *Annual review of control, robotics, and autonomous systems* (2020).

[194] Harish Chaandar Ravichandar, Iman Salehi, and Ashwin P Dani. “Learning Partially Contracting Dynamical Systems from Demonstrations.” In: *Conference on Robot Learning*. 2017.

[195] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. “On stochastic optimal control and reinforcement learning by approximate inference”. In: *Proceedings of Robotics: Science and Systems (R:SS)* (2012).

[196] Scott Reed et al. “A Generalist Agent”. In: *Transactions on Machine Learning Research* (2022).

[197] Moritz Reuss et al. “Goal-conditioned imitation learning using score-based diffusion policies”. In: *arXiv preprint arXiv:2304.02532* (2023).

[198] Danilo Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: *International Conference on Machine Learning*. 2015.

[199] Danilo Jimenez Rezende et al. “Normalizing flows on Tori and Spheres”. In: *International Conference on Machine Learning*. 2020.

[200] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer. 2015, pp. 234–241.

[201] Peter J Rossky, Jimmie D Doll, and Harold L Friedman. “Brownian dynamics as smart Monte Carlo simulation”. In: *The Journal of Chemical Physics* 69.10 (1978), pp. 4628–4633.

[202] Leonel Rozo and Vedant Dave. “Orientation Probabilistic Movement Primitives on Riemannian Manifolds”. In: *Conference on Robot Learning*. 2022.

[203] Nataniel Ruiz et al. “Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.

[204] Chitwan Saharia et al. “Photorealistic text-to-image diffusion models with deep language understanding”. In: *Advances in Neural Information Processing Systems* (2022).

[205] Saeed Saremi et al. “Deep energy estimator networks”. In: *arXiv preprint arXiv:1805.08306* (2018).

[206] Matteo Saveriano, Fares J Abu-Dakka, and Ville Kyrki. “Learning stable robotic skills on Riemannian manifolds”. In: *Robotics and Autonomous Systems* 169 (2023), p. 104510.

[207] Stefan Schaal. “Dynamic movement primitives-a framework for motor control in humans and humanoid robotics”. In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[208] Stefan Schaal. “Is imitation learning the route to humanoid robots?” In: *Trends in cognitive sciences* 3.6 (1999), pp. 233–242.

[209] Stefan Schaal. “Learning from demonstration”. In: *Advances in Neural Information Processing Systems*. 1997.

[210] Stefan Schaal, Christopher G Atkeson, and Sethu Vijayakumar. “Scalable techniques from nonparametric statistics for real time robot learning”. In: *Applied Intelligence* 17.1 (2002), pp. 49–60.

[211] Charles Schaff and Matthew R Walter. “Residual Policy Learning for Shared Autonomy”. In: *Proceedings of Robotics: Science and Systems (R:SS)*. 2020.

[212] John Schulman et al. “Motion planning with sequential convex optimization and convex collision checking”. In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270.

[213] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[214] Muhammet Yunus Seker et al. “Conditional Neural Movement Primitives.” In: *Proceedings of Robotics: Science and Systems (R:SS)*. 2019.

[215] Seiji Shaw, Ben Abbatematteo, and George Konidaris. “RMPs for safe impedance control in contact-rich manipulation”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2022.

[216] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. “CLIPort: What and where pathways for robotic manipulation”. In: *Conference on Robot Learning*. 2022.

[217] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. “Perceiver-actor: A multi-task transformer for robotic manipulation”. In: *Conference on Robot Learning*. 2023.

[218] Tom Silver et al. “Residual policy learning”. In: *arXiv preprint arXiv:1812.06298* (2018).

[219] Anthony Simeonov et al. “Neural descriptor fields: SE(3)-equivariant object representations for manipulation”. In: *International Conference on Robotics and Automation*. IEEE. 2022.

[220] Anthony Simeonov et al. “Shelving, Stacking, Hanging: Relational Pose Diffusion for Multi-modal Rearrangement”. In: *Conference on Robot Learning*. 2023.

