SAFE REINFORCEMENT LEARNING FOR ROBOTICS

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN

von der

KIT-Fakultät für Informatik

des

Karlsruher Instituts für Technologie (KIT)

genehmigte

DISSERTATION

von

ABHISHEK PADALKAR

aus Ruichhattishi, India

Tag der mündlichen Prüfung:	04.02.2026
1. Referent:	Prof. Dr. Gerhard Neumann
2. Referentin:	Prof. Dr. Dongheui Lee

ABSTRACT

Despite its proven potential to learn tasks from scratch across diverse domains, the requirement for a large number of training episodes, coupled with safety concerns, remains a major limiting factor for applying Reinforcement Learning (RL) in robotics. Learning skills directly on real robots is time-consuming, causes wear and tear, and risks damage to both the robot and its environment due to unsafe exploratory actions. Although learning skills in simulation and transferring them to real robots has shown promise, its success is limited by the reality gap between simulation and the physical world. This challenge is particularly pronounced for tasks involving contact with the environment, as contact dynamics are difficult to model and simulate accurately. Moreover, designing realistic simulations is itself a tedious and time-consuming process.

This work addresses these challenges by developing methods that incorporate task knowledge and constraints to guide RL, thereby improving sample efficiency and safety during exploration and execution. Unknown task knowledge and actions are learned by an RL agent exploring within constrained environments, enabling learning directly on real robots, especially for contact-rich tasks. The first method in this work, Reinforcement Learning with Shared Control Templates (RL-SCT) leverages hand-designed task knowledge and safety constraints to reduce state and action spaces, enabling safe and sample-efficient learning on real robots. The second method, Kernelized Guided Reinforcement Learning (KGRL) enhances this by integrating human demonstrations to initialize policies and enforce constraints via Linearly Constrained Null-Space Kernelized Movement Primitives (LC-NS-KMP). KGRL improves task knowledge extraction over RL-SCT by deriving task completion strategies directly from demonstrations. It introduces state-dependent, uncertainty-aware exploration noise derived from demonstration variance and imposes linear inequality constraints on the robot's state to ensure safe guided exploration. The third method, Smooth Kernelized Guided Reinforcement Learning (sKGRL) extends KGRL by incorporating smooth exploration strategy, utilizing smooth kernels (e.g., radial basis functions) and robot state history to minimize abrupt, high-acceleration actions, further enhancing safety. Unlike classical RL, our methods enforce constraints explicitly, allowing for simpler reward functions without cost terms for constraint violations. Together, these contributions enable efficient and safe reinforcement learning directly on real robots, advancing RL's applicability for complex, contact-rich robotic manipulation tasks.

Evaluated on real robot and simulated environments, these frameworks outperform existing classical RL methods in safety and efficiency metrics, such as reduced spillage of liquid, collisions, interaction forces and more importantly the number of training episodes. By incorporating shared control, demonstration-guided learning, and smooth exploration, this work offers a holistic approach to real-world robotic reinforcement learning. We offer extensive evaluation of these methods to showcase their ability to handle diverse manipulation tasks. The findings in this work provide novel frameworks for scalable, safe RL, advancing the field toward practical deployment in complex, contact-rich robotic manipulation tasks.

ZUSAMMENFASSUNG

Trotz seines nachgewiesenen Potenzials, Aufgaben in verschiedenen Domänen von Grund auf zu erlernen, bleibt die Notwendigkeit einer großen Anzahl von Trainingsepisoden in Kombination mit Sicherheitsbedenken ein wesentlicher limitierender Faktor für die Anwendung von Reinforcement Learning (RL) in der Robotik. Das Erlernen von Fähigkeiten direkt auf realen Robotern ist zeitaufwendig, verursacht Verschleiß und birgt das Risiko von Schäden sowohl am Roboter als auch an seiner Umgebung durch unsichere explorative Aktionen. Obwohl das Erlernen von Fähigkeiten in Simulationen und deren Übertragung auf reale Roboter vielversprechend ist, wird der Erfolg durch die sogenannte „Reality Gap" – also die Diskrepanz zwischen Simulation und physischer Welt – eingeschränkt. Diese Herausforderung ist besonders ausgeprägt bei Aufgaben, die den Kontakt mit der Umgebung erfordern, da sich Kontaktdynamiken nur schwer genau modellieren und simulieren lassen. Darüber hinaus ist die Erstellung realistischer Simulationen selbst ein mühsamer und zeitintensiver Prozess.

Diese Arbeit adressiert diese Herausforderungen, indem Methoden entwickelt werden, die Aufgabenwissen und Einschränkungen in das RL einbeziehen, um so die Stichprobeneffizienz und Sicherheit während der Exploration und Ausführung zu verbessern. Unbekanntes Aufgabenwissen und Aktionen werden von einem RL-Agenten erlernt, der in eingeschränkten Umgebungen exploriert, was ein direktes Lernen auf realen Robotern ermöglicht – insbesondere bei kontaktintensiven Aufgaben.

Die erste Methode dieser Arbeit, Reinforcement Learning with Shared Control Templates (RL-SCT), nutzt handgestaltetes Aufgabenwissen und Sicherheitsbeschränkungen, um Zustands- und Aktionsräume zu reduzieren und dadurch sicheres und stichprobeneffizientes Lernen auf realen Robotern zu ermöglichen. Die zweite Methode, Kernelized Guided Reinforcement Learning (KGRL), erweitert diesen Ansatz, indem sie menschliche Demonstrationen integriert, um Politiken zu initialisieren und Einschränkungen über Linearly Constrained Null-Space Kernelized Movement Primitives (LC-NS-KMP) durchzusetzen. KGRL verbessert die Extraktion von Aufgabenwissen im Vergleich zu RL-SCT, indem es Strategien zur Aufgabenerfüllung direkt aus Demonstrationen ableitet. Es führt zustandsabhängiges, unsicherheitsbewusstes Explorationsrauschen ein, das aus der Varianz der Demonstrationen abgeleitet wird, und erzwingt lineare Ungleichungsbeschränkungen für den Roboterzustand, um eine sichere, geführte Exploration zu gewährleisten.

Die dritte Methode, Smooth Kernelized Guided Reinforcement Learning (sKGRL), erweitert KGRL durch die Einbeziehung einer glatten Explorationsstrategie unter Verwendung glatter Kerne (z. B. radialer Basisfunktionen) und der Zustandsverlaufshistorie des Roboters, um abrupte, hochbeschleunigte Bewegungen zu minimieren und die Sicherheit weiter zu erhöhen. Im Gegensatz zu klassischem RL erzwingen unsere Methoden Beschränkungen explizit, wodurch einfachere Belohnungsfunktionen ohne zusätzliche Kostenbegriffe für Beschränkungsverletzungen möglich werden. Gemeinsam ermöglichen diese Beiträge ein effizientes und sicheres Reinforcement Learning direkt auf realen Robotern und fördern damit die Anwendbarkeit von RL auf komplexe, kontaktintensive Manipulationsaufgaben in der Robotik.

Evaluierungen auf realen Robotern und in simulierten Umgebungen zeigen, dass diese Frameworks bestehende klassische RL-Methoden in Sicherheits- und Effizienzmetriken übertreffen – etwa durch geringere Flüssigkeitsverschüttung, weniger Kollisionen, geringere Interaktionskräfte und vor allem eine reduzierte Anzahl von Trainingsepisoden. Durch die Kombination von Shared Control, demonstrationsgeleitetem Lernen und glatter Exploration bietet diese Arbeit einen ganzheitlichen Ansatz für das Reinforcement Learning in der realen Welt. Wir präsentieren eine umfassende Evaluation dieser Methoden, um ihre Fähigkeit zur Bewältigung vielfältiger Manipulationsaufgaben zu demonstrieren. Die Ergebnisse dieser Arbeit liefern neuartige Frameworks für skalierbares, sicheres Reinforcement Learning und treiben das Feld in Richtung praktischer Anwendungen in komplexen, kontaktintensiven robotischen Manipulationsaufgaben voran.

ACKNOWLEDGEMENTS

First of all, I would like to thank my parents, Tara and Bajarang. Through your actions, you have taught me never to give up, even in the face of overwhelming odds. Thank you for supporting my decisions and encouraging me to explore new paths.

I would like to express my heartfelt gratitude to my wife, Deepika. You entered my life at a critical phase, just as I was about to conclude one chapter and embark on an even more ambitious journey. Your support was essential in helping me complete this phase and prepare for the next.

I would also like to thank my aunts—Vidya, Shaila, and Lanka; my uncles—Kailas, Bhimraj, Kondiba, and Dinkar; and my parents-in-law, Anuradha and Madhukar, for their constant encouragement and blessings.

A special thanks to Abhinav, my brother and true partner in crime, for standing by me through the most challenging moments. I am equally grateful to my cousins Poonam, Shubham, Swapnil, Shraddha, Shashikant, Vrushali, and Shipra, my sister-in-law, for always standing firmly behind me and motivating me whenever I needed strength. Vikas, the warmth of your love still lives in my heart; we miss you.

I would like to thank the friends who encouraged me in the most unconventional ways—Digvijay, Mahesh, Mukesh, Hrishikesh, and Ajinkya. Each of you has taught me something radically different, helping me navigate equally varied situations.

Finally, I would like to thank all the people who helped me professionally. I am grateful to my maths teachers—Mrs. Dahale, Mr. Khandave, and Mrs. Chavhanke—for laying the early foundation of my academic journey.

Thanks to João Silvério, my research mentor, for finally teaching me how to put maths to work. Without your support, this work would not have reached completion. You were always there—quite literally next door—ready to help. Thank you for your patience and constant encouragement. I am grateful to Freek for giving me the opportunity to work in one of the most advanced robotics lab. I would also like to thank all my colleagues and teammates, who helped me navigate both professional and personal challenges.

Lastly, I would like to thank my PhD supervisor, Prof. Dr. Gerhard Neumann, for your guidance and for continually pushing me toward excellence. With your support, I was able to tackle scientific and professional challenges in a pragmatic, relaxed, and collegial environment.

In loving memory of my grandparents, Sita and Kisan . . .

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS AND SYMBOLS

LIST OF ABBREVIATIONS

RL	Reinforcement Learning
PPO	Proximal Policy Optimization
SAC	Soft Actor-Critic
TQC	Truncated Quantile Critics
NN	Neural Networks
SCT	Shared Control Templates
LfD	Learning from Demonstrations
GMM	Gaussian Mixture Models
GMR	Gaussian Mixture Regression
ProMPs	Probabilistic Movement Primitives
KMP	Kernelized Movement Primitives
KL divergence	Kullback-Leibler divergence
RL-SCT	Reinforcement Learning with Shared Control Templates
LC-NS-KMP	Linearly Constrained Null-Space Kernelized Movement Primitives
KGRL	Kernelized Guided Reinforcement Learning
sKGRL	Smooth Kernelized Guided Reinforcement Learning
GPR	Gaussian Process Regression

$\mathcal{S}$	State space in RL
$\boldsymbol{s} \in \mathcal{S}$	RL state
$\mathcal{A}$	Action space in RL
$\boldsymbol{a} \in \mathcal{A}$	RL action
$r \in \mathbb{R}$	Reward value
π	Policy
γ	Discount factor
D	Set of demonstrations
$\boldsymbol{\xi}^{\mathcal{I}}$	Input vector in demonstrations D
$\boldsymbol{\xi}^{\mathcal{O}}$	Output vector in demonstrations D
$\hat{\boldsymbol{\xi}}$	Null-space action
$\boldsymbol{\xi}_*^{\mathcal{I}}$	Current input
$\boldsymbol{\xi}_*^{\mathcal{O}}$	Current output
$\boldsymbol{p}$	Pose
t	Time
$\mathcal{N}$	Gaussian distribution
$\boldsymbol{\mu}$	Mean
$\boldsymbol{\Sigma}$	Variance
$\mathbb{E}$	Expectation

1

INTRODUCTION

1.1 MOTIVATION AND BACKGROUND

1.1.1 *Reinforcement Learning*

Reinforcement Learning (RL) has demonstrated remarkable potential in solving complex, high-dimensional, and highly non-linear problems, achieving super-human performance in Atari games [3] and mastering Go [4]. Extending these feats, vision-based RL agents have surpassed professional human drivers in high-fidelity racing simulators like Gran Turismo 7, optimizing split-second decisions in dynamic, multi-vehicle environments [5]. RL is often most effective when skills are learned from scratch, as evidenced by AlphaGo Zero outperforming its predecessor [6], which was pretrained with human demonstrations.

This paradigm has propelled advances in robotics as well, such as RL-trained humanoid robots achieving robust locomotion across diverse terrains via end-to-end policies that integrate perception and control [7]. In robotics, RL has garnered significant attention for enabling sophisticated motor skills, such as juggling [8], in-hand manipulation [9], ball-in-cup [10], pick-and-place [11], and quadrupedal locomotion over variable terrain [12]. These advances position RL as a transformative approach for autonomous robots in applications like industrial automation and domestic assistance.

1.1.2 *Learning from Demonstration*

Learning from Demonstration (LfD) [13] offers an alternative by enabling robots to imitate and adapt demonstrated motions, with frameworks such as Dynamic Movement Primitives (DMPs) [14], Probabilistic Movement Primitives (ProMPs) [15], and Kernelized Movement Primitives (KMPs) [16] which effectively address common real-world challenges such as generalizing to new situations and avoiding obstacles. However, these methods often struggle in dynamic environments where demonstrations inadequately represent task dynamics, particularly during in-contact tasks. In such tasks, the policy often receives out-of-distribution states as an input, resulting in failures. Impedance control and other compliance strategies help mitigate these issues, however, learning a robust LfD policy that can adapt to such uncertainties remains a significant challenge. Reinforcement learning addresses this challenge by training a reactive policy that considers the current state of both the robot and its environment.

1.1.3 *Challenges in Reinforcement Learning on Real Robots*

Despite its remarkable potential, deploying RL in real-world robotic systems is hindered by two critical challenges: ensuring safety during exploration and execution to prevent damage to robots or environments, and achieving sample-efficient learning to make RL scalable for practical applications. Traditional RL algorithms rely on extensive trial-and-error, requiring thousands of episodes to learn complex tasks, which is time-consuming and impractical for real-world deployment [17]. This is exacerbated by hardware wear from erratic exploration, potentially accelerating actuator degradation in prolonged training [18]. This inefficiency is particularly pronounced in tasks involving physical interactions, such as connector insertion, where unmodeled dynamics and kinematic uncertainties can lead to task failures or safety risks. Moreover, RL often relies on unstructured exploration strategies [19, 20], such as random action perturbations (e.g., ϵ-greedy or Gaussian noise). These methods add randomly sampled unstructured exploration noise to the policy actions, leading to erratic, abrupt robot behavior and lack of temporal consistency, increasing the risk of hardware damage.

Transfer learning from simulation to real robots aims to mitigate these issues by learning policies in simulation and then transferring them to real robots, but is limited by the sim-to-real gap, where discrepancies between simulated and real-world dynamics hinder performance [21]. Moreover, generating realistic simulation models is a tedious task, often limited by parameter identification and simulator capabilities [22]. Recent advancements, such as NVIDIA IsaacSim's GPU-accelerated contact modeling in insertion tasks [23, 24], have improved sim-to-real gap; yet, achieving robust contact dynamics requires significant modelling efforts and parameter identification. Learning directly on real robots eliminates the need for meticulous simulation modeling but exacerbates safety concerns due to the large number of trials required.

1.1.4 *Leveraging Task Knowledge and Constraints*

Manipulation tasks—such as pouring a drink or inserting a connector—differ from intricate motor skills in that they typically have well-defined goals and involve interactions with purpose-designed objects (e.g., bottles, glasses, connectors, sockets) [25–27]. Such interactions, as well as the task itself, impose specific constraints on the robot's state during execution. Human demonstrations provide an additional valuable source of task knowledge and constraints, implicitly encoding strategies for task completion as well as uncertainty and confidence measures [13]. Notably, demonstrations exhibit low variance in degrees of freedom (DoF) that are tightly constrained and higher variance

in those that allow more freedom for motion—effectively specifying task-relevant constraints.

Task constraints can be broadly categorized as *soft* or *hard* constraints. Soft constraints promote preferred behaviors—such as maintaining alignment between a connector and its socket—and, although not critical to task safety, contribute to more efficient and stable learning. Hard constraints specify conditions that must not be violated, for instance, avoiding collisions with the environment.

Relearning such constraints and intrinsic task knowledge from scratch through RL—over thousands of episodes—is inefficient and unnecessary. The implicit knowledge embedded in object interactions, task specifications and goals, and demonstrations about constraints and exploration strategies should be used to guide RL. Rather than relearning what is already known about the task, the robot should focus on *how* to perform it well—optimizing performance with minimal errors. This approach is particularly beneficial for contact-rich tasks—a largely unsolved class of manipulation problems—such as insertions, where the goal state is known but optimal behaviors (e.g., interaction forces) are difficult to hardcode or demonstrate precisely.

Although the task knowledge as priors have been used to guide robotic RL [28, 29], and explicit constraint-enforcement and safe-RL methods have been developed for robots [30–32], the simultaneous incorporation of task knowledge and explicit, enforced constraints during robot learning remains under-explored.

1.2 Problem Statement and Research Goals

With this overview and motivation in mind, this dissertation investigates the overarching question:

> *How can Reinforcement Learning be enabled directly on real robots by incorporating available task knowledge and constraints to ensure both safety and sample efficiency?*

To answer this question, we developed three main methods, each addressing a specific research question, as outlined below.

1. *How can we utilize hard-coded task knowledge and constraints to guide RL for safe and sample-efficient learning?*

   We develop *Reinforcement Learning with Shared Control Templates (RL-SCT)* in Chapter 3, a framework that integrates a shared control framework to encode task knowledge, enforce hard safety constraints, and reduce the effective state and action spaces, enabling safe and efficient learning.

2. *How can we leverage human demonstrations to guide RL while maintaining safety?*

   We present *Kernelized Guided Reinforcement Learning (KGRL)*, which extracts task completion strategy and uncertainty-aware, state-dependent exploration strategy from demonstrations, while supporting the enforcement of linear inequality constraints during learning for safety (see Chapter 4).

3. *How can we promote safe exploration in RL?*

   We propose *Smooth KGRL (sKGRL)*, which leverages smooth-kernel-based continuity priors and uncertainty-aware exploration to produce smooth, structured, and temporally consistent exploration (see Chapter 5).

Additionally, our methods simplify the reward design process by leveraging prior task knowledge and explicitly enforcing constraints. Explicit enforcement of constraints eliminates the need for implicit encoding of them via penalty terms in reward functions. Furthermore, partial task completion strategies enable sparse reward functions without specifying immediate dense rewards terms, further simplifying reward design.

In the following section, we provide a brief overview of the main contributions corresponding to the research questions above.

1.3 Overview of Contributions

1.3.1 *Reinforcement Learning with Shared Control Templates (RL-SCT)*

The first method, *Reinforcement Learning with Shared Control Templates* (RL-SCT), presented in [33], leverages Shared Control Templates (SCTs) [1] to integrate engineered task knowledge and safety constraints. Originally developed for shared control, SCTs are particularly well suited for guiding RL for several reasons. First, shared control enables users to command high-dimensional robotic systems through low-dimensional inputs, which, in the RL context, substantially reduces the dimensionality of the action space. Second, because user preferences vary, shared control promotes agency and adaptability by allowing freedom of movement within a constrained action space. This characteristic provides RL with opportunities for exploration while maintaining safety. Third, shared control typically enforces motion limits to ensure safety, a property that directly translates to safer exploration in RL. By mapping low-dimensional actions to high-dimensional robot control and enforcing constraints that restrict motion ranges, RL-SCT effectively reduces the action space, and ensures safe and efficient exploration. Additionally, RL-SCT facilitates shaped reward generation by exploiting prior information about task

progress and completion encoded within SCTs. In this framework, the reward function does not need to model constraint violations implicitly, as such constraints are enforced during exploration.

Experimental results provide empirical evidence that integrating SCTs with RL improves sample efficiency and ensures safer learning on real robotic systems, including tasks involving contact dynamics. RL-SCT reduces unnecessary errors during training and accelerates convergence in tasks where the object interaction model is fully known. For tasks with only partial interaction modeling, such as grid-clamp insertion, RL-SCT enables faster and safer learning by preventing collisions and minimizing interaction forces.

1.3.2 *Kernelized Guided Reinforcement Learning (KGRL)*

The second method, *Kernelized Guided Reinforcement Learning* (KGRL), presented in [34] uses human demonstrations to initialize a nominal policy and extract soft task constraints via a novel movement primitive representation, Linearly Constrained Null-Space Kernelized Movement Primitives (LC-NS-KMP). KGRL provides a *null-space* projector that allows actions generated by RL policies to modify the mean behavior of the LfD policy in accordance with the variance and correlations in the demonstrations. Consequently, the same null-space action results in larger deviations in states with high variance and smaller deviations in states with low variance. This uncertainty-awareness facilitates state-dependent exploration. Additionally, KGRL supports the imposition of linear inequality constraints on the robot state to ensure safety during exploration. KGRL improves task knowledge extraction over RL-SCT by deriving task completion strategies directly from demonstrations.

KGRL contributes a novel RL framework that leverages the abovementioned properties to enable reinforcement learning in complex real-world tasks. KGRL was demonstrated on a challenging BNC connector assembly task from the NIST assembly task board 1 [35], which requires precise insertion while maintaining compliance to prevent damage, followed by a sequence of translations and rotations to lock the connector. KGRL is particularly well-suited for such tasks because it 1) allows specification of hard constraints to ensure safe state-space exploration, and 2) enables uncertainty-aware, state-dependent exploration, which reduces unnecessary exploration in low-variance regions of the motion—thereby facilitating efficient and safe RL.

1.3.3 *Smooth Kernelized Guided Reinforcement Learning (sKGRL)*

Third method, Smooth Kernelized Guided Reinforcement Learning (sKGRL) [36] extends KGRL by incorporating smooth exploration strategy to minimize abrupt, high-acceleration actions.

Smooth exploration—characterized by controlled, gradual, and informed action selection—is a desirable property, as it promotes safe and efficient learning. By leveraging smooth kernels (e.g., radial basis function) and robot state history, sKGRL promotes trajectory continuity, enhancing safety for contact-rich tasks. sKGRL combines the strong continuity priors imposed by smooth kernels with KGRL's state-dependent exploration to ensure that future states remain close to previously visited ones, thereby promoting smooth trajectories. Specifically, sKGRL prepends the distribution over future states with the history of measured states, assigning low uncertainty to these observed states, which encourages continuity within each training episode.

1.4 Dissertation Structure

The remainder of this dissertation discusses all the contributions in detail, and it is organized as follows. Chapter 2 reviews the theoretical foundations and background methods that are used to develop the methodologies in this work. Chapter 3 presents RL-SCT, detailing the integration of Shared Control Templates with RL, along with experimental evaluations on real robotic systems. Chapter 4 introduces KGRL, derives a novel movement primitive framework—Linearly Constrained Null-Space Movement Primitives (LC-NS-KMP)—and describes the extraction of task knowledge from demonstrations, as well as its application to a complex connector insertion task on a real robot. Chapter 5 extends KGRL to sKGRL, focusing on smooth exploration strategies and their impact on learning efficiency and safety. Finally, Chapter 6 summarizes the key findings, discusses the implications of this work, and outlines potential directions for future research.

2

BACKGROUND

In this chapter, we summarize the theoretical background necessary to understand the methods developed in this work. First, we discuss mathematical methods used in the derivation of KGRL. We introduce Kullback-Leibler (KL) divergence followed by an overview of constrained quadratic optimization, the latter of which is used in KGRL for constraint handling. We then review key concepts from Learning from Demonstrations, focusing on probabilistic modelling of trajectories using Gaussian Mixture Models (GMMs), Probabilistic Movement Primitives (ProMPs), and Kernelized Movement Primitives (KMPs).

Following that, we will discuss the shared control methods which support the modelling of task knowledge and constraints, forming the basis of RL-SCT. Finally, we provide a brief overview of RL, covering key concepts and algorithms that underpin the RL methods employed in this work. We specifically focus on off-policy algorithms, as they are used in all three methods developed in this work.

2.1 MATHEMATICAL METHODS

2.1.1 *Kullback-Leibler Divergence*

The Kullback-Leibler (KL) divergence [37] is a fundamental measure in information theory that quantifies the difference between two probability distributions. Given the original distribution $\mathcal{P}_p$ and an approximate distribution $\mathcal{P}_q$, the KL divergence $D_{\mathrm{KL}}(\mathcal{P}_p \| \mathcal{P}_q)$ measures the information loss when $\mathcal{P}_q$ is used to represent $\mathcal{P}_p$. It is formally defined as

$$D_{KL}(\mathcal{P}_p \| \mathcal{P}_q) = \int \mathcal{P}_p(\mathbf{x}) \log \frac{\mathcal{P}_p(\mathbf{x})}{\mathcal{P}_q(\mathbf{x})} d\mathbf{x},$$

where $\mathbf{x}$ is a random variable. The KL divergence is always non-negative and equals zero if and only if $\mathcal{P}_p = \mathcal{P}_q$ almost everywhere. Owing to its asymmetric nature, it does not constitute a true metric but remains a widely used tool in statistics, machine learning, and RL for tasks such as model fitting, regularization, and policy optimization.

KL Divergence Between Two Multivariate Gaussian Distributions

In many applications in robotics and robot learning, policies or motion primitives are represented as Gaussian distributions. In such cases, it is often necessary to quantify the discrepancy between two Gaussian distributions—for instance, when aligning a learned policy

with a demonstrated behavior or regularizing policy updates. The KL divergence provides a principled measure of this discrepancy.

Given two multivariate Gaussian distributions

$$\mathcal{P}_p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p), \quad \mathcal{P}_q(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q),$$

the KL divergence from $\mathcal{P}_p$ to $\mathcal{P}_q$ is a closed-form expression

$$D_{\mathrm{KL}}(\mathcal{P}_p \| \mathcal{P}_q) = \frac{1}{2}\Bigg[\log \frac{|\boldsymbol{\Sigma}_q|}{|\boldsymbol{\Sigma}_p|} - \mathcal{D} + \mathrm{tr}\Big(\boldsymbol{\Sigma}_q^{-1}\boldsymbol{\Sigma}_p\Big) + (\boldsymbol{\mu}_q - \boldsymbol{\mu}_p)^\top \boldsymbol{\Sigma}_q^{-1}(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p)\Bigg], \tag{2.1}$$

where $\mathcal{D}$ is the dimensionality of the variable $\mathbf{x}$, $\mathrm{tr}(.)$ is the trace of a matrix, and $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ represent the mean and the variance of a Gaussian distribution, respectively.

This expression reveals two main sources of divergence: (i) the difference in means, captured by the term $\frac{1}{2}(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p)^\top \boldsymbol{\Sigma}_q^{-1}(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p)$, which measures how far the centers of the two distributions are apart; and (ii) the difference in covariances, represented by the remaining terms, which capture discrepancies in shape, scale, and orientation between the distributions.

Decomposition into Mean and Covariance Costs

The KL divergence can be viewed as the sum of two interpretable components: a mean alignment cost and a covariance alignment cost. The mean alignment cost can be defined as

$$\mathcal{L}_{\boldsymbol{\mu}} = \frac{1}{2}(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p)^\top \boldsymbol{\Sigma}_q^{-1}(\boldsymbol{\mu}_q - \boldsymbol{\mu}_p),$$

which penalizes the deviation between the means of the two distributions, weighted by the precision of $\mathcal{P}_q$. This term enforces similarity in expected behavior.

The covariance alignment cost is given by

$$\mathcal{L}_{\boldsymbol{\Sigma}} = \frac{1}{2}\left[\log \frac{|\boldsymbol{\Sigma}_q|}{|\boldsymbol{\Sigma}_p|} - \mathcal{D} + \mathrm{tr}\Big(\boldsymbol{\Sigma}_q^{-1}\boldsymbol{\Sigma}_p\Big)\right],$$

which penalizes differences in uncertainty and correlation structure between $\mathcal{P}_p$ and $\mathcal{P}_q$.

Thus, the KL divergence can be written as

$$D_{\mathrm{KL}}(\mathcal{P}_p \| \mathcal{P}_q) = \mathcal{L}_{\boldsymbol{\mu}} + \mathcal{L}_{\boldsymbol{\Sigma}}.$$

In practice, these two costs can be weighted separately to control the balance between matching the mean behavior and the uncertainty characteristics of the distributions as

$$\mathcal{L}_{\mathrm{KL}} = \alpha_{\boldsymbol{\mu}}\, \mathcal{L}_{\boldsymbol{\mu}} + \alpha_{\boldsymbol{\Sigma}}\, \mathcal{L}_{\boldsymbol{\Sigma}},$$

where α_μ and α_Σ are tunable weights. This decomposition is particularly useful in LfD when demonstrations encode both nominal behaviors (means) and variability (covariances), allowing separate control over the mean behavior of the modelled trajectories and variability during trajectory execution.

2.1.2 *Constrained Quadratic Optimization*

In robotics, control policies often need to satisfy constraints arising from safety, kinematics, or task-specific requirements. For instance, a robot may need to remain within joint limits, avoid collisions, or limit applied forces during manipulation tasks. These requirements can naturally be formulated as quadratic optimization problems with linear equality or inequality constraints, providing a computationally efficient and analytically interpretable framework for safe policy execution. We use this technique to derive LC-NS-KMP and KGRL in Chapter 4.

Quadratic Optimization with Linear Equality Constraints

Consider a quadratic program with linear equality constraints

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad f(\boldsymbol{x}) = \tfrac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x}, \\ &\text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned}$$

where $\boldsymbol{Q} \succ 0$, $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$.

Above constrained quadratic problem can be solved with corresponding *Lagrangian* as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha}) = \tfrac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\alpha}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}),$$

where $\boldsymbol{\alpha} \in \mathbb{R}^m$ are the Lagrange multipliers.

The necessary conditions for optimality, known as the *Karush–Kuhn–Tucker (KKT) conditions*, are

$$\nabla_{\boldsymbol{x}}\mathcal{L} = \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\alpha} = \boldsymbol{0}, \quad \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}.$$

The optimal solution $\boldsymbol{x}^*$ and $\boldsymbol{\alpha}^*$ can be obtained by solving above linear system as

$$\begin{bmatrix} \boldsymbol{Q} & \boldsymbol{A}^\top \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^* \\ \boldsymbol{\alpha}^* \end{bmatrix} = \begin{bmatrix} -\boldsymbol{c} \\ \boldsymbol{b} \end{bmatrix}.$$

If the unconstrained minimizer $\boldsymbol{x}_0 = -\boldsymbol{Q}^{-1}\boldsymbol{c}$ already satisfies $\boldsymbol{A}\boldsymbol{x}_0 = \boldsymbol{b}$, the constraints are *inactive*, and hence $\boldsymbol{\alpha}^* = 0$, and $\boldsymbol{x}^* = \boldsymbol{x}_0$.

Such linear equality constraints constrain the solution on a hyperplane.

Quadratic Optimization with Linear Inequality Constraints

Similar to the linear equality constraints, linear inequality constraints can be used to impose volumetric constraints.

Consider a quadratic program with linear inequality constraints,

$$\begin{aligned} &\min_{\boldsymbol{x}\in\mathbb{R}^n} \tfrac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x}, \\ &\quad \text{s.t. } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}. \end{aligned}$$

The corresponding Lagrangian is given by

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{x}^\top Q\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} - \boldsymbol{\alpha}^\top (A\boldsymbol{x} - \boldsymbol{b}),$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_m]^\top$ with Lagrange multipliers $\alpha_i \geq 0 \; \forall \; i \in \{1, 2, \ldots, m\}$.

The KKT conditions are generalized to include complementary slackness such that

$$\begin{aligned} &\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\alpha}^*) = \boldsymbol{Q}\boldsymbol{x}^* + \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\alpha}^* = 0, \\ &\boldsymbol{A}\boldsymbol{x}^* \leq \boldsymbol{b}, \\ &\alpha_i^* (\boldsymbol{A}_i \boldsymbol{x}^* - b_i) = 0, \\ &\alpha_i^* \geq 0. \end{aligned}$$

In this work, these quadratic programs are solved using the Proximal Interior-Point Quadratic Programming (PIQP) solver [38], which efficiently handles both equality and inequality constraints.

2.2 LEARNING FROM DEMONSTRATIONS AND MOVEMENT PRIMITIVES REPRESENTATION

Developing novel robotic skills traditionally requires extensive programming, tuning, and expertise in robotics and control, making the process time-consuming and inflexible. Learning from Demonstrations (LfD), also known as imitation learning, enables robots to acquire skills by observing expert behavior, eliminating the need for explicit programming [13]. Demonstrations naturally encode task completion strategies, constraints, and motor coordination, making LfD particularly suited to complex, high-dimensional tasks that are difficult to specify analytically [39].

A standard LfD pipeline comprises three stages: data acquisition, where demonstrations are collected; policy learning, where a model capturing the demonstrated behavior is inferred; and policy execution and adaptation, where the learned skill is deployed on the robot [39].

Learning from Demonstrations encompasses several methodological families that differ in how they represent, learn, and generalize demonstrated behaviors. These approaches can be broadly categorized into *Behavioral cloning*, *Inverse reinforcement learning*, and *Movement primitive representations*. We provide a brief overview of each category below.

2.2.1 *Approaches in Learning from Demonstrations*

Direct policy learning using Behavioral Cloning

Behavioral Cloning (BC) methods aim to infer a mapping from observed states to actions directly from demonstration data. Such methods are known as direct policy learning. BC frames LfD as a supervised learning problem: given pairs of observed states and expert actions, BC learns a policy that predicts the corresponding action for each encountered state [40]. The learning objective is typically formulated as minimizing a loss function that quantifies the discrepancy between the robot's predicted actions and the expert's actions. A typical approach to BC is to train a classifier or regressor to predict an expert's behavior given training data of the encountered observations (input) and actions (output) performed by the expert.

BC offers several advantages, including conceptual simplicity and computational efficiency. However, it suffers from *distribution shift*: small prediction errors can accumulate, causing the robot to encounter states that were not represented in the demonstrations, where the learned policy may fail [13, 41]. With limited number of demonstrations, BC often generates brittle policies that fail when the robot encounters situations outside the distribution of the demonstrations [42, 43].

Inverse Reinforcement Learning

Another important paradigm in LfD is Inverse Reinforcement Learning (IRL), which aims to infer the underlying reward function that explains expert behavior [43, 44]. Unlike direct policy learning, which maps states directly to actions, IRL assumes that experts act optimally with respect to some latent objective. By recovering this reward function, policies can be derived through reinforcement learning, allowing robots to generalize beyond the specific demonstrated trajectories. Extensions such as Maximum Entropy IRL [45] and Generative Adversarial Imitation Learning (GAIL) [43] further improve robustness and generalization by resolving ambiguity in reward inference and enabling learning from limited or suboptimal demonstrations.

IRL has been applied to tasks such as pick-and-place, tool use, and object sorting, guiding policies to reproduce demonstrated skills in new contexts [44]. In locomotion and motion synthesis, it allows humanoid robots or simulated agents to acquire walking, balancing, or stylistic motion behaviors that are difficult to encode manually [43].

The effectiveness of IRL depends on the coverage and quality of the demonstration set, as incomplete demonstrations can lead to biased or under-specified reward functions. Moreover, IRL methods often discard demonstrations after extracting the reward function and require additional interactions with the environment during the reinforcement learning phase, which can render them data-inefficient [43].

Movement Primitives

Movement Primitives (MPs) offer an explicit structure for representing continuous trajectories by providing compact, reusable parameterizations of motor behaviors that encode the essential characteristics of demonstrated motions while supporting generalization to new goals, timings, and environmental contexts [15, 16]. While direct policy learning and IRL address how to imitate or infer expert strategies, they often lack such trajectory-level representations, making MPs a compelling choice for tasks requiring smooth, adaptable motion generation.

Among the most important and widely used formulations are Dynamic Movement Primitives (DMPs) [14], which embed trajectories within stable nonlinear dynamical systems; Gaussian Mixture Models / Gaussian Mixture Regression (GMM/GMR) [39], often used within MP pipelines, which encode demonstrations as mixtures of Gaussian components and retrieve smooth, generalized trajectories; Probabilistic Movement Primitives (ProMPs) [15], which model motion as distributions over trajectories to capture variability; and Kernelized Movement Primitives (KMPs) [16], which enable trajectory adaptation through nonparametric regression.

MPs can be adapted to new targets, timings, and environmental constraints without retraining, making them highly flexible. They serve as reusable building blocks that can be composed, sequenced, or blended to create more complex skills. Formulations such as DMPs ensure stable convergence toward goals and robustness to perturbations. Probabilistic formulations, such as ProMPs, capture variability across demonstrations and allow for uncertainty-aware motion planning and execution. KMPs inherit these probabilistic advantages while eliminating the need to manually define basis functions and their associated parameters.

In this work, we use task-completion strategies and uncertainty-aware exploration strategies extracted from demonstrations to guide reinforcement learning. These demonstrations are encoded using the probabilistic representations introduced above, enabling retrieval of trajectories along with their variances. Hence, for a better understanding of the methods presented in this work, this section reviews the foundational components of probabilistic movement primitive representations, focusing on three main approaches: (i) GMM and GMR [39, 46], (ii) ProMPs [15], and (iii) KMPs [16]. A mathematical overview of these techniques is essential for understanding the derivation of the novel movement primitive framework, Linearly Constrained Null-Space Kernelized Movement Primitives (LC-NS-KMP), and the subsequent Kernelized Guided Reinforcement Learning (KGRL) framework presented in Chapter 4.

2.2.2 *Demonstrations*

We begin by formalizing the notion of demonstrations, which will be used consistently throughout this dissertation.

Consider a set D of M demonstrations $D = \{\{\boldsymbol{\xi}^{\mathcal{I}}_{n,m}, \boldsymbol{\xi}^{\mathcal{O}}_{n,m}\}_{n=1}^{N}\}_{m=1}^{M}$, where N is the length of a trajectory comprised of input $\boldsymbol{\xi}^{\mathcal{I}} \in \mathbb{R}^{\mathcal{I}}$ and corresponding output $\boldsymbol{\xi}^{\mathcal{O}} \in \mathbb{R}^{\mathcal{O}}$, where $\mathcal{I}$ and $\mathcal{O}$ denote the dimensionalities of the input and output vectors, respectively. Demonstrations can be obtained through various methods, such as teleoperation on real or virtual robots, by observing an expert control policy or by hand-guiding the robot—known as *kinesthetic teaching* [13]. In this work, we employ kinesthetic teaching as the primary source of demonstrations.