[221] Vikas Sindhwani, Stephen Tu, and Mohi Khansari. “Learning contracting vector fields for stable imitation learning”. In: *arXiv preprint arXiv:1804.04878* (2018).

[222] Jascha Sohl-Dickstein et al. “Deep unsupervised learning using nonequilibrium thermodynamics”. In: *International Conference on Machine Learning*. 2015.

[223] Joan Sola, Jeremie Deray, and Dinesh Atchuthan. “A micro Lie theory for state estimation in robotics”. In: *arXiv preprint arXiv:1812.01537* (2018).

[224] Yang Song and Stefano Ermon. “Generative modeling by estimating gradients of the data distribution”. In: *Advances in Neural Information Processing Systems* (2019).

[225] Yang Song and Stefano Ermon. “Improved techniques for training score-based generative models”. In: *Advances in Neural Information Processing Systems* (2020).

[226] Yang Song and Diederik P Kingma. “How to train your Energy-Based Models”. In: *arXiv preprint arXiv:2101.03288* (2021).

[227] Yang Song et al. “Score-based generative modeling through stochastic differential equations”. In: *arXiv preprint arXiv:2011.13456* (2020).

[228] Martin Sundermeyer et al. “Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 13438–13444.

[229] Richard S Sutton, Doina Precup, and Satinder Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[230] Akinori Tanaka. “Discriminator optimal transport”. In: *Advances in Neural Information Processing Systems* (2019).

[231] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. “A Boolean Task Algebra for Reinforcement Learning”. In: *Advances in Neural Information Processing Systems* (2020).

[232] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. “Reinforcement learning of motor skills in high dimensions: A path integral approach”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2397–2403.

[233] Emanuel Todorov. “Compositionality of optimal control laws”. In: *Advances in Neural Information Processing Systems* (2009).

[234] Marc Toussaint. “Robot trajectory optimization using approximate inference”. In: *International Conference on Machine Learning*. 2009.

[235] Marc Toussaint and Christian Goerick. “A Bayesian view on motor control and planning”. In: *From Motor Learning to Interaction Learning in Robots*. Springer, 2010, pp. 227–252.

[236] Hugo Touvron et al. “Llama 2: Open foundation and fine-tuned chat models”. In: *arXiv preprint arXiv:2307.09288* (2023).

[237] Aleš Ude et al. “Orientation in Cartesian space dynamic movement primitives”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014. DOI: `10.1109/ICRA.2014.6907291`.

[238] Jonas Umlauft and Sandra Hirche. “Learning stable stochastic nonlinear dynamical systems”. In: *International Conference on Machine Learning*. 2017.

[239] Julen Urain, Davide Tateo, and Jan Peters. “Learning stable vector fields on Lie groups”. In: *Robotics and Automation Letters (RA-L)*. 2022.

[240] Julen Urain et al. “Composable Energy Policies for Reactive Motion Generation and Reinforcement Learning”. In: *Proceedings of Robotics: Science and Systems (R:SS)*. 2021.

[241] Julen Urain et al. “Composable energy policies for reactive motion generation and reinforcement learning”. In: *The International Journal of Robotics Research (IJRR)* (2023).

[242] Julen Urain et al. “ImitationFlows: Learning Deep Stable Stochastic Dynamic Systems by Normalizing Flows”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2020.

[243] Julen Urain et al. “Learning Implicit Priors for Motion Optimization”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 7672–7679. DOI: 10.1109/IROS47612.2022.9981264.

[244] Julen Urain et al. “SE(3)-DiffusionFields: Learning cost functions for joint grasp and motion optimization through diffusion”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2023).

[245] Nikolaus Vahrenkamp et al. “Integrated grasp and motion planning”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2883–2888.

[246] Jur Van Den Berg et al. “Reciprocal n-body collision avoidance”. In: *Robotics research*. Springer, 2011, pp. 3–19.

[247] Benjamin Van Niekerk et al. “Composing value functions in reinforcement learning”. In: *International Conference on Machine Learning*. 2019.

[248] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in Neural Information Processing Systems* (2017).