2.2.3 *Gaussian Mixture Models*

Gaussian Mixture Models (GMM) are probabilistic models that represent the data as a weighted sum of multiple Gaussian distributions. In robotics, GMMs are useful for modeling complex, multimodal data such as trajectories collected during demonstrations [39, 46]. Each trajectory point is treated as a sample, and GMM captures both the mean and variance within the dataset. GMMs represent complex

state-action or trajectory distributions, while Gaussian Mixture Regression (GMR) derives smooth, continuous mappings from these models, facilitating policy generation for robotic manipulation tasks. This section provides an overview of GMM and GMR, detailing their mathematical formulations, algorithmic mechanics, and applications in LfD for robotics.

A GMM models a probability density function over the dataset D as a mixture of C Gaussian components, with dimensionality $\mathcal{D} = \mathcal{I} + \mathcal{O}$ as

$$\mathcal{P}(\mathbf{x}) = \sum_{c=1}^{C} p_c \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c), \tag{2.2}$$

where

$$\mathbf{x} = \begin{bmatrix} \xi^{\mathcal{I}} \\ \xi^{\mathcal{O}} \end{bmatrix},$$

$p_c \in [0, 1]$ are the priors satisfying

$$\sum_{c=1}^{C} p_c = 1,$$

and, $\boldsymbol{\mu}_c \in \mathbb{R}^{\mathcal{D}}$ and $\boldsymbol{\Sigma}_c \in \mathbb{R}^{\mathcal{D}\times\mathcal{D}}$ are the mean and variance of the c^{th} Gaussian.

The c^{th} Gaussian component is defined as

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) = \frac{1}{\sqrt{(2\pi)^{\mathcal{D}} \, |\boldsymbol{\Sigma}_c|}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_c)^{\top} {\boldsymbol{\Sigma}_c}^{-1} (\mathbf{x} - \boldsymbol{\mu}_c) \right), \tag{2.3}$$

where $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ can be decomposed as

$$\boldsymbol{\mu}_c = \begin{bmatrix} \boldsymbol{\mu}_c^{\mathcal{I}} \\ \boldsymbol{\mu}_c^{\mathcal{O}} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\Sigma}_c = \begin{bmatrix} \boldsymbol{\Sigma}_c^{\mathcal{I}\mathcal{I}} & \boldsymbol{\Sigma}_c^{\mathcal{I}\mathcal{O}} \\ \boldsymbol{\Sigma}_c^{\mathcal{O}\mathcal{I}} & \boldsymbol{\Sigma}_c^{\mathcal{O}\mathcal{O}} \end{bmatrix}.$$

The parameters $\{p_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^{C}$ are estimated by maximizing the log-likelihood

$$\mathcal{L}(\{p_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^{C}) = \sum_{n=1}^{\hat{N}} \log \left(\sum_{c=1}^{C} p_c \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right),$$

where $\hat{N} = NM$ is the total number of the points in the dataset D. This is typically achieved using the Expectation-Maximization (EM) algorithm, which iterates between:

- *E-Step*: Compute responsibilities, the posterior probability of data point $\mathbf{x}_n$ belonging to component c as

$$p'_{nc} = \frac{p_c \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{j=1}^{C} p_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}.$$

- *M-Step*: Update parameters $\{p_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}_{c=1}^{C}$ as

$$p_c = \frac{p'_c}{\hat{N}},$$

$$\boldsymbol{\mu}_c = \frac{\sum_{n=1}^{\hat{N}} p'_{nc} \mathbf{x}_n}{p'_c},$$

$$\boldsymbol{\Sigma}_c = \frac{\sum_{n=1}^{\hat{N}} p'_{nc} (\mathbf{x}_n - \boldsymbol{\mu}_c)(\mathbf{x}_n - \boldsymbol{\mu}_c)^\top}{p'_c},$$

where,

$$p'_c = \sum_{n=1}^{\hat{N}} p'_{nc}.$$

2.2.4 *Gaussian Mixture Regression*

Gaussian Mixture Regression (GMR) [47] is an inference technique that uses the joint distribution learned by GMM to predict output variables $\boldsymbol{\xi}^{\mathcal{O}}$, given input variables $\boldsymbol{\xi}^{\mathcal{I}}$. GMR extends GMMs by deriving smooth, continuous mappings from input to output variables. GMR yields a full probability distribution for the outputs, enabling modeling of variability in predicted trajectories. This property is particularly valuable for robotics, where GMR can generate reference trajectory distributions, providing uncertainty metrics for further modification of the robot trajectories.

GMR leverages a GMM trained on a joint distribution $\mathcal{P}(\boldsymbol{\xi}^{\mathcal{I}}, \boldsymbol{\xi}^{\mathcal{O}})$ to compute the conditional distribution $\mathcal{P}(\boldsymbol{\xi}^{\mathcal{O}} \mid \boldsymbol{\xi}^{\mathcal{I}})$, which serves as a policy or regression function in LfD [46].

A probabilistic reference trajectory $T_r = \{\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1}^{N}$ can be generated using GMR from the learned GMM, where $\hat{\boldsymbol{\mu}}_n$ and $\hat{\boldsymbol{\Sigma}}_n$ are means and covariance matrices computed at each new input $\boldsymbol{\xi}_*^{\mathcal{I}}$. For a given input $\boldsymbol{\xi}_*^{\mathcal{I}}$, conditional expectation $\hat{\boldsymbol{\mu}}_n$ and conditional covariance $\hat{\boldsymbol{\Sigma}}_n$ are calculated as

$$\hat{\boldsymbol{\mu}}_n = \mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}} | \boldsymbol{\xi}_*^{\mathcal{I}}) = \sum_{c=1}^{C} w_c \hat{\boldsymbol{\mu}}_c, \tag{2.4}$$

$$\hat{\boldsymbol{\Sigma}}_n = \mathrm{cov}(\boldsymbol{\xi}_*^{\mathcal{O}} | \boldsymbol{\xi}_*^{\mathcal{I}}) = \mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}} \boldsymbol{\xi}_*^{\mathcal{O}^\top} | \boldsymbol{\xi}_*^{\mathcal{I}}) - \mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}} | \boldsymbol{\xi}_*^{\mathcal{I}}) \mathbb{E}^\top(\boldsymbol{\xi}_*^{\mathcal{O}} | \boldsymbol{\xi}_*^{\mathcal{I}}) \tag{2.5}$$

where,

$$w_c = \frac{p_c \mathcal{N}(\boldsymbol{\xi}_*^{\mathcal{I}} \mid \boldsymbol{\mu}_c^{\mathcal{I}}, \boldsymbol{\Sigma}_c^{\mathcal{I}\mathcal{I}})}{\sum_{j=1}^{C} p_j \mathcal{N}(\boldsymbol{\xi}_*^{\mathcal{I}} \mid \boldsymbol{\mu}_j^{\mathcal{I}}, \boldsymbol{\Sigma}_j^{\mathcal{I}\mathcal{I}})},$$

$$\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}} \boldsymbol{\xi}_*^{\mathcal{O}^\top} | \boldsymbol{\xi}_*^{\mathcal{I}}) = \sum_{c=1}^{C} w_c (\boldsymbol{\Sigma}'_c + \hat{\boldsymbol{\mu}}_c \hat{\boldsymbol{\mu}}_c^\top),$$

and

$$\hat{\boldsymbol{\mu}}_c = \boldsymbol{\mu}_c^{\mathcal{O}} + \boldsymbol{\Sigma}_c^{\mathcal{O}\mathcal{I}} (\boldsymbol{\Sigma}_c^{\mathcal{I}\mathcal{I}})^{-1} (\boldsymbol{\xi}_*^{\mathcal{I}} - \boldsymbol{\mu}_c^{\mathcal{I}}),$$
$$\boldsymbol{\Sigma}'_c = \boldsymbol{\Sigma}_c^{\mathcal{O}\mathcal{O}} - \boldsymbol{\Sigma}_c^{\mathcal{O}\mathcal{I}} (\boldsymbol{\Sigma}_c^{\mathcal{I}\mathcal{I}})^{-1} \boldsymbol{\Sigma}_c^{\mathcal{I}\mathcal{O}}.$$

2.2.5 *GMM and GMR for Learning from Demonstration*

GMM and GMR are core probabilistic techniques in LfD, enabling robots to model and generalize complex, multimodal demonstration data for tasks such as manipulation, trajectory planning, and locomotion [39, 46]. GMMs represent data as weighted combinations of Gaussian distributions, effectively capturing the variability inherent in human demonstrations—such as diverse joint configurations or insertion strategies—by modeling both mean and covariance information across high-dimensional trajectories. Building upon GMMs, GMR provides continuous, smooth mappings between task variables, allowing robots to generate generalized trajectories or control policies across novel contexts [48]. This probabilistic framework captures multiple demonstration strategies and quantifies uncertainty, improving robustness under noise and environmental variability.

However, GMM parameter estimation via the EM algorithm is sensitive to initialization and prone to local optima, while selecting the number of mixture components typically relies on heuristics like the Bayesian Information Criterion. GMR inherits these challenges and can face computational and accuracy limitations in high-dimensional or strongly nonlinear mappings. Despite these drawbacks, GMMs and GMR remain pivotal in LfD for their ability to represent multimodal behaviors and synthesize adaptable, probabilistically grounded motion policies.

2.2.6 *Probabilistic Movement Primitives*

Probabilistic Movement Primitives (ProMPs) [15, 49] are a powerful LfD framework in robotics for modeling and generating complex, multimodal trajectories. ProMPs leverage probabilistic representations to encode the variability inherent in human demonstrations, enabling robots to generalize and adapt movements. This section provides an overview of ProMPs, their mathematical formulation, and their applications in robotics, with a focus on their probabilistic nature.

ProMPs represent a trajectory as a weighted combination of basis functions, typically Gaussian kernels, augmented with a probabilistic structure to capture variability across demonstrations. We start by modelling the demonstrated trajectories using a parametric representation and subsequently introduce the probabilistic notion.

Consider a parametric trajectory representation

$$\boldsymbol{\xi}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}) = \boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})^{\top}\mathbf{w}, \tag{2.6}$$

with,

$$\boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}}) = \begin{bmatrix} \boldsymbol{\varphi}(\boldsymbol{\xi}^{\mathcal{I}}) & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\varphi}(\boldsymbol{\xi}^{\mathcal{I}}) & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \boldsymbol{\varphi}(\boldsymbol{\xi}^{\mathcal{I}}) \end{bmatrix},$$

where the matrix $\boldsymbol{\Theta} \in \mathbb{R}^{\mathcal{B}\mathcal{O}\times\mathcal{O}}$, weight vector $\mathbf{w} \in \mathbb{R}^{\mathcal{B}\mathcal{O}}$, with $\boldsymbol{\varphi}(\boldsymbol{\xi}^{\mathcal{I}})$ being a $\mathcal{B}$-dimensional basis function.

Consider weights $\mathbf{w}$ in Equation (2.6) are drawn from Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$, hence we can write

$$\boldsymbol{\xi}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}) \sim \mathcal{N}\left(\boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})^{\top}\boldsymbol{\mu}_w, \boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})^{\top}\boldsymbol{\Sigma}_w\boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})\right). \tag{2.7}$$

Such probabilistic way of modelling demonstrations enables generalization to novel situations or combining and blending movements in a principled manner. ProMPs allow temporal modulation of the movements and learning of stroke based and periodic motions.

The parameters $\boldsymbol{\mu}_w$ and $\boldsymbol{\Sigma}_w$ can be estimated from M trajectories in D with linear ridge regression as

$$\begin{aligned} \mathbf{w}_m &= (\boldsymbol{\Theta}^{\top}\boldsymbol{\Theta} + \chi\boldsymbol{I})^{-1}\boldsymbol{\Theta}^{\top}\bar{\boldsymbol{\xi}}_m^{\mathcal{O}}, \\ \boldsymbol{\mu}_w &= \frac{1}{M}\sum_{m=1}^{M}\mathbf{w}_m, \\ \boldsymbol{\Sigma}_w &= \frac{1}{M}\sum_{m=1}^{M}(\mathbf{w}_m - \boldsymbol{\mu}_w)(\mathbf{w}_m - \boldsymbol{\mu}_w)^{\top}, \end{aligned} \tag{2.8}$$

where, $\bar{\boldsymbol{\xi}}_m^{\mathcal{O}}$ is the output in m^{th} trajectory from the demonstrations D and χ is a small ridge factor.

ProMPs facilitate trajectory generation by sampling from the weight distribution as

$$\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w),$$

producing a new trajectory as

$$\boldsymbol{\xi}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}) = \boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})^{\top}\mathbf{w}.$$

This allows robots to generate diverse yet statistically consistent movements.

Moreover, ProMPs support conditioning on specific via-points or goals, enabling adaptation to new task requirements. ProMPs can be adapted to new via-point or goal $\boldsymbol{y}_t$ with covariance $\boldsymbol{\Sigma}_y$ at time t as

$$p(\mathbf{w}|\boldsymbol{y}_t) \sim \mathcal{N}(\boldsymbol{\mu}_w^{\text{cond}}, \boldsymbol{\Sigma}_w^{\text{cond}}),$$

where the conditional mean and covariance are derived using Gaussian conditioning,

$$\boldsymbol{\mu}_w^{\text{cond}} = \boldsymbol{\mu}_w + \boldsymbol{\Sigma}_w\boldsymbol{\Theta}\left(\boldsymbol{\Sigma}_y + \boldsymbol{\Theta}^{\top}\boldsymbol{\Sigma}_w\boldsymbol{\Theta}\right)^{-1}\left(\boldsymbol{y}_t - \boldsymbol{\Theta}^{\top}\boldsymbol{\mu}_w\right),$$

and,

$$\boldsymbol{\Sigma}_w^{\text{cond}} = \boldsymbol{\Sigma}_w - \boldsymbol{\Sigma}_w \boldsymbol{\Theta} \Big(\boldsymbol{\Sigma}_y + \boldsymbol{\Theta}^\top \boldsymbol{\Sigma}_w \boldsymbol{\Theta} \Big)^{-1} \boldsymbol{\Theta}^\top \boldsymbol{\Sigma}_w.$$

This conditioning enables ProMPs to adapt trajectories to new initial conditions, endpoints, or environmental constraints, such as navigating around obstacles in motion planning.

ProMPs can generate complex motions by combining or blending multiple primitives. Each primitive i is represented as a distribution over trajectories, $\mathcal{P}_i(\mathbf{y}) = \mathcal{N}(\boldsymbol{\mu}_{y,i}, \boldsymbol{\Sigma}_{y,i})$. When several primitives are to be activated simultaneously, their influence can be weighted by time- or context-dependent activation functions h_i. The blended trajectory distribution is obtained as a product of the individual ProMPs, normalized over all active primitives:

$$\mathcal{P}(\mathbf{y}) \propto \prod_i \mathcal{P}_i(\mathbf{y})^{h_i} = \mathcal{N}(\boldsymbol{\mu}_y^*, \boldsymbol{\Sigma}_y^*) \tag{2.9}$$

where the resulting mean and covariance are given by:

$$\boldsymbol{\Sigma}_y^* = \left(\sum_i h_i \boldsymbol{\Sigma}_{y,i}^{-1} \right)^{-1}, \tag{2.10}$$

$$\boldsymbol{\mu}_y^* = \boldsymbol{\Sigma}_y^{-1} \Big(\sum_i h_i \boldsymbol{\Sigma}_{y,i}^{-1} \Big) \boldsymbol{\mu}_{y,i} \tag{2.11}$$

This probabilistic blending mechanism enables smooth transitions between motion primitives, allowing robots to switch between or merge skills without discontinuities [15].

Probabilistic Movement Primitives provide a robust, flexible framework for modeling and generalizing multimodal trajectory distributions in robotics LfD. Their probabilistic formulation captures variability, enabling adaptable and robust trajectory generation for robotic manipulation tasks. However, challenges in parameter estimation continue to pose difficulties for ProMPs, their reliance on linear basis function models limits their expressiveness, and the fact that standard ProMPs operate as offline models restricts their adaptability in dynamic and non-stationary environments. Nevertheless, ProMPs remain a valuable and widely used framework for robot learning.

2.2.7 *Kernelized Movement Primitives (KMP)*

Kernelized Movement Primitives (KMP) [16] is another important approach for learning probabilistic trajectories from demonstrations. Approaches such as DMPs and ProMPs, represent trajectories through predefined basis functions with tunable weights, allowing robots to generalize learned motions to new goals or timings. However, these methods often rely on manually designed basis functions, which can

limit their flexibility in capturing nonlinear and high-dimensional motion patterns.

KMPs address these limitations by extending MPs into a kernel-based, nonparametric framework that flexibly models complex, nonlinear relationships between inputs and outputs without relying on manually designed basis functions. This formulation enables efficient handling of high-dimensional tasks, while the probabilistic structure of KMPs naturally represents motion uncertainty, supporting adaptable and robust behavior in dynamic environments. By combining the structure of movement primitives with the flexibility of kernel regression, KMPs offer several advantages: they generalize effectively across varying task conditions, require fewer demonstrations, smooth interpolation and extrapolation of motions, and can incorporate new constraints such as via-points or modified end conditions without retraining.

To formally describe the underlying principles of KMP, this section presents the mathematical formulation of the framework. The derivation begins by expressing demonstrated trajectories D as probabilistic distributions and reformulating the learning problem within a kernelized regression framework.

A probabilistic policy can be learned from demonstrations D using Gaussian Mixture Model (GMM) such that,

$$\mathcal{P}(\boldsymbol{\xi}^{\mathcal{I}}, \boldsymbol{\xi}^{\mathcal{O}}) \sim \sum_{c=1}^{C} p_c \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c), \tag{2.12}$$

where, p_c, $\boldsymbol{\mu}_c$,and $\boldsymbol{\Sigma}_c$ are the prior probability, mean and variance of the c^{th} Gaussian. We can employ Gaussian Mixture Regression (GMR) to obtain a reference trajectory $T_r = \{\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1}^{N}$ from the learned GMM, where $\hat{\boldsymbol{\mu}}_n$ and $\hat{\boldsymbol{\Sigma}}_n$ are means and covariance matrices computed at each new input. At the same time, a parametric trajectory can also be learned from the same demonstrations as,

$$\boldsymbol{\xi}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}) = \boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})^{\top} \mathbf{w}, \tag{2.13}$$

with

$$\boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}}) = \begin{bmatrix} \boldsymbol{\varphi}(\boldsymbol{\xi}^{\mathcal{I}}) & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\varphi}(\boldsymbol{\xi}^{\mathcal{I}}) & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \boldsymbol{\varphi}(\boldsymbol{\xi}^{\mathcal{I}}) \end{bmatrix},$$

where the matrix $\boldsymbol{\Theta} \in \mathbb{R}^{\mathcal{B}\mathcal{O} \times \mathcal{O}}$, weight vector $\mathbf{w} \in \mathbb{R}^{\mathcal{B}\mathcal{O}}$, with $\boldsymbol{\varphi}(\boldsymbol{\xi}^{\mathcal{I}})$ being a $\mathcal{B}$-dimensional basis function. Consider weights $\mathbf{w}$ are drawn from $\mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w)$, hence we can write

$$\boldsymbol{\xi}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}) \sim \mathcal{N}(\boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})^{\top} \boldsymbol{\mu}_w, \boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})^{\top} \boldsymbol{\Sigma}_w \boldsymbol{\Theta}(\boldsymbol{\xi}^{\mathcal{I}})). \tag{2.14}$$

In order to derive a non-parametric representation, the original KMP formulation [16] proposes to minimize the Kullback-Leibler (KL) divergence between the above-mentioned two Gaussian distributions, represented by T_r and Equation (2.14), leading to a *mean minimization subproblem* with cost function

$$\begin{aligned}\underset{\boldsymbol{\mu}_w}{\textbf{argmin}} \sum_{n=1}^{N} \frac{1}{2}(\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n)^\top \hat{\boldsymbol{\Sigma}}_n^{-1}(\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n) \\ + \frac{1}{2}\lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w,\end{aligned} \tag{2.15}$$

and *covariance minimization subproblem* with cost function

$$\begin{aligned}\underset{\boldsymbol{\Sigma}_w}{\textbf{argmin}} \sum_{n=1}^{N} \frac{1}{2}\Bigg[- \log\left|\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\,\boldsymbol{\Sigma}_w\,\boldsymbol{\Theta}(\boldsymbol{\xi}_n^{\mathcal{I}})\right| \\ + \operatorname{tr}\Big(\hat{\boldsymbol{\Sigma}}_n^{-1}\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\,\boldsymbol{\Sigma}_w\,\boldsymbol{\Theta}(\boldsymbol{\xi}_n^{\mathcal{I}})\Big)\Bigg] + \frac{1}{2}\lambda_c \operatorname{tr}(\boldsymbol{\Sigma}_w),\end{aligned} \tag{2.16}$$

where $\operatorname{tr}(.)$ is the trace of a matrix, and λ and λ_c are regularization terms.

The solution of Equations (2.15) and (2.16) leads to the formulation of KMPs. For an input $\boldsymbol{\xi}_*^{\mathcal{I}}$, the mean prediction of a KMP is given by

$$\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}}) = \boldsymbol{k}^*(\boldsymbol{K} + \lambda\boldsymbol{\Sigma})^{-1}\boldsymbol{\mu}, \tag{2.17}$$

and covariance prediction of a KMP is given by

$$\operatorname{cov}(\boldsymbol{\xi}_*^{\mathcal{O}}) = \frac{N}{\lambda_c}\Big(\boldsymbol{k}(\boldsymbol{\xi}_*^{\mathcal{I}}, \boldsymbol{\xi}_*^{\mathcal{I}}) - \boldsymbol{k}^*(\boldsymbol{K} + \lambda_c\boldsymbol{\Sigma})^{-1}\boldsymbol{k}^{*\top}\Big), \tag{2.18}$$

where

$$\begin{aligned}&\boldsymbol{\xi}_*^{\mathcal{O}} = f(\boldsymbol{\xi}_*^{\mathcal{I}}),\\ &\boldsymbol{\mu} = [\hat{\boldsymbol{\mu}}_1^\top, \hat{\boldsymbol{\mu}}_2^\top, \ldots, \hat{\boldsymbol{\mu}}_N^\top]^\top,\\ &\boldsymbol{\Sigma} = \operatorname{blockdiag}(\hat{\boldsymbol{\Sigma}}_1, \hat{\boldsymbol{\Sigma}}_2, \ldots, \hat{\boldsymbol{\Sigma}}_N),\\ &\boldsymbol{k}^* = [\boldsymbol{k}(\boldsymbol{\xi}_*^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}),\ \ldots,\ \boldsymbol{k}(\boldsymbol{\xi}_*^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}})],\\ &\boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & \boldsymbol{k}(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \ldots & \boldsymbol{k}(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \\ \boldsymbol{k}(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & \boldsymbol{k}(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \ldots & \boldsymbol{k}(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{k}(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & \boldsymbol{k}(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \ldots & \boldsymbol{k}(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \end{bmatrix},\end{aligned}$$

matrix $\boldsymbol{k}(\boldsymbol{\xi}_i^{\mathcal{I}}, \boldsymbol{\xi}_j^{\mathcal{I}}) = k(\boldsymbol{\xi}_i^{\mathcal{I}}, \boldsymbol{\xi}_j^{\mathcal{I}})\boldsymbol{I}$ is obtained by applying the kernel treatment [34] on basis functions as $\varphi(\boldsymbol{\xi}_i^{\mathcal{I}})^\top\varphi(\boldsymbol{\xi}_j^{\mathcal{I}}) = k(\boldsymbol{\xi}_i^{\mathcal{I}}, \boldsymbol{\xi}_j^{\mathcal{I}})$, with kernel function k. In this dissertation, we only focus on the mean minimization subproblem as our goal is to extract a policy for the robot to track.

While KMPs have some considerations, such as sensitivity to kernel choice and implicit parameter representation, these also contribute to their flexibility and expressive power. The kernel-based formulation allows KMP to capture complex, nonlinear motions and adapt to varying task conditions in ways that traditional parametric approaches cannot. The use of kernel matrices can pose challenges in interpretability and computational cost for high-dimensional trajectories with large number of points, but variants such as null-space KMPs [50] offer effective strategies to mitigate computational cost.

KMP Formulation for Developing a Guided RL Framework: KGRL

One of the primary aims of this work is to develop a method that guides reinforcement learning using task completion strategies and uncertainty-aware exploration strategies extracted from demonstrations, while making RL safer by enforcing constraints during exploration and execution.

KMPs provide a natural foundation for developing such a guided RL framework due to three key properties. First, KMPs offer a probabilistic representation of motion, enabling the extraction of task knowledge in the form of trajectories and associated uncertainties, which is essential for guiding exploration. Second, modulating KMPs in uncertainty-aware manner [51] and enforcing constraints [52] on the generated trajectories has been demonstrated previously. Third, KMPs are nonparametric, eliminating the need for predefined basis functions and reducing the number of hyperparameters that must be tuned, which simplifies their integration with RL algorithms.

Given these characteristics, KMPs constitute a natural and effective choice for building the guided RL framework, KGRL, presented in Chapter 4.

2.2.8 *Aleatoric and Epistemic Uncertainties in LfD*

Two kinds of uncertainties arise while learning movement primitives from demonstrations, namely *aleatoric uncertainties* and *epistemic uncertainties* [53]. Aleatoric uncertainties represent the variability inherent in the demonstrations, including the different possible ways to perform a task as well as measurement or sensor noise in the data. This type of uncertainty is captured by probabilistic movement representations such as ProMPs or when fitting a Gaussian or a GMM to the demonstrations. In contrast, epistemic uncertainties arise from a lack of sufficient data or incomplete knowledge about the task, reflecting areas of the input space where the model has limited experience. Unlike aleatoric uncertainty, epistemic uncertainty can be reduced by providing additional demonstrations or richer training data. Figure 2.1, reproduced from [51], illustrates the distinction between aleatoric and epistemic uncertainties as captured by GMR and Gaussian Process

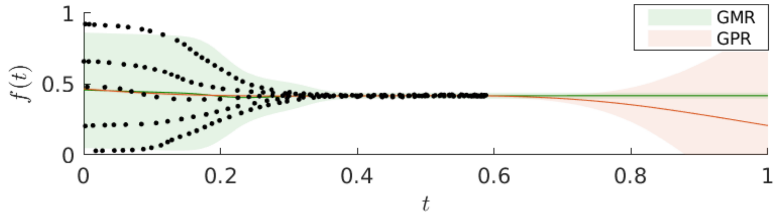


Figure 2.1: Aleatoric and epistemic uncertainties are captured by Gaussian Mixture Regression (GMR) and Gaussian Process Regression (GPR), respectively, with green and red shaded areas representing demonstration variability and uncertainty due to the absence of training data (black dots). Image reproduced from [51].

Regression (GPR) [54], respectively. Silvério et al. [51] extend KMP to model both aleatoric and epistemic uncertainties, enabling robots to behave safely and be compliant even in regions with limited or no training data.

Variability in the demonstrations reflects the confidence and diverse strategies of the teacher in the demonstrated motion at different phases of the task. In this dissertation, we use aleatoric uncertainty, which captures such variability, to deliver an uncertainty-aware, state-dependent exploration strategy for RL in Chapter 4.

To guide reinforcement learning using hand-crafted task knowledge, we require a framework that enables systematic modelling of such knowledge and constraints, and facilitates simplified control of the robot taking advantage of modelled knowledge. The shared autonomy framework called Shared Control Templates (SCTs) is such a framework, which we leverage in this work to facilitate the guided RL framework, RL-SCT, presented in Chapter 3.

In shared control, the robot's state is jointly controlled by a human operator and assistive algorithms running in the background, enabling a collaborative framework where both contribute to task execution [55]. Shared control primarily aims to reduce the operator's cognitive load by delegating complex, repetitive, or deterministic tasks to an assistive system, which enhances overall performance, efficiency, and user comfort [56]. This approach leverages the strengths of human decision-making and robot's capabilities, allowing the assistive algorithms to handle low-level control tasks, such as trajectory control, constraint enforcement, or collision avoidance, while the human provides high-level guidance or situational judgment [57]. Ultimately, shared control fosters a symbiotic relationship between human and machine, optimizing performance while maintaining user agency.

In most applications of shared control, the aim is to map low-dimensional input commands to task-relevant motion on a high-dimensional robotic system. An example is EDAN (EMG-controlled Daily AssistaNt) [58], which consists of an electric wheelchair (with 3 degrees of freedom (DoFs)) with an 8 DoF articulated arm. EDAN typically takes a 3D input signal extracted from surface electromyography or given from a joystick. It maps these inputs to 6D end-effector movements, which are then mapped to the movement of the overall 11 DoF system (excluding the degrees of freedom in the hand) through whole-body motion control [1].

Performing tasks such as opening doors or pouring liquids with such systems presents several challenges. First, the 6D end-effector action space is too complex for a user to control directly, as typical user commands are limited to at most three dimensions. Second, the system must respect numerous constraints—for instance, avoiding excessive tilting of a full bottle before the pouring phase and minimizing unnecessary translations during pouring to prevent spillage outside the target container. Third, these tasks are inherently multi-phase, involving sequential steps such as grasping the bottle, moving it toward the mug, tilting, and finally pouring. Lastly, both the state and action spaces in these tasks are continuous, adding further complexity to control.

To address these challenges, Quere et al. [1] proposed Shared Control Templates (SCTs), which consist of several components, namely

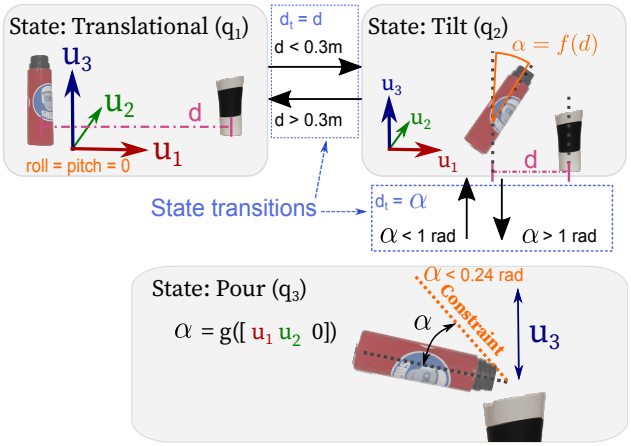


Figure 2.2: The Shared Control Template (SCT) for pouring liquid. Different phases of a task are modeled as different SCT states (q_i) in a Finite State Machine (*Translational*, *Tilt*, *Pour*). In each state, an Input Mapping maps the 3D user input commands u_1, u_2, u_3 to 6D end-effector motions. Active Constraints (shown in orange text) limit the range of motion.

1) Input mappings, 2) Active constraints, 3) Finite state machine, and 4) Transition functions. SCTs divide the task in several SCT states and each state of SCT has above-mentioned components. We will discuss the SCTs with an example SCT for pouring liquid, as shown in Figure 2.2.

INPUT MAPPINGS (IMS) Typically, the user provides input commands through electromyography or a joystick [1]. An Input Mapping converts these 3D user inputs to phase-dependent 6D end-effector motions.

For example, in the *Translational* and *Tilt* states in Figure 2.2, all 3 inputs u_1, u_2, u_3 are mapped to the 3 translational Cartesian DoFs. In the *Pour* state, u_3 is mapped to translation in Z-direction and the vector $[u_1\ u_2\ 0]$ is mapped to the tilt angle of the thermos via a scalar projection.

Formally, an IM is a function which computes a desired displacement $\Delta\boldsymbol{H} \in \mathrm{SE}(3)$ in task space from an n-dimensional input $\boldsymbol{u}_t$ with $n \leq 6$ at time step t (dropping the subscript i in q_i from now on):

$$\begin{aligned} \mathrm{map}_q &: \mathbb{R}^n \to SE(3) \\ \boldsymbol{u}_t &\mapsto \Delta\boldsymbol{H}. \end{aligned} \tag{2.19}$$

The displacement computed from Equation (2.19) is then applied on the end-effector pose $\boldsymbol{H}_t$:

$$\begin{aligned} \mathrm{displace}_q &: SE(3), SE(3) \to SE(3) \\ \boldsymbol{H}_t, \Delta\boldsymbol{H} &\mapsto \boldsymbol{H}_{t+1}^{\mathrm{im}}. \end{aligned} \tag{2.20}$$

ACTIVE CONSTRAINTS (ACS) Active Constraints [59] constrain the end-effector pose that results from applying an IM. As illustrated in Figure 2.2 during the phase *Pour*, the tilt angle α of the bottle is constrained, to avoid excessive pouring. Another example of AC in the *Tilt* phase is the value of the tilting angle being a function of the distance from the target.

Formally, after applying the IM to obtain $\boldsymbol{H}_{t+1}^{\text{im}}$, geometric constraints can be enforced using AC. An AC [1] applies a projection of the form,

$$\begin{aligned} \text{project}_q \colon SE(3) &\to SE(3) \\ \boldsymbol{H}_{t+1}^{\text{im}} &\mapsto \boldsymbol{H}_{t+1}^{\text{ac}}, \end{aligned} \tag{2.21}$$

where $\boldsymbol{H}_{t+1}^{\text{ac}}$ is the constrained end-effector pose. This constraint could for example be the arc-circle path traced by the door handle when opening a door or orientation constraints depending on the end-effector position when approaching an object, as in the *Tilt* phase in Figure 2.2.

FINITE STATE MACHINE (FSM) AND TRANSITION FUNCTIONS Different phases of a task require different IMs and ACs, as illustrated in Figure 2.2. For this reason, these phases are represented as states in a finite-state machine (FSM). The FSM triggers transitions from one state to the next by monitoring transition functions, which measure when certain distances $d_t \in \mathbb{R}$ exceeds pre-specified thresholds.

During the approach in the first state in the pouring task for instance, d_t is the distance between thermos tip $\boldsymbol{x}_{\text{th},t}$ and mug tip $\boldsymbol{x}_{\text{mug},t}$ positions at t, i.e. $d_t = ||\boldsymbol{x}_{\text{th},t} - \boldsymbol{x}_{\text{mug},t}||$. In the last state, when actually pouring, the distance is the tilting angle $d_t = \alpha_t$. Both are highlighted as blue rectangles in Figure 2.2.

A key aspect of SCTs is that they facilitate task completion through shared control by defining task-relevant IMs and ACs, but the user always remains in control, i.e. determines the speed of movement, the amount of liquid that is poured, etc.

SCTs generalize the IMs, ACs and Transition distances for a set of similar objects involved in similar tasks. Object specific parameterization, e.g. coordinate frames involved in the SCT specification, size of the object, can be retrieved from a world model.

SCTs has empirically shown that they reduce the workload of a human operator, even with a low-throughput input devices. However, they need careful design of intuitive input mappings, and transition functions often requiring several design iterations.

2.3.1 *SCTs as Movement Primitives*

From the perspective of motion representation, SCTs can be viewed as a form of movement primitives. Each SCT state encodes a phase-

specific motion through its Input Mapping and Active Constraints, defining a structured, reusable behavior in task space. The finite-state machine sequences these states, effectively chaining multiple primitives to execute a complex, multi-phase task. This interpretation highlights how SCTs capture task-relevant strategies and constraints, making them compatible with frameworks that leverage movement primitives for learning, adaptation, or guided RL.

2.4 REINFORCEMENT LEARNING

In this work, Reinforcement Learning (RL) is used in the context of policy learning for robotic manipulation tasks. This section provides a brief overview of RL, focusing on the mathematical foundations and key concepts relevant to the algorithms employed in the experimental evaluations.

2.4.1 *Preliminaries*

Reinforcement Learning (RL) is a machine learning paradigm that enables agents to learn optimal behaviors through interaction with its environment. In RL framework, an agent observes the current state, selects and executes an action in the environment, and receives a reward based on the outcome, iteratively improving its policy to maximize cumulative rewards over time.

Formally, RL is modeled as a Markov Decision Process (MDP), defined by states, actions, transition probabilities, rewards, and a discount factor. MDP is formalized as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, $p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the transition probability, $r(\boldsymbol{s}, \boldsymbol{a}) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, and $\gamma \in [0, 1)$ the discount factor [60]. At time t, an agent in state $\boldsymbol{s}_t \in \mathcal{S}$ selects action $\boldsymbol{a}_t \in \mathcal{A}$ according to policy $\pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$, transitions to state $\boldsymbol{s}_{t+1} \sim p(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$, and receives reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. The objective is to find an optimal policy π^* that maximizes the expected cumulative discounted reward given by

$$J(\pi) = \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{a}_t) \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right].$$

State space:

State space $\mathcal{S}$ represents all possible configurations of the environment that the agent can encounter. In robotics, this may include positions, velocities, sensor readings, and other relevant features that are necessary to define the state of the robot and the environment. State space in robotics policy search is usually continuous and high-dimensional, reflecting the complexity of real-world environments.

Action space:

Action space $\mathcal{A}$ encompasses all possible actions the agent can take. In robotic applications, actions often correspond to motor commands, joint angles, or end-effector positions. Similar to state space, action space is typically continuous and high-dimensional in robotics, necessitating sophisticated algorithms to effectively explore and optimize policies.

Reward

The reward function $r(\boldsymbol{s}, \boldsymbol{a})$ provides feedback to the agent about the desirability of its actions in given states. It quantifies the immediate benefit of taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$. In robotics, designing an appropriate reward function is crucial, as it directly influences the learning process and the quality of the resulting policy. Rewards can be sparse or dense, depending on the task requirements. The reward function may also include terms that evaluate the quality of the agent's behavior, such as cost terms for action usage, force interactions, and motion jerks.

Transition dynamics

The outcome of an action is modeled by the transition probability $p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$, which defines the likelihood of moving to state $\boldsymbol{s}'$ after taking action $\boldsymbol{a}$ in state $\boldsymbol{s}$. In robotics, transition dynamics can be complex and stochastic due to factors like sensor noise, actuator uncertainty, and environmental variability, and it is often unknown to the agent.