[249] Pascal Vincent. “A connection between score matching and denoising autoencoders”. In: *Neural computation* 23.7 (2011), pp. 1661–1674.

[250] Lirui Wang, Yu Xiang, and Dieter Fox. “Manipulation trajectory optimization with online grasp synthesis and selection”. In: *Proceedings of Robotics: Science and Systems (R:SS)* (2019).

[251] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. “Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning”. In: *International Conference on Learning Representations*. 2022.

[252] Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. “Improving exploration in soft-actor-critic with normalizing flows policies”. In: *arXiv preprint arXiv:1906.02771* (2019).

[253] Bowen Wen et al. "SE(3)-Tracknet: Data-driven 6D pose tracking by calibrating image residuals in synthetic domains". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 10367–10373.

[254] Thomas Weng et al. "Neural grasp distance fields for robot manipulation". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 1814–1821.

[255] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. "Model predictive path integral control: From theory to parallel computation". In: *Journal of Guidance, Control, and Dynamics* 40.2 (2017), pp. 344–357.

[256] Markus Wulfmeier, Dominic Zeng Wang, and Ingmar Posner. "Maximum entropy deep inverse reinforcement learning." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2016.

[257] Zhou Xian et al. "Unifying Diffusion Models with Action Detection Transformers for Multi-task Robotic Manipulation". In: *Conference on Robot Learning*. 2023.

[258] Mandy Xie et al. "Geometric Fabrics for the Acceleration-based Design of Robotic Motion". In: *arXiv preprint arXiv:2010.14750* (2020).

[259] Yiheng Xie et al. "Neural fields in visual computing and beyond". In: *Computer Graphics Forum*. Vol. 41. 2. Wiley Online Library. 2022, pp. 641–676.

[260] Mengyuan Yan et al. "Learning probabilistic multi-modal actor models for vision-based robotic grasping". In: *International Conference on Robotics and Automation (ICRA)*. 2019.

[261] Xinchen Yan et al. "Learning 6-dof grasping interaction via deep geometry-aware 3D representations". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 3766–3773.

[262] Zhutian Yang et al. "Compositional Diffusion-Based Continuous Constraint Solvers". In: *Conference on Robot Learning* (2023).

[263] J.S. Yuan. "Closed-loop manipulator control using quaternion feedback". In: *IEEE Journal on Robotics and Automation* (1988). DOI: 10.1109/56.809.

[264] Ekim Yurtsever et al. "A survey of autonomous driving: Common practices and emerging technologies". In: *IEEE access* 8 (2020), pp. 58443–58469.

[265] Martijn JA Zeestraten et al. "An approach for imitation learning on Riemannian manifolds". In: *IEEE Robotics and Automation Letters (RA-L)* (2017).

[266] Andy Zeng et al. “Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching”. In: *The International Journal of Robotics Research* (2022).

[267] Andy Zeng et al. “Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language”. In: *International Conference on Learning Representations*. 2022.

[268] Andy Zeng et al. “Transporter networks: Rearranging the visual world for robotic manipulation”. In: *Conference on Robot Learning*. 2021.

[269] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. “Adding conditional control to text-to-image diffusion models”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023.

[270] Tianyi Zhang, Jiankun Wang, and Max Q-H Meng. “Generative adversarial network based heuristics for sampling-based path planning”. In: *IEEE/CAA Journal of Automatica Sinica* (2021).

[271] Ziyuan Zhong et al. “Guided conditional diffusion for controllable traffic simulation”. In: *International Conference on Robotics and Automation (ICRA)*. 2023.

[272] Brian D Ziebart, J Andrew Bagnell, and Anind K Dey. “Modeling interaction via the principle of maximum causal entropy”. In: *International Conference on Machine Learning*. 2010.

[273] Brian D Ziebart et al. “Maximum Entropy Inverse Reinforcement Learning.” In: *AAAI*. 2008.

[274] Matt Zucker et al. “CHOMP: Covariant hamiltonian optimization for motion planning”. In: *The International journal of robotics research* 32.9-10 (2013), pp. 1164–1193.