Discount factor

The discount factor γ determines the importance of future rewards relative to immediate rewards. A value of γ close to 1 emphasizes long-term rewards, while a value closer to 0 prioritizes immediate rewards. In robotics, the choice of γ can significantly affect the learned policy, especially in tasks requiring long-term planning and foresight.

Markov property

Markov property assumes that the next state depends only on the current state and action, not on the sequence of past states and actions. This property simplifies the modeling of decision-making processes and is fundamental to the formulation of MDPs. It can be mathematically expressed as

$$p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \ldots, \boldsymbol{s}_0, \boldsymbol{a}_0) = p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t).$$

The initial state distribution $p(\boldsymbol{s}_0)$ specifies the starting conditions for the agent.

2.4.2 *State and Action Value Functions*

The value function $V^{\pi}(\boldsymbol{s})$ estimates how much future reward an agent can expect when starting from a particular state and following a given policy π. It provides a way to evaluate how good different states are in terms of long-term outcomes. By capturing the expected return over

time, the value function helps guide the learning and improvement of the policy. The value function in RL is given by

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r(\boldsymbol{s}_t, \boldsymbol{a}_t) \mid \boldsymbol{s}_0 = \boldsymbol{s}\right].$$

The value function can be estimated recursively, also known as Bellman equation:

$$V^{\pi}(\boldsymbol{s}_t) = \mathbb{E}_{\pi}\left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma V^{\pi}(\boldsymbol{s}_{t+1}) \mid \boldsymbol{s}_t\right].$$

The action-value function or Q-function $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ extends this idea by evaluating the expected future reward for taking a specific action $\boldsymbol{a}$ in a given state $\boldsymbol{s}$ and then following the policy π. It allows the agent to compare possible actions and choose those that lead to higher long-term rewards. The action-value function is given by

$$Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r(\boldsymbol{s}_t, \boldsymbol{a}_t) \mid \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a}\right].$$

The Bellman equation for the action-value function is given by

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}_{\pi}\left[r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma Q^{\pi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \mid \boldsymbol{s}_t, \boldsymbol{a}_t\right].$$

2.4.3 *Comparison of On-policy and Off-Policy RL Algorithms*

RL algorithms can be broadly categorized into on-policy and off-policy methods, distinguished by how they collect and utilize interaction data. Off-policy algorithms learn an optimal target policy using data generated by a separate behavior policy, decoupling exploration from policy optimization. This enables the reuse of heterogeneous data sources—such as past trajectories, demonstrations, or interactions from other agents—without requiring execution of the target policy during data collection. Prominent examples include Q-learning [61], Deep Q-Networks (DQN) [3], and off-policy variants of Soft Actor-Critic (SAC) [62]. In robotics, the ability to exploit previously gathered experience is particularly advantageous, as it reduces reliance on costly and time-intensive real-world interactions.

In contrast, on-policy methods—such as SARSA [63], Proximal Policy Optimization (PPO) [64], and Trust Region Policy Optimization (TRPO) [65]—learn directly from data generated by the current policy, which must be executed to collect new samples. This tight coupling between data collection and policy updates limits the reuse of past data and demands careful balancing of exploration and exploitation, contributing to lower sample efficiency compared to off-policy approaches. These differences in data usage, exploration strategies, and efficiency have significant implications for robotics, where data collection on real systems is expensive and often mechanically taxing.

Off-policy algorithms are well-suited for robotic domains due to their ability to reuse experience via replay buffers and leverage external data, accelerating training while minimizing wear on hardware. Methods such as SAC effectively handle high-dimensional, continuous state and action spaces. However, off-policy learning also introduces challenges, including training instability arising from distributional shifts between behavior and target policies [60], increased algorithmic complexity, and risks of value overestimation [66]. Furthermore, successful learning depends on adequate coverage of the state-action space by the policy, without which learning may be incomplete or biased.

Among off-policy methods, Soft Actor-Critic (SAC) [62] has become a leading approach for continuous control due to its entropy-regularized objective, which promotes robust exploration and stable learning. Its extension, Truncated Quantile Critics (TQC) [67], mitigates overestimation bias, improving policy reliability in precision-critical tasks. In this work, SAC and TQC are employed to learn continuous control policies in simulation and on real robotic systems. The remainder of this section provides an overview of off-policy algorithms, detailing their mathematical foundations and relevance to robotic control.

2.4.4 *Deep Q-Networks (DQN)*

Deep Q-Networks (DQN) [3] are a foundational off-policy RL algorithm for discrete action spaces. DQN approximates the action-value function $Q(\boldsymbol{s}, \boldsymbol{a})$ using a neural network with parameters $\boldsymbol{\phi}$ and minimizes the loss

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')\sim D_{\mathcal{R}}}\Big[\big(Q_{\boldsymbol{\phi}}(\boldsymbol{s},\boldsymbol{a}) - y\big)^2\Big],$$

where the target y is computed as

$$y = r + \gamma \max_{\boldsymbol{a}'} Q_{\bar{\boldsymbol{\phi}}}(\boldsymbol{s}', \boldsymbol{a}').$$

Experience replay $D_{\mathcal{R}}$ and target network $\bar{\boldsymbol{\phi}}$ stabilize learning. DQN laid the groundwork for off-policy learning, but its restriction to discrete actions limits direct applicability to robotics. Continuous control algorithms extend these ideas to high-dimensional, continuous action spaces.

2.4.5 *Deep Deterministic Policy Gradient (DDPG)*

DDPG [68] extends DQN to handle continuous actions. DDPG defines an *Actor*, a deterministic policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{s})$ that outputs actions, and a *Critic* $Q_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{a})$ estimates expected return.

The critic is updated as,

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}')\sim D_{\mathcal{R}}}\left[\left(Q_{\boldsymbol{\phi}}(\boldsymbol{s},\boldsymbol{a}) - y\right)^2\right],$$

$$y = r + \gamma Q_{\bar{\boldsymbol{\phi}}}(\boldsymbol{s}', \pi_{\bar{\boldsymbol{\theta}}}(\boldsymbol{s}')).$$

The actor is updated as,

$$\nabla_{\boldsymbol{\theta}} J \approx \mathbb{E}_{\boldsymbol{s}\sim D_{\mathcal{R}}}\left[\nabla_{\boldsymbol{a}} Q_{\boldsymbol{\phi}}(\boldsymbol{s},\boldsymbol{a})\big|_{\boldsymbol{a}=\pi_{\boldsymbol{\theta}}(\boldsymbol{s})} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(\boldsymbol{s})\right].$$

Exploration is achieved by adding noise $\mathcal{N}_t$ to the actor output as $\boldsymbol{a}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_t) + \mathcal{N}_t$.

DDPG demonstrates how off-policy methods can learn continuous control policies, making it applicable to robotic tasks. However, exploration noise parameters and target network update rate greatly affect performance in DDPG.

2.4.6 *Soft Actor-Critic (SAC)*

The Soft Actor-Critic (SAC), introduced by Haarnoja et al. [62], is an off-policy deep RL algorithm that integrates a maximum entropy framework to optimize policies for both expected rewards and exploration. This dual objective makes SAC particularly effective for continuous control tasks in robotics, where diverse exploration in high-dimensional action spaces is critical.

Mechanics of SAC

Unlike traditional RL, which maximizes the expected cumulative reward $\mathbb{E}_{(\boldsymbol{s}_t,\boldsymbol{a}_t)\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(\boldsymbol{s}_t,\boldsymbol{a}_t)\right]$, SAC incorporates policy entropy to encourage exploration, defining the objective as

$$J(\pi) = \mathbb{E}_{(\boldsymbol{s}_t,\boldsymbol{a}_t)\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(\boldsymbol{s}_t,\boldsymbol{a}_t) + \alpha\mathcal{H}(\pi(\cdot \mid \boldsymbol{s}_t))\right)\right],$$

where

$$\mathcal{H}(\pi(\cdot \mid \boldsymbol{s}_t)) = -\mathbb{E}_{a\sim\pi(\cdot|\boldsymbol{s}_t)}[\log \pi(\boldsymbol{a} \mid \boldsymbol{s}_t)]$$

is the Shannon entropy of the policy and $\alpha > 0$ is a temperature parameter balancing reward maximization and exploration.

SAC employs three primary components:

1. *Actor*: A stochastic policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s})$, typically parameterized as a Gaussian distribution by a neural network with parameters $\boldsymbol{\theta}$, outputs actions $\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s})$. For continuous action spaces, the policy is often represented as $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{s}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{s}))$, where $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{s})$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{s})$ are the mean and covariance, respectively.

2. *Critic*: Two action-value functions $Q_{\boldsymbol{\phi}_1}(\boldsymbol{s}, \boldsymbol{a})$ and $Q_{\boldsymbol{\phi}_2}(\boldsymbol{s}, \boldsymbol{a})$, parameterized by neural networks with parameters $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$, estimate the expected return. The use of two critics mitigates overestimation bias by taking the minimum value, inspired by Double Q-learning [66].

3. *Replay Buffer*: A buffer $D_{\mathcal{R}}$ stores transitions $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', d)$, where $d \in \{0, 1\}$ indicates whether $\boldsymbol{s}'$ is terminal. This enables off-policy learning by sampling past experiences.

The critic is trained to minimize the Bellman residual, accounting for the entropy-augmented objective.

For a transition $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', d) \in D_{\mathcal{R}}$, the target value for critic is

$$y = r + \gamma(1-d)\mathbb{E}_{\boldsymbol{a}' \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}')} \left[\min_{i=1,2} Q_{\bar{\boldsymbol{\phi}}_i}(\boldsymbol{s}', \boldsymbol{a}') - \alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}' \mid \boldsymbol{s}') \right], \quad (2.22)$$

where,

$$\boldsymbol{a}' \sim \pi_{\boldsymbol{\theta}_i}(.|\boldsymbol{s}'),$$

and $Q_{\bar{\boldsymbol{\phi}}_i}$ are target critics with parameters $\bar{\boldsymbol{\phi}}_i$ updated via an exponential moving average $\bar{\boldsymbol{\phi}}_i \leftarrow \zeta \boldsymbol{\phi}_i + (1-\zeta)\bar{\boldsymbol{\phi}}_i$, with $\zeta \ll 1$. The critic loss is

$$\mathcal{L}_Q(\boldsymbol{\phi}_i) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}') \sim D_{\mathcal{R}}} \left[\left(Q_{\boldsymbol{\phi}_i}(\boldsymbol{s}, \boldsymbol{a}) - y \right)^2 \right], \quad i = 1, 2. \quad (2.23)$$

The actor is updated to maximize the expected action-value minus the entropy penalty

$$\mathcal{L}_\pi(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s} \sim D_{\mathcal{R}}, \boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s})} \left[\min_{i=1,2} Q_{\boldsymbol{\phi}_i}(\boldsymbol{s}, \boldsymbol{a}) - \alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}) \right]. \quad (2.24)$$

To ensure computational tractability, the expectation over actions is approximated using the reparameterization trick, where $\boldsymbol{a} = f_{\boldsymbol{\theta}}(\boldsymbol{s}, \epsilon)$, with $\epsilon \sim \mathcal{N}(0, I)$, allowing gradient-based optimization.

The temperature α is often learned by minimizing

$$\mathcal{L}_\alpha = \mathbb{E}_{\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}} \left[-\alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}) - \alpha \bar{\mathcal{H}} \right], \quad (2.25)$$

where $\bar{\mathcal{H}}$ is a target entropy, typically set to $-\dim(\mathcal{A})$ [69].

The overall SAC algorithm iteratively updates the actor and critics using mini-batches sampled from the replay buffer, ensuring stable and efficient learning. The policy learning using SAC is summarized in Algorithm 1.

Algorithm 1 Soft Actor-Critic (SAC)

Initialize replay buffer $D_{\mathcal{R}}$
Initialize actor parameters $\boldsymbol{\theta}$, critic parameters $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2$
Set target critic parameters $\bar{\boldsymbol{\phi}}_1 \leftarrow \boldsymbol{\phi}_1, \bar{\boldsymbol{\phi}}_2 \leftarrow \boldsymbol{\phi}_2$
for each iteration **do**
    for each environment step **do**
        Sample action $\boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t)$
        Execute $\boldsymbol{a}_t$ in the environment
        Observe reward r_t and next state $\boldsymbol{s}'_t$
        Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t, d_t)$ in $D_{\mathcal{R}}$
    end for
    for each gradient step **do**
        Sample mini-batch of transitions $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', d) \sim D_{\mathcal{R}}$
        Compute target value using Equation (2.22)
        Update critics by minimizing critic loss in Equation (2.23)
        Update actor for maximizing action-value in Equation (2.24)
        Update target critics: $\bar{\boldsymbol{\phi}}_i \leftarrow \zeta \boldsymbol{\phi}_i + (1-\zeta)\bar{\boldsymbol{\phi}}_i$ for $i = 1, 2$
        Optionally update α by minimizing cost in Equation (2.25)
    end for
end for

SAC's maximum entropy framework ensures robust exploration and its off-policy nature allows it to leverage diverse data sources, such as human demonstrations or simulated trajectories, reducing the need for costly real-world interactions. The use of two critics enhances stability, addressing the high-dimensional sensory inputs common in robotics.

However, SAC is susceptible to overestimation bias in value function approximations, which can degrade policy performance in precision-critical robotic applications. Overestimation bias is a tendency of RL methods to produce Q-value estimates that are higher than the true expected returns due to the noisy value estimates [67].

2.4.7 *Truncated Quantile Critics (TQC)*

The Truncated Quantile Critics (TQC), proposed by Kuznetsov et al. [67], enhances SAC by adopting a distributional RL approach and a truncation mechanism to mitigate overestimation bias, improving reliability.

Mechanics of TQC

TQC extends SAC by modeling the full distribution of the action-value function, $Z(\boldsymbol{s}, \boldsymbol{a})$, rather than its expectation, $Q(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}[Z(\boldsymbol{s}, \boldsymbol{a})]$. It uses a quantile-based representation, approximating $Z(\boldsymbol{s}, \boldsymbol{a})$ as a mixture of continuous quantile functions [67], and introduces a truncation

mechanism to control overestimation. The key components are in TQC are

1. *Distributional Critic*: For each state-action pair $(\boldsymbol{s}, \boldsymbol{a})$, the critic outputs a set of N quantiles $\{Z_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{a}, \tau_i)\}_{i=1}^{N}$, where $\tau_i \in [0, 1]$ are quantile levels (e.g., $\tau_i = i/(N+1)$). The action-value is approximated as the mean $Q_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{a}) \approx \frac{1}{N} \sum_{i=1}^{N} Z_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{a}, \tau_i)$. TQC uses M_c critics (e.g., $M_c = 2$) to enhance stability.
2. *Truncation Mechanism*: To mitigate overestimation, TQC discards the top k quantiles (where k is a hyperparameter) from each critic's distribution, computing a conservative action-value as the mean of the remaining $N - k$ quantiles. The final action-value is the minimum across the M_c critics' truncated means, further reducing optimism.

The critic is trained using quantile regression to align the predicted distribution with the target distribution.

For a transition $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}', d) \in D_{\mathcal{R}}$, the target quantile is

$$y_i = r + \gamma(1-d)\mathbb{E}_{\boldsymbol{a}' \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}')} \left[Z_{\bar{\boldsymbol{\phi}}}(\boldsymbol{s}', \boldsymbol{a}', \tau_i) - \alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}' \mid \boldsymbol{s}') \right],$$

where $Z_{\bar{\boldsymbol{\phi}}}$ is the target critic.

Distributional Quantile Loss

While SAC minimizes the mean-squared Bellman error of the scalar Q-function,

$$\mathcal{L}_Q(\boldsymbol{\phi}) = \mathbb{E}_{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}') \sim D_{\mathcal{R}}} \left[(Q_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{a}) - y)^2 \right],$$

TQC extends this idea to a distributional setting by predicting a set of N quantiles $\{Z_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{a}, \tau_i)\}_{i=1}^{N}$. The critic is trained to minimize the *distributional quantile loss*, defined as

$$\mathcal{L}_Z(\boldsymbol{\phi}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}') \sim D_{\mathcal{R}}} \left[\rho_{\tau_i}(y_i - Z_{\boldsymbol{\phi}}(\boldsymbol{s}, \boldsymbol{a}, \tau_i)) \right],$$

where $\rho_{\tau_i}(u) = |\tau_i - \mathbb{I}\{u < 0\}| \, L_\delta(u)$ is the quantile Huber loss [67]. In contrast to the scalar Q-function loss, this objective aligns the predicted distribution with the target across multiple quantiles, capturing both the mean and the uncertainty of returns.

2.4.8 *Aleatoric and Epistemic Uncertainty in RL*

Similar to LfD methods, two forms of uncertainty also arise in reinforcement learning, namely *aleatoric* and *epistemic* uncertainties. Aleatoric uncertainty reflects stochasticity inherent in the environment, such as noisy transitions or sensor disturbances, and is typically

modeled by learning distributions over returns rather than single-point value estimates. In contrast, epistemic uncertainty results from limited interaction data and expresses the agent's lack of knowledge about unexplored regions of the state–action space; unlike aleatoric uncertainty, it can be reduced through additional exploration and experience.

In our guided RL approach, KGRL, presented in Chapter 4, we specifically focus on aleatoric uncertainty which stems from *variability in demonstrations*, as described in Section 2.2.8. Variability in demonstrations reflects the confidence and diverse strategies of the teacher in the demonstrated motion at different phases of the task.

3

GUIDING REAL-WORLD REINFORCEMENT LEARNING FOR IN-CONTACT MANIPULATION TASKS WITH SHARED CONTROL TEMPLATES

In this chapter, we present *Reinforcement Learning with Shared Control Templates (RL-SCT)*, a framework that leverages Shared Control Templates (SCTs) to encode known constraints arising from object interactions and task geometry. In Chapter 1, we argued that leveraging hard-coded task knowledge and constraints can make RL both safer and more efficient, facilitating direct deployment on real robotic systems. RL-SCT explicitly models and integrates task constraints into the learning process, rather than implicitly embedding them in the reward function. See Section 2.3 for more details on SCTs.

This explicit representation reduces the effective state and action spaces, thereby lowering the dimensionality of the RL problem. The unknown aspects of the task are then learned by the RL agent through exploration within this constrained environment. RL-SCT accelerates learning by eliminating the need to rediscover modeled constraints, ensures safe exploration through explicit constraint enforcement, and simplifies reward design by removing the need to implicitly encode constraint violations, which are already prevented by the framework.

We evaluate RL-SCT on two representative manipulation tasks using a 7-DoF robotic arm: a pouring task and a grid-clamp placement task (analogous to peg-in-hole task). The proposed framework enables direct real-robot learning, with the pouring task converging in 65 episodes (16 minutes) and the grid-clamp placement task in 75 episodes (17 minutes). Both tasks achieve strong safety guarantees and rely on simple reward functions, demonstrating above-mentioned desirable properties of RL-SCT.

The remainder of this chapter has been published as [33] Abhishek Padalkar et al. "Guiding real-world reinforcement learning for in-contact manipulation tasks with Shared Control Templates." In: *Autonomous Robots* 48.4 (2024), p. 12.

3.1 INTRODUCTION

The potential of reinforcement learning (RL) in solving highly non-linear, high-dimensional problems is evident from its super-human level performance on Atari games [3] and in mastering Go [4]. RL can perform at its full potential when skills are learned from scratch, as evidenced by the example of AlphaGo Zero [6] (trained from scratch) outperforming AlphaGo [4] (pretrained with human games).

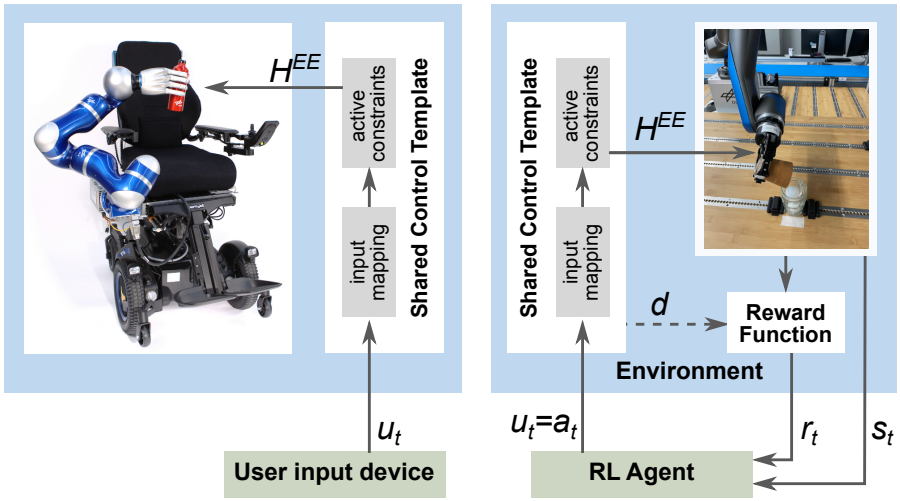
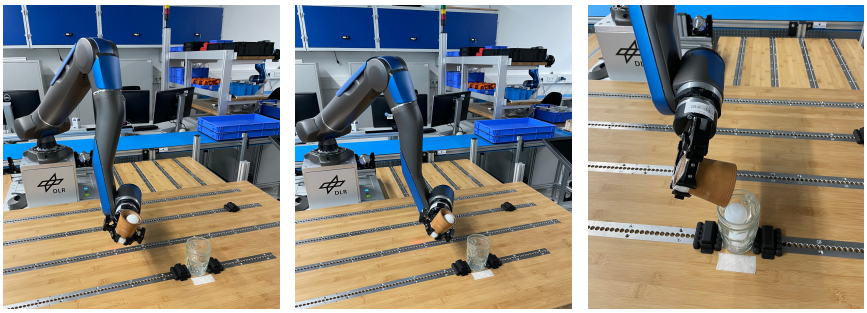


Figure 3.1: In previous work, we developed Shared Control Templates (SCTs) to support human users in executing tasks of daily living, such as pouring liquids or opening drawers [1]. In this context (left illustration), users provide commands to control the robot, which has many degrees of freedom. Efficiency and safety is improved with Input Mappings – which map low-dimensional user commands $\boldsymbol{u}$ to the many degrees of freedom of the robot $\boldsymbol{H}^{EE}$ – and Active Constraints – which limit the range of motion, for instance to avoid collisions.
In this work, we use SCTs to analogously achieve efficiency and safety in RL (right illustration). We also demonstrate how transition functions in the SCT's Finite State Machine (not depicted above; see Figure 3.3 for details) can be used to automatically generate shaped reward functions and further speed up learning. The resulting framework is called *Reinforcement Learning with Shared Control Templates* (RL-SCT). RL-SCT is evaluated on the two tasks illustrated in Figure 3.2.

In robotics, RL has piqued the interest of researchers by learning intricate and impressive motor skills such as juggling [8], ball-in-cup [10], in-hand manipulation [9], pick-and-place [11], or locomotion over highly variable terrain [12]. Although executing episodes on real robots is much more time-consuming than simulating Go games, it is still feasible and plays an important role in the acquisition of complex robot motor skills. However, the requirement for a large number of episodes has limited the application of RL in real-world object manipulation. Learning in simulation and then transferring skills to the real robot is limited by the accuracy of the simulation. Such limitations become even more apparent in tasks involving contacts, which are discussed in detail by [17]. In this chapter, we present a framework which enables learning directly on the real robot, safely and efficiently.

One of the aspects that sets human manipulation skills in daily activities (e.g. pouring a drink, opening a door) apart from the intricate



(a) Pouring task.

(b) Grid-clamp insertion.

Figure 3.2: Safer and more efficient RL with RL-SCT is demonstrated on the SARA robot [2] on two tasks: pouring and grid-clamp insertion. The latter involves contacts with the environment that are typically difficult to model in simulation. Our approach allows the robot to successfully learn the task while minimizing the interaction forces.

motor skills mentioned before is that these activities often involve interactions with objects that have a dedicated purpose (e.g. bottles, glasses, doors, sockets). Such objects are specifically designed with purposeful interactions in mind, i.e. they *afford* such interactions [25–27]. We consider it inefficient to use RL to re-learn such interactions – and the constraints arising from them – from scratch in thousands of episodes. What is known should not be learned.

In other words, the robot should not use RL to learn *what to do* with an object – this is already known and intrinsic to the object – but rather *how to do it well*, i.e. efficiently and without making many mistakes. This is especially true for tasks involving contact, such as insertion tasks, where the goal state is known but optimal behaviors (e.g. interaction forces) are difficult both to hardcode and demonstrate. In these cases, specifying partial constraints can greatly reduce the number of episodes needed to learn the task.

This approach raises the question of how such purposeful interactions and constraints involved should be represented and incorporated in the RL process. In our research on shared control [1], we have serendipitously discovered that representations for shared control are also ideally suited for guiding RL, for several reasons. First, shared

control aims at enabling users to control high-dimensional robotic systems with low-dimensional input commands. For RL, this approach substantially reduces the action space. Second, user preferences vary, and shared control must foster agency and empowerment by providing freedom of movement within the reduced action space. For RL, this means that there is room for exploration within such representations. Third, shared control often limits the range of motion for safety reasons. For RL, this also means that exploration becomes safe. This has the following advantages:

1. Safer learning, as constraints are enforced during learning;
2. Faster learning, as constraints need not be learned; and
3. Simplified reward function design, as constraints need no longer be implicitly represented in the reward.

Together, these advantages greatly facilitate skill acquisition on real robots via RL. Indeed, our key message in this chapter is that representing constraints implicitly in shaped reward functions, can require as much human design effort as explicitly modeling these constraints, but that the latter leads to safer and faster learning.

3.1.1 *Contributions*

In this chapter, we investigate how RL can be guided by the framework for shared control – Shared Control Templates (SCTs) [1] (see Section 2.3 for details), – thereby making it safer and more efficient. As shown in Figure 3.1, we replace the human user by an RL agent to complete the task autonomously. Our method, *Reinforcement Learning with Shared Control Templates* (RL-SCT), introduced in Section 3.3, allows us to model task space constraints and knowledge about the motion required for task completion flexibly, learn the complete task model from the robot-environment interactions, and optimize secondary costs associated with quality of the task.

The main contributions of the work presented in this chapter are:

1. A framework that leverages task knowledge represented in a shared control method to make reinforcement learning in the real world possible by explicitly constraining both state and actions spaces,
2. an approach to simplify the generation of shaped rewards based on prior information about task progress and completion encoded in SCTs, and
3. showing that applying SCTs to RL leads to more sample efficient and safer learning on a real robot even for tasks involving contact with environment.

Our approach is validated using a simulated KUKA IIWA robot on a pouring task and DLR's SARA robot on pouring and grid-clamp insertion tasks, the latter requiring contacts with the environment (Section 3.4). Our results show that RL-SCT avoids unnecessary mistakes during the learning process and enables faster learning for the tasks where the object interaction model is fully known. For tasks where only partial interaction modeling is possible, e.g. tasks with contacts such as the grid-clamp insertion, we show that RL-SCT makes learning faster, safer – both by avoiding collisions and minimizing interaction forces when they are required to complete the task – and hence possible directly on the real robot.

3.2 RELATED WORK

As contact dynamics are often difficult to model, tasks involving contacts are interesting for model-free RL [70, 71]. However, learning in-contact tasks with RL is challenging, as exploration can be unsafe due to collisions, and the number of episodes required to learn a task can be prohibitively high. Simulation and sim-to-real methods resolve these issues, but come with their own set of challenges when simulating in-contact tasks [21]. RL-SCT enables RL to be performed directly on the real robot, by ensuring safety through Active Constraints, and improving efficiency by providing low-dimensional action spaces and generating shaped reward functions.

In this section, we first describe RL approaches that are conceptually similar to (components of) RL-SCT. In Section 3.2.5, we then describe learning approaches that have been applied to in-contact manipulation on real robots.

3.2.1 *Safe and Efficient Reinforcement Learning with State-Dependent Action Masking*

In Reinforcement Learning, Markov Decision Process (MDP) is formalized as $(\mathcal{S}, \mathcal{A}, T, R)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function T, and reward function R. To speed up learning, several approaches to reduce and modify these spaces and functions have been proposed, including action masking, action manifolds, and reward shaping.

To improve the safety during exploration and/or to speed up learning, various approaches have been proposed that constrain the actions that can be performed in specific states. An example in the discrete domain is *action masking* [72], where the set of actions $\mathcal{A}'_{s_i}$ that is known to be invalid or suboptimal in certain states is excluded from the action space in those states, i.e. $\mathcal{A}'_{s_i} \subset \mathcal{A}$, where $\mathcal{A}'_{s_i}$ denotes the set of masked actions at s_i. As fewer actions need to be explored, action space shaping typically speeds up learning [73].

In robotics, similar concepts have been explored to also ensure safety during exploration. Cheng et al. [31] proposed to use *Control Barrier Functions* (CBFs) [74] to avoid unsafe states for the robot while an RL agent explored the state-action space.

Liu et al. [75] demonstrate that a constrained RL problem can be converted into an unconstrained one by performing RL on the tangent space of the constrained manifold. Their results obtained in simulation are promising, but the approach is limited to problems where the constraints are differentiable, which restricts its application domain. RL-SCT is free of differentiability assumptions, making it flexible by design.

Affordances in RL

Khetarpal et al. [27] apply the theory of affordances [25] to RL, which allows faster learning by action space reduction and precise learning of transition models. First, 'intent' is defined as the desired outcome of an action. Affordances then capture the subset of the state-action space where the intent is achieved. Such subset represents a partial model of the environment. Intents and affordances are a principled two-step procedure for defining state-dependent actions masks. This approach is also noteworthy for combining action masking with a decomposition of the overall policy into different intents and affordances, see also Section 3.2.3. Within RL-SCT approach, each state can be considered to have an intent for a subphase of the task, and the Active Constraints implement the continuous action masking. RL-SCT adds to this the Input Mappings, the automatic generation of shaped rewards, and the representation of subtasks in a finite state machine. The method in [27] is demonstrated on grid worlds and continuous 2D planes, and has, to the best of our knowledge, not yet been scaled up to the high-dimensional action spaces of robots.

Affordance-based learning has also been used to generate high-level skill sequences for domestic tasks [76–78]. A human guides the agent with instructions adhering to the affordances. In RL-SCT, we rather consider low-level actions that control end-effector positions, for instance for in-contact manipulation.

3.2.2 *Efficient Reinforcement Learning with Action Manifolds*

An alternative to defining state-dependent action masking/constraints is to define a reduced action space for the entire state space.

Luck et al. [79] combine policy search and dimensionality reduction in an expectation-maximization framework and use probabilistic PCA for obtaining a latent space during the learning process.

Kolter, Ng, et al. [80] present a method that uses a (possibly inaccurate) simulator to identify a low-dimensional subspace of policies.

An instance on a real system is learned using these low-dimensional policies using much less data.

Padalkar et al. [81, 82] proposed a partial task specification in the robot task space using the Task Frame Formalism [83, 84], learning the remaining specification of the task with RL. This method successfully demonstrated a significant reduction in robot-environment interactions when cutting vegetables with a light-weight robot. Although this work ensures safety in the directions in which motion is fully specified, it does not guarantee the safety in the directions explored by the RL policy, a problem addressed in RL-SCT.

Reinhart and Steil [85] propose a two-step solution with a skill memory, an organization of motion primitives in low-dimensional, topology preserving embedding space and policy search leveraging the low-dimensional skill parameterization. Skill parameterization can be predefined or automatically discovered with parameterized self-organizing maps.

He and Ciocarlie [86] present a framework that discovers a synergy space and learns a multi-task policy that operates on this low-dimensional action space. Learned synergies can be used across multiple manipulation tasks.

In this chapter, we refer to action manifolds as *Input Mappings*, a term used in SCTs due to their role in shared control, where user *inputs* (e.g. through a joystick) are *mapped* to robot actions [87].

Manifold learning has also been applied to reducing the state space (rather than the action space as described above), for instance using PCA and GPLVM [88–91]. In DeepRL this is known as *state representation learning* [92], and the automatic learning of the state space representations is the main motivations for using DeepRL. A full overview of this field beyond the scope of this paper, as our focus in this work is not on learning state space manifolds.

3.2.3 *RL with Sequential Tasks and/or Task Decompositions.*

An early RL approach to representing subtasks is the *options* framework, where the agent uses macro actions that span multiple time steps to represent subpolicies for subtasks [93, 94]. Our work differs in that the subtasks are represented in the SCT, and the SCT is part of the environment, rather than the agent. The RL agent does not represent subpolicies or subtasks internally, as will be highlighted in Figure 3.3.

Daniel et al. [95] propose an approach to learning sequencing motor primitives while simultaneously improving individual motor primitives.

Kroemer et al. [96] propose to learn a probabilistic multiphase model of the task, a motion primitive for each task phase, and a RL policy for sequencing the motion primitives that use the task model. Although

these methods solve multiphase tasks; they do not take constraints in the environment into account, and hence exploratory actions can lead the robot into potential hazardous situations during the learning process. In RL-SCT, we propose to learn a single policy for all phases of the task, whereas the above-mentioned approaches use different motion primitives for different phases.

Krishnan et al. [97] propose an approach where inverse RL learns subtasks with subgoals and local cost functions from a latent space representation of demonstrations obtained with unsupervised learning. Such latent space and decomposed representation is used to accelerate RL.

Koert et al. [98] present an approach where a robot learns and improves to combine skills for sequential tasks with human input during learning to accelerate the learning. It is an interactive framework where human advises robot on planned high-level action, provides feedback on the outcome of the action, and provides subgoal rewards. This approach uses human-guided RL for solving decision-making problems while choosing predefined low-level policies, whereas we propose to learn control policies with RL that generate inputs for underlying MDP simplified by SCTs by incorporating the task knowledge.

Khetarpal et al. [27] developed theory of affordances [25] for RL agent, which allows faster learning by action space reduction and precise learning of transition models. With intent defined as the desired outcome of an action, affordances capture the subset of state-action space where the intent is achieved. Such subset represents a partial model of the environment. In RL-SCT, each SCT-state models an intent and define IM and AC to achieve the intent with state dependent action space reduction.

3.2.4 *Reward Shaping*

In reward shaping, a sparse terminal reward R is transformed into a dense, immediate reward R' by adding knowledge about the task and its subphases [99]. It does not change $\mathcal{S}$, $\mathcal{A}$, or T. Shaped rewards are more informative, and thus speed up learning. In this chapter, we argue that action space shaping is more effective that reward shaping, and contrary to common belief, often not more difficult to design.

3.2.5 *Reinforcement Learning of Manipulation Tasks with Contacts*

Zhao et al. [100] presented an approach of meta-reinforcement learning by encoding human demonstrations for different tasks in a latent space and using the latent space variables to generalize the RL policy for different types of insertion tasks. The policy can be trained offline with data collected from sources like human demonstrations, replay

Criteria	On real robot	Guidance	f_{ext} as state	f_{ext} in reward
[100]	✓	D	-	-
[101]	✓	D, S	✓	-
[102]	✓	D	-	-
[103]	✓	-	✓	✓
[104]	✓	C	-	-
[105]	✓	C	-	-
[106]	✓	C	-	-
[107]	✓	S	✓	✓
[108]	✓	-	✓	-
[70]	✓	D	-	✓
[71]	✓	D	✓	✓
RL-SCT (our approach)	✓	C	✓	✓

Table 3.1: Comparison with state-of-the-art approaches on learning in-contact tasks using RL. f_{ext} denotes contact forces with the environment. (D: Demonstrations, C: Constraints and residual learning, S: Pretrained in simulation)

buffers from previous experiments and data collected from hand-coded solutions.

Vecerik et al. [102] train a neural network to extract features from images during insertion task, with the same neural network being used to compute a binary reward. The features are used as input to the RL policy. Critic and actor are pretrained to mimic human demonstrations. Davchev et al. [106] present an approach to learn insertion task by learning a residual policy to support an imitation learning policy learned with Dynamic Motion Primitives. Kozlovsky, Newman, and Zacksenhouse [105] presented an approach to learn asymmetric impedance matrices to learn a policy in simulation and then transfer the solution to the real robot. Lee et al. [104] combine an RL policy with a model-based solution in the region where the task model is uncertain to learn the insertion task with a binary reward. The poses of the objects used as state are estimated by a vision system. All the above approaches do not include force measurements in the state and do not optimize interaction forces while executing the task. We address this limitation in RL-SCT by introducing interaction forces

in the agent state and a secondary cost in the reward to minimize interaction forces.

Apolinarska et al. [101] applied RL for assembly of timber joints. They use human demonstrations for initializing the policy. The policy is learned in simulation with domain randomization and then transferred to the real robot. Luo et al. [108] also presented an approach to learn assembly tasks by learning a variable impedance controller. They use end-effector force/torque readings filtered by a low-pass filter and directly inject them in the second layer of the neural network representing the policy in order to provide direct haptic information to the policy. These approaches use interaction force in the state given to the RL policy but do not optimize interaction cost.

Kim, Na, and Song [107] presented a method to learn impedance parameters using RL for insertion tasks and then transferred the learned policy to a real robot which presents a need for fairly accurate simulation. Beltran-Hernandez et al. [103] proposed a method to learn force controllers on a position-controlled robot using an end-effector force/torque sensor with RL. Both of these approaches use force data in the state and try to optimize interaction forces. Beltran-Hernandez et al. [103] do not use any method to guide or constrain RL exploration and hence results in collision during the learning.

Table 3.1 compares the approaches for learning contact tasks to RL-SCT. We address the various shortcomings of this related work in RL-SCT by leveraging pre-specified task knowledge to improve sample efficiency and simplifying reward function design. RL-SCT facilitates direct learning on real robots hence alleviating the need for simulation. Simplified reward functions in RL-SCT allow us to optimize secondary costs, e.g. interaction force costs, right from the beginning of learning, producing significantly lower forces during learning and resulting in learned policies that generate minimal interaction forces with the environment.

3.3 Shared Control Templates for Reinforcement Learning

As discussed in Section 2.3, Shared Control Templates (SCTs) support a disabled person in controlling a complex system like EDAN (*EMG-controlled Daily AssistaNt*) [58], which consists of an electric wheelchair with an articulated arm. SCTs map 3D input signals extracted from surface electromyography to 6D end-effector movements, which are then mapped to the movement of the overall 11 DoF EDAN system through whole-body motion control [1]. A key aspect of SCTs is that they facilitate task completion through shared control by defining task-relevant Input Mappings (IMs) and Active Constraints (ACs), but the user always remains in control, i.e. determines the speed of movement, the amount of water that is poured, etc. Additionally,

SCTs structure tasks into phases represented as states in a Finite-State Machine (FSM), with transitions triggered by monitoring task-relevant transition distance functions.

The key insight that we propose is that *components that facilitate human control of the robot through shared control are conceptually very similar to those used to facilitate reinforcement learning*. Our aim is to demonstrate this conceptually and empirically.

Figure 3.3 shows how SCTs are integrated in the RL agent-environment interface. In this section, we describe how the main components of SCTs – Input Mappings, Active Constraints, and the Transition Functions between the FSM states – are conceptually similar to state-dependent action space shaping (see Section 3.2.1), action manifolds (see Section 3.2.2), and reward shaping (see Section 3.2.4), respectively.

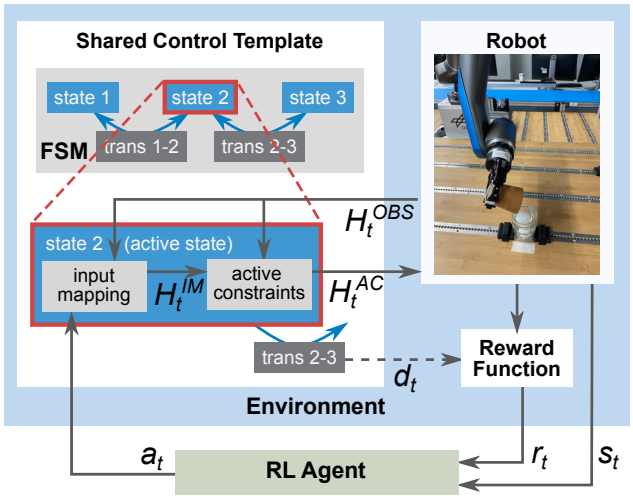


Figure 3.3: Agent-environment interface in RL-SCT components. The input $\boldsymbol{u}_t = \boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$ to the IM of an SCT is computed by the RL policy π, $\boldsymbol{s}_t$ is the state for the RL agent. d_t is the distance function to determine transitions in the FSM; it can be used as a shaped reward function. All time-varying variables are annotated with the same t, in practice the agent-environment loop and the controllers using the end-effector poses H may run at different frequencies.

3.3.1 *Action Space Shaping with SCTs*

Using IMs to map 3D user commands to 6D Cartesian commands, the action space $\mathcal{A}$ of the MDP is reduced to 3D. Although IMs can accept inputs up to 6D, more than 3 inputs are rarely needed in most tasks [1].

The application of AC is conceptually equivalent with continuous state-dependent action masking, in that not all actions defined by the IM have an effect[1].

Typically, in RL, constraints are encoded *implicitly* in the reward function, through reward terms that encourage actions that do not violate them. A direct consequence of such approach is that an agent will need to violate these constraints in order to learn them. In many robotic tasks, violating constraints can lead to physical damage. SCTs address the above issues by precluding the robot from violating constraints through the definition of Active Constraints.

3.3.2 *Reward Shaping with SCTs*

In RL, the main role of the reward function R is usually to provide feedback about whether the task has been achieved, e.g. was the exit to the maze found, was water poured into the glass, etc. We call this the *primary reward*. Designing a primary reward function is easiest if this reward is sparse (e.g. 1 or 0) and terminal (i.e. r_T^{prim} is given at the end of an episode) [109]. However, this is also the least informative type of reward function.

Primary reward shaping with the SCT transition distance

Reward shaping is the process of redesigning the sparse primary reward function so that it becomes dense and/or immediate, to make the reward function more informative and speed up learning. The process of designing an SCT is similar to that of shaping a reward function; we now explain how a shaped reward function can be generated automatically from an SCT.

In an SCT, the transition distance functions that yield d_t are designed in such a way that it monotonically decreases as the robot moves towards the next phase of the task. As this constitutes a gradient towards the overall task, d_t can be used directly as immediate primary rewards in addition to a terminal sparse reward, as follows:

$$r_t = r_T^{\text{prim}} \qquad \text{Sparse} \tag{3.1}$$

$$r_t = r_T^{\text{prim}} + k(d_{t-1} - d_t), \qquad \text{Shaped} \tag{3.2}$$

where k is a constant and r_T^{prim} a terminal reward which is zero except at $t = T$. Note that the term $d_{t-1} - d_t$ encourages progress towards the next SCT state, and thus task completion, by rewarding a decrease in transition distance.

1 The underlying implementation is slightly different, in that the action is projected into the future in state space, and the resulting end-effector position is projected back onto a constraint if the constraint is violated [1].

Secondary costs

While the primary reward provides feedback about task completion, in robotics it is also desirable for movements to have low accelerations and low force interactions. As such measures can commonly be measured at each time step, we add M such measures to the immediate rewards as

$$r_t^{\text{secon}} = -\sum_{i=1}^{M} \boldsymbol{v}_{i,t}^{\top} \boldsymbol{R}_i \boldsymbol{v}_{i,t}, \tag{3.3}$$

where $\boldsymbol{v}_{i,t}$ is a vector representing a physical quantity whose magnitude is to be minimized during learning, and $\boldsymbol{R}_i$ is a diagonal matrix with positive entries representing a weighting factor. In this work we focus on two types of secondary rewards that penalize actions $\boldsymbol{a}_t$ and interaction forces $\boldsymbol{f}_t$, with which the overall cost function becomes:

$$r_t = \underbrace{\overbrace{r_T}^{\text{Sparse}} + \overbrace{k(d_{t-1} - d_t)}^{\text{Shaped part}}}_{\text{Primary}} \underbrace{-\boldsymbol{a}_t^{\top} \boldsymbol{R}_a \boldsymbol{a}_t - \boldsymbol{f}_t^{\top} \boldsymbol{R}_f \boldsymbol{f}_t}_{\text{Secondary}}. \tag{3.4}$$

3.4 EVALUATION

We first evaluated RL-SCT against RL without SCT in simulation on a pouring task with a KUKA IIWA (Figure 3.4), and perform the same task directly on the real 7-DoF SARA robot (shown in Figure 3.2a). We used Bullet Real-Time Physics Simulation [110] for simulating the pouring liquid task. To showcase the safe and efficient learning properties of our method, we also learn a grid-clamp placement task (Figure 3.2b), showing how RL-SCT allows tasks involving contacts with the environment to be learned directly on the real robot. These experiments on the real robots aim at demonstrating the ability of RL-SCT to learn the tasks directly on the real robot, safely and efficiently.

As RL-SCT performs action space shaping and reward shaping in the environment (see Figure 3.3), the underlying policy representation and RL algorithm used by the agent are unaffected by applying RL-SCT. To highlight this, we apply both Soft Actor Critic (SAC) [62] and Truncated Quantile Critics (TQC) [67] from the open-source implementation in Stable-Baseline3 [111].

The hyperparameters for these algorithms were configured for experiments involving the IIWA robot in simulation (pouring task) and the real SARA robot (pouring task and grid clamp insertion task). Table 3.2 summarizes key hyperparameters used for each setup. In all experiments, the policy is a feed-forward neural network with 2 hidden layers with 256 neurons in each hidden layer. The weights of the neural network are initialized to random values.

Table 3.2: Hyperparameters for SAC and TQC in robotic experiments.

Hyperparameter	SAC (IIWA)	TQC (IIWA)	SAC (SARA)
Learning Rate	8×10^{-4}	8×10^{-4}	1×10^{-3}
Replay Buffer Size	10^6	10^6	10^6
Discount Factor	0.95	0.95	0.98
Soft Update Coefficient	0.02	0.02	0.02
Training Frequency	8	8	8
Gradient Steps	8	8	8
Learning Starts	1000	1000	1000
Batch Size	256	256	256

3.4.1 *Pouring Task*

The pouring task consists of transferring the liquid from a container (thermos) attached to the end-effector of the robot to a target container (mug) placed in the environment. The SCT for the pouring task is visualized in Figure 2.2 and explained in Table 3.3. In our experiments, the liquid is replaced by two ping-pong balls with 4cm diameter, as illustrated in Figures 3.2a and 3.4.

The SCT for this task was taken *as is* from our previous work on assistive robotics [1], where it was shown that using the SCT for shared control enables users to complete the pouring task more than twice as fast, on average. SCTs were an essential component for winning the Cybathlon Challenges for Assistive Robots in 2023 [112, 113].

The state for this task is $\boldsymbol{s}_t = \boldsymbol{x}_{\text{th},t}$ where $\boldsymbol{x}_{\text{th},t}$ is the 6D pose of the thermos tip expressed in the mug tip frame. In RL-SCT, as shown in Figure 3.3, a policy generates an action $\boldsymbol{a}_t \in \mathbb{R}^3$ which acts as input $\boldsymbol{u}_t$ to the SCT, which in turn generates a desired end-effector pose (Section 2.3). When running RL without SCT in the ablation study, the policy generates the 6D end-effector velocity as action $\boldsymbol{a}_t$ which is used to compute the target end-effector pose. The end-effector poses are given as references to the Cartesian position controller, which runs at 100Hz for the simulated KUKA IIWA robot and 8KHz for the SARA robot.

On the real robot, the pose of the mug in the robot base frame was fixed and known beforehand. The pose of the thermos, grasped by the robot gripper, was calculated from the forward kinematics of the robot.

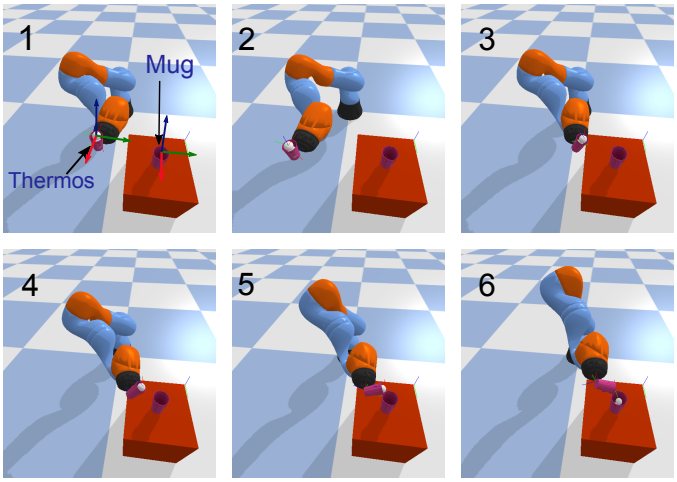


Figure 3.4: Experimental setup for the pouring task with simulated KUKA IIWA holding the thermos and the target mug placed on the table with thermos tip and mug tip coordinate frames.

Reward functions

We evaluate our approach by comparing the above mentioned four scenarios. The baseline uses RL without SCTs, with a designed reward function, which is either shaped (RL-Shaped) or sparse (RL-Sparse). Our proposed method based on SCTs is also evaluated with a shaped (RL-SCT-Shaped) and sparse (RL-SCT-Sparse) reward function.

Sparse reward function:

For the pouring task, the sparse reward function used in both RL-Sparse and RL-SCT-Sparse is

$$r_T = \begin{cases} 200, & \text{on successful termination} \\ -10, & \text{on termination due to collision} \\ 0, & \text{otherwise,} \end{cases} \tag{3.5}$$

$$r_t = r_T - \boldsymbol{a}_t^\top \boldsymbol{a}_t, \tag{3.6}$$

where $\boldsymbol{a}_t^\top \boldsymbol{a}_t$ a secondary cost component related to the action magnitude.

The task is considered successful only if both balls are successfully poured into the mug. In the experiments with the real robot, a human observing the task provided the feedback about the success of the task. The task is considered failed if the robot collides with the mug or the table, one or both balls are spilled out of the thermos or a pre-defined time limit in terms of time steps per episode is reached. In the event of collision with the table or target mug, the episode is terminated.

This sparse reward function is not very informative, as the primary reward consists only of three discrete rewards given only at the end of the episode.

Shaped reward function from the SCT

The reward function for RL-SCT-Shaped is automatically derived from d_t in the SCTs, as described in Section 3.3.2, i.e.

$$r_t = r_T + 200(d_{t-1} - d_t) - \boldsymbol{a}_t^\top \boldsymbol{a}_t, \tag{3.7}$$

where r_T is the same as in Equation (3.6).

Hand-designed shaped reward function

We also hand-designed a shaped reward function for the pouring task. Our aim here is to show the intricacy of the design process for manually shaped reward functions, and its similarity to the process of designing an SCT.

First, the pouring task is divided in two phases, 1) transport the thermos near the mug without spilling the liquid, and 2) pour the liquid by tilting the thermos around the appropriate axis. The translation phase takes the thermos to a fixed distance near the target mug, and the pouring phase rotates the thermos avoiding any translation. These phases need to be identified correctly, and give the reward for not tilting the thermos in the first phase and reward for tilting around the correct axis and not around the other axes in the second phase.

The implementation of this reward function for RL-Shaped is given by Equation (3.8), where j_t is the distance between the thermos tip and the mug tip, and ψ_t^x, ψ_t^y and ψ_t^z are the angular positions, expressed as Euler angles (roll, pitch and yaw) of the thermos tip in the mug tip frame (orientations of the tip frames are depicted in Figure 3.4).

$$r_t^{\text{prim}} = r_t^{\text{dist}} + r_t^{\text{tilt}}/4 - \boldsymbol{a}_t^\top \boldsymbol{a}_t + r_T, \tag{3.8}$$

$$r_{\text{dist}} = \begin{cases} 15(j_{t-1} - j_t), & \text{if } j_t > 0.04 \\ 15(j_{t-1} - j_t) + 0.2, & \text{otherwise} \end{cases} \tag{3.9}$$

$$r_t^{\text{tilt}} = \begin{cases} -|\delta\psi_t^x| - |\delta\psi_t^y| - |\delta\psi_t^z|, & \text{if } j_t > 0.04 \\ 20\delta\psi_t^x - |\delta\psi_t^y| - |\delta\psi_t^z|, & \text{otherwise} \end{cases} \tag{3.10}$$

In the design of the shaped reward function, we recognize many similarities to SCT design, e.g. the definition of phases, transition thresholds between the phases, definition of constraints, etc. Table 3.3 provides a full analysis of the similarities.

Results

Figure 3.5 and Figure 3.6 show the rate of success, spillage and collision with the environment in simulation both with SAC and TQC. The curves show the mean and standard deviation obtained from 10 different learning sessions. The results from the different combinations of reward types and presence/absence of SCTs give important insights

Task state	Classical RL reward function	RL-SCT
Translational	Reward for moving towards target, penalty for spilling (by directly penalizing rotational motion) and penalty for collision	**IM:** 3 inputs mapped to translational motion (No rotational motion needed) **AC:** Stay above table to avoid collisions **Transition distance:** Distance of mug tip from thermos tip
Tilt	Reward for moving and tilting towards the target simultaneously, penalty for spilling liquid (by penalizing translational and rotational motion in undesired directions), penalty for collision	**IM:** 3 inputs mapped to translational motion (Rotational motion is controlled by AC) **AC:** Stay above table to avoid collisions, tilt towards the target depending on distance **Transition distance:** Distance of mug tip from thermos tip
Pour	Reward for tilting towards the mug, penalty for spilling liquid (by penalizing translation to avoid spillage outside the target), and penalty for collision	**IM:** 1 input mapped to Z-axis translational motion, the others to the tilting motion (no horizontal motion) **AC:** Stay above table and mug to avoid collision **Transition distance:** Tilting angle

Table 3.3: Comparison of knowledge modeled using typical RL reward functions and RL-SCT for a pouring task.

about the influence of SCT on the learning process, and failures during learning.

Using SAC (Figure 3.5) RL-Sparse is not able to learn the task in 1400 (maximum) episodes [A]. RL-Shaped converges within 1200 episodes [B], but also shows very high spill rate and collision rate due to unconstrained exploration [C]. RL-SCT-Sparse converges within 600 episodes [D], shows very low spill rate and no collisions [E] due to the constraints. RL-SCT-Shaped converges within 250 episodes [G] showing the best performance overall, with small initial spill rate [F] and no collisions.

Using TQC (Figure 3.6) RL-Sparse is not able to learn the task in 1400 (maximum) episodes [H]. RL-Shaped converges within 700 episodes [I], but also shows very high spill rate and collision rate due to unconstrained exploration [J]. RL-SCT-Sparse converges within 1000 episodes [K] and shows very low spill rate and no collisions [L] due to the constraints. RL-SCT-Shaped converges within 250 episodes [M] showing the best performance overall, with small initial spill rate [N] and no collisions. Figure 3.8 summarizes these results.

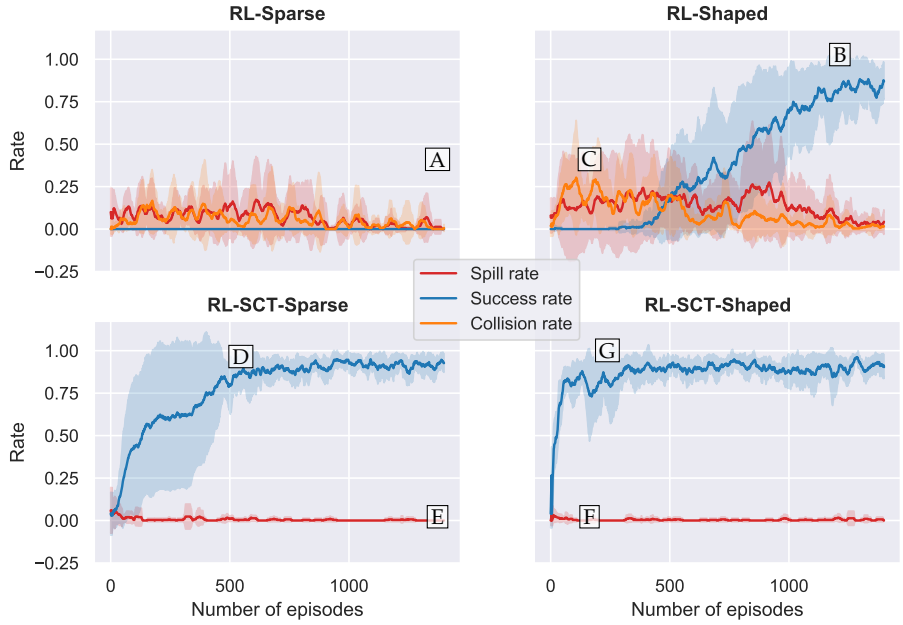


Figure 3.5: Success, spill and collision rates vs number of episodes, in different experimental settings in simulation using SAC. Each point shows the average of 10 learning sessions, together with one standard deviation.

The results on the real robot are shown in Figure 3.7. As we wanted to avoid collisions as much as possible during training, and running more than 1000 episodes multiple times takes prohibitively long on the real robot, we only ran RL-SCT-Shaped with SAC. Figure 3.7 shows the results of 5 independent learning sessions. The learning agent achieves a success rate of 1 within 65 episodes amounting to 9700 time steps, on average. This corresponds to 16 minutes of training time, without considering the time taken for resetting the environment.

Discussion

From the learning curves and the summary in Figure 3.8, we derive the following conclusions. As expected Shaped (yellow/green) outperforms Sparse (red/blue) by several orders of magnitude with respect to convergence speed as shaped rewards are more informative (top two graphs). The boxplots further confirm that SCTs speed up learning (blue vs. red and green vs. blue); indeed RL-Sparse never converges within 1400 episodes in any of the 10 learning sessions.

From an RL perspective, speed of convergence and the rewards achieved after convergence are the most important measures of success. From a robotics perspective, safety is just as important, and this aspect is highlighted in the lower two rows of boxplots. We observe that with SCTs (boxplots to the right of the vertical gray lines), there are hardly any collisions or spills; especially the 0 spills and 0 collisions on the robot are of importance. With SAC, the median percentage

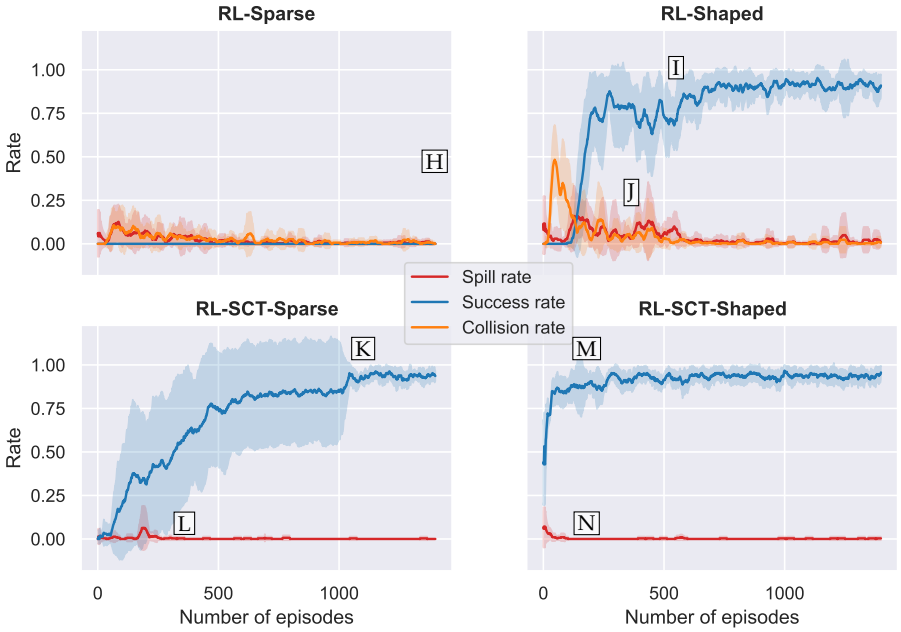


Figure 3.6: Success, spill and collision rates vs number of episodes, in different experimental settings in simulation using TQC. Each point shows the average of 10 learning sessions, together with one standard deviation.

(over 10 learning sessions) of episodes that involves collision is 12 and 20 for sparse/shaped rewards respectively (bottom left graph, red/yellow) . On a real robot, this would lead to an unacceptable amount of wear-and-tear, which is why we do not run experiments on the robot without SCT. This confirms that RL-SCT leads to safer learning, enabling RL directly on the real robot.

In comparison to SAC, we see a much lower rate of spills and collisions without SCTs (red/yellow) with TQC, i.e. between 2-5%. This is because in many learning sessions, the robot learns to not move at all. The robot does not receive the reward for completing the task then, but it also does not get the penalties for collisions. The reason for the low rates is not that collisions are avoided during the movement; rather there is hardly any movement at all. With RL-Shaped, we see more collisions and spills for TQC than for RL-Sparse. This is because the reward gradient in the shaped reward leads to more movement than the sparse reward.

The results show that RL-SCT can learn the multi-phase pouring task on the real robot safely (no collisions) and efficiently (convergence in less than 100 episodes). Furthermore, safe exploration is not sacrificed when using a sparse reward function.

In these experiments, reward shaping was essential to making RL without SCTs feasible within 1400 episodes. Therefore, if the RL expert needs to invest time in designing a complex multi-phase shaped reward function as in Equation (3.8) to make learning feasible, we

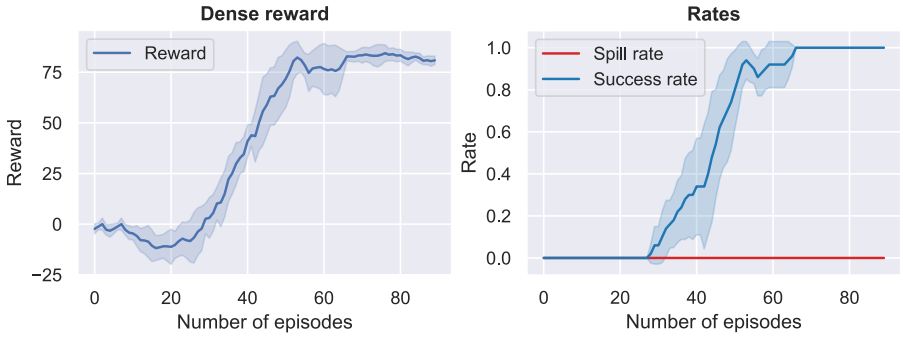


Figure 3.7: Success rate, spill rate and reward for RL-SCT-Shaped on the real robot using SAC. Each point shows the mean and one standard deviation over 5 learning sessions.

argue that this time is better invested in designing *explicit* multi-phase constraints. This will speed up learning even more, and, critically, ensure safety during learning, as constraints no longer need to be violated in order to learn them. What is known need not be learned.

3.4.2 *Grid Clamp Insertion Task*

In order to evaluate the ability of RL-SCT to learn tasks involving contacts, a grid-clamp insertion task was learned on the SARA robot (Figure 3.2b). Grid clamps are used in DLR's Factory of Future setup to reconfigure variable workstations. The robot learns to insert the grid-clamp into the grid holes on the table, similar to a peg-in-hole task.

During the insertion process, 5 peg-like heads, which secure the grid-clamp on the table, are needed to be inserted simultaneously and snapped into the holes on the table, as shown in Figure 3.9 (top). The task is challenging due to the kinematic inaccuracy mainly in the horizontal plane (x–y), the pose of the grid-clamp (estimated using forward kinematics of the robot) and the holes on the table. Additional inaccuracies arise from the grasp of the grid-clamp, as shown in Figure 3.9, as well as the inherent inaccuracy of the Cartesian impedance controller and table pose calibration.

In our experiments, we place the grid-clamps at 6 different locations to be picked up by the robot, with a user resetting the setup every 6^{th} episode. This way we ensure that the robot learns under uncertainties arising from both the grasp and the robot configuration. During the experiments, the grid-clamp is grasped and transported above the hole using a hand designed grid-clamp pick up skill. The RL policy takes over when the grid-clamp is above the target hole.

It is possible to complete the task successfully despite the above-mentioned uncertainties by appropriately reacting to the contact forces. While the task can still be executed successfully with high contact forces, minimizing them is critical for safe long-term operation of

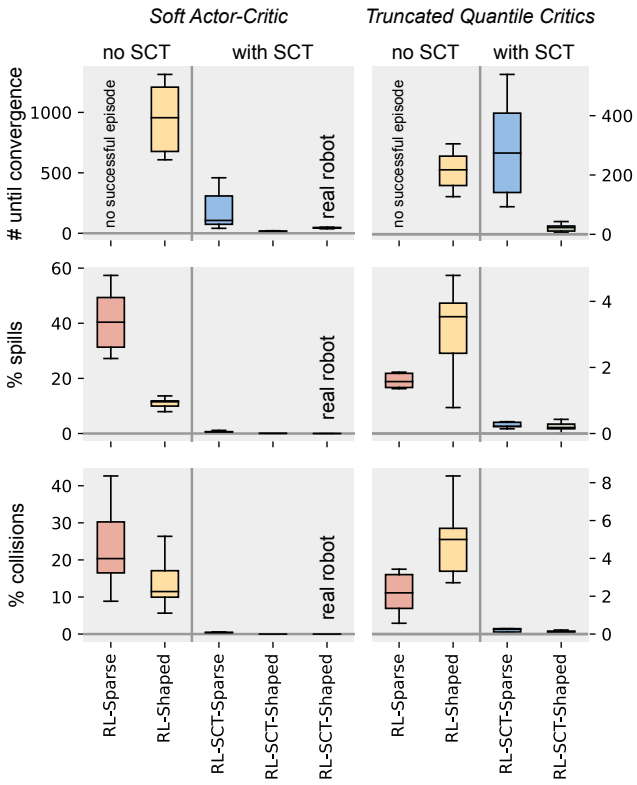


Figure 3.8: Boxplots summarizing the liquid pouring experiments. Left: SAC (including real robot experiment), Right: TQC. '# until convergence': number of episodes until 10 subsequent episodes achieve the task. '% spills': number of episodes in which a spill occurred. '% collisions': number of episodes in which a collision occurred.

the robot. We therefore minimize contact forces by including them as a secondary reward, with the primary reward ensuring the task completion.

The SCT used for learning this task has the following design:

- **Phases:** one phase is used, governing the translational motion of the peg towards the hole.
- **Constraints:** the rotational degrees of freedom are constrained (we assume negligible uncertainty on the hole orientation) and the robot is free to move in translational DoFs within a cuboid constraint of size 0.8cm $\times$ 0.8cm $\times$ 10cm.
- **Transitions:** the transition distance d_t is the z-axis distance between the grid-clamp and the hole.

For the grid-clamp insertion task (Section 3.4.2), the 6D state given to the policy is $\boldsymbol{s}_t = [\boldsymbol{x}_t^\top \ \boldsymbol{f}_t^\top]^\top$ where $\boldsymbol{x}_t \in \mathbb{R}^3$ is the position of the grid-clamp (attached to the robot gripper) expressed in the hole frame and $\boldsymbol{f}_t \in \mathbb{R}^3$ is the force measured at the grid-clamp frame by the integrated force/torque sensor.

Figure 3.9: Bottom view of a grid-clamp (left) and uncertainty when grasping grid-clamps (middle/right).

Reward function

Using the outlined SCT, the policy mainly has to learn the contact dynamics during insertion. Particularly, the robot is encouraged to reduce the contact force by introducing a secondary cost component for the measured force $\boldsymbol{f}_t$ at the grid-clamp frame. The reward function is thus given by:

$$r_T = \begin{cases} 50, & \text{on successful termination} \\ 0, & \text{otherwise.} \end{cases} \tag{3.11}$$

$$r_t = r_T + 300(d_{t-1} - d_t) - \frac{1}{12}\boldsymbol{f}_t^\top \boldsymbol{f}_t - \boldsymbol{a}_t^\top \boldsymbol{a}_t. \tag{3.12}$$

We determine task success from the z coordinate of the end-effector; if it drops below a threshold, this indicates the clamp has been placed successfully. To evaluate the effect of the interaction force cost on the interaction forces during learning, two sets of experiments were conducted: 1) using the interaction force cost in the reward function ($\boldsymbol{f}_t^\top \boldsymbol{f}_t$) and 2) without using interaction force costs.

Results

In both cases, with and without using interaction force cost in the reward function, the robot learns the grid-clamp insertion task in less than 70 episodes amounting to 9820 time steps ($\approx$ 17 minutes) as shown in Figure 3.10. It achieves 100% success in 70 episodes [O]. Figure 3.10 also shows the average interaction force per episode, computed, for episode i, as $\boldsymbol{f}_i = \sum_{n=1}^{N_i} \boldsymbol{f}_{i,n}^\top \boldsymbol{f}_{i,n} / N_i$, where $\boldsymbol{f}_{i,n}$ is the interaction force measured at step n and N_i is the number of steps in the episode.

Comparison of the average interaction forces, [P] and [Q], shows that the robot uses significantly less force during the learning process when the reward function contains the secondary cost term associated with interaction forces. This happens without significantly affecting other components of learning, particularly the time required to achieve 100%

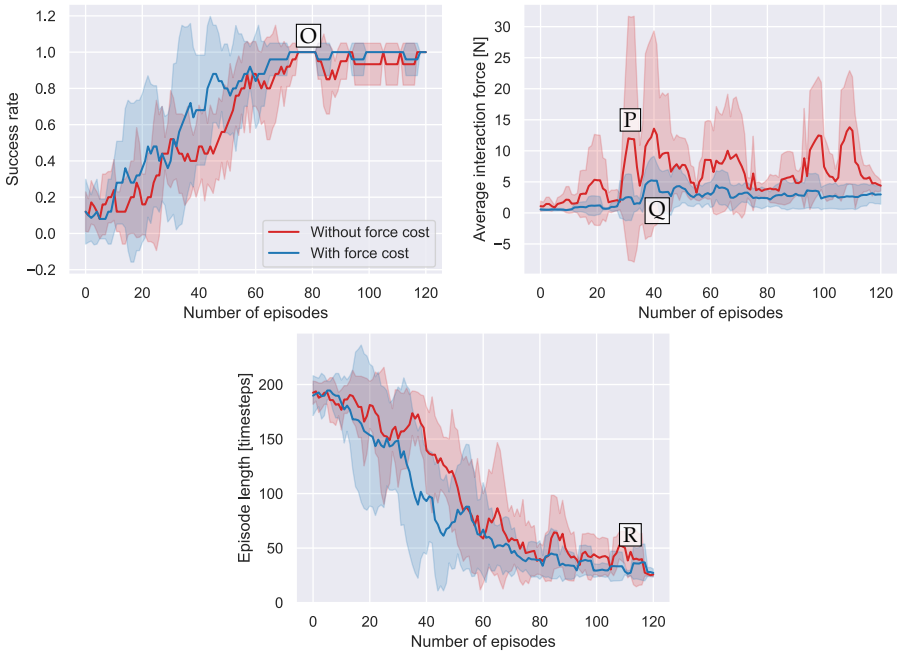


Figure 3.10: Learning performance of the SAC policy on the grid-clamp insertion task for RL-SCT-Shaped on the real robot. The plots show learning performance with and without force interaction cost in the reward function. Each point shows the mean and one standard deviation over 5 learning sessions.

success rate [O] and the speed of task completion in terms of time steps [R] when the policy is learned. Figure 3.11 shows the comparison of the rewards gained over number of episodes. For both cases, learning in terms of reward converges in ≈70 episodes.

Discussion

RL-SCT can also effectively learn a task involving contact forces directly on the real robot as demonstrated by learning the grid-clamp insertion. Notably, through the definition of a transition-distance-based primary reward, the reward curves (both with and without force cost) converge quickly (Figure 3.11). The secondary reward penalizing interaction forces then allows the learning of a policy that uses an optimal force profile.

It is worth emphasizing that both tasks were learned not only in a small amount of time but also with safety for both the robot and the environment, with no collisions observed. This was ensured by the SCT constraints in task space. Moreover, we highlight that, despite achieving 100% success rate after 70 episodes, in the setting without force cost the robot keeps exploring, resulting in subsequent failures (Figure 3.10-left). We observed that continued exploration with inadequate force behaviors often leads the robot to apply too high contact forces on the environment, increasing the likelihood of

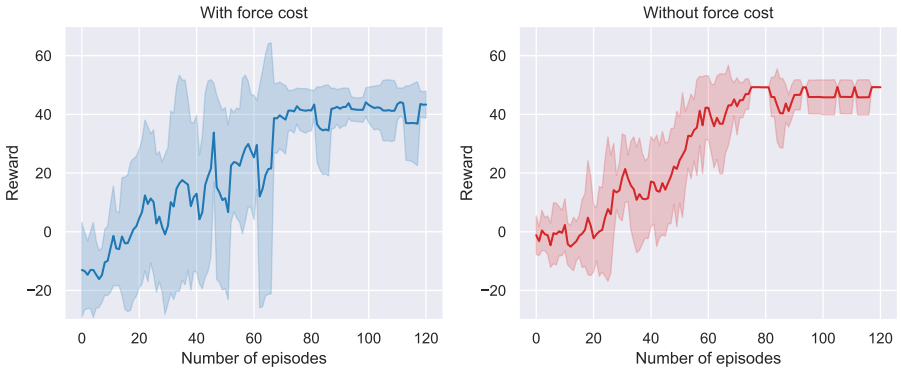


Figure 3.11: Comparison of rewards achieved by learning agent with reward functions with and without interaction force cost. Each point shows the mean and one standard deviation over 5 learning sessions.

failure even after the task has been learned. Such undesired force profiles can be seen in Figure 3.10-center after the 80–90 episode range.

3.5 CONCLUSION

We proposed a framework – RL-SCT – to guide reinforcement learning with constraints that are represented as Shared Control Templates. We have demonstrated that the properties that users expect from shared control – empowerment through freedom of movement, safety by enforcing constraints, low-dimensional input commands to facilitate control – are properties that are also advantageous for robot reinforcement learning.

Our experiments show that the explicit representation of constraints leads to faster learning, and without the need to design complicated reward functions to represent these constraints. Particularly, we demonstrated that RL-SCT facilitates reinforcement learning on real robots. In a pouring task (without contacts between robot and environment) we showed that RL-SCT allows the robot to learn the task in 16 minutes without dangerous interactions with the environment. Given the importance of safety during in-contact tasks, we also applied our approach to a grid-clamp insertion task in the presence of position uncertainties to learn a policy which succeeds at the task while minimizing contact forces. Similarly to the pouring task, the robot was able to quickly learn the task in $\approx$17 minutes, exhibiting low contact forces when compared to a baseline which did not account for contact force minimization. In view of the difficulty to accurately model contacts in simulation (and the subsequent reality gap) our results gain special relevance as we show that RL-SCT can be used to learn directly on the robot safely and efficiently, while minimizing interaction forces.

Despite the successful results obtained, some limitations of RL-SCT should be highlighted. On the one hand, our approach is tailored

to the learning of tasks involving the use of objects with known constraints. It is less suited for learning intricate motor skills, such as those required for juggling, ball-in-cup, or locomotion. On the other hand, the design of SCTs can be cumbersome, especially for new tasks. However, we argue that designing SCTs leads to safer and faster learning than the classical approach of carefully designing shaped rewards, which also often takes a significant amount of time (and trial and error, with all the potentially dangerous interactions it entails). At the same time, we believe that re-using SCTs from already existing tasks in new ones is a promising way to mitigate the design effort.

In future work we will investigate methods to extract the required SCT components from demonstrations [114, 115], namely constraints and nominal solutions to complete a given task. Having an initial policy that can be extracted from demonstrations and which only fails in specific conditions can help further simplify the reward functions used by RL-SCT. Motion primitive representations which capture aleatoric and epistemic uncertainties [116, 117] are promising approaches to build on, to achieve such goals.

4

TOWARDS SAFE AND EFFICIENT LEARNING IN THE WILD: GUIDING RL WITH CONSTRAINED UNCERTAINTY-AWARE MOVEMENT PRIMITIVES

In this chapter, we present a novel RL framework, *Kernelized Guided Reinforcement Learning (KGRL)*, which utilizes human demonstrations to extract task knowledge for guiding RL, while enabling the imposition of linear inequality constraints on the robot state for safety. To facilitate the extraction of such knowledge from demonstrations, we introduce a novel movement primitive representation termed Linearly Constrained Null-Space Kernelized Movement Primitives (LC-NS-KMP). This representation allows learning a nominal policy from demonstrations which respects linear inequality constraints on the robot state. Furthermore, LC-NS-KMP provides a soft null-space projector that enables actions generated by an RL policy to modulate the mean behavior of the nominal policy in accordance with the variance and correlations observed in the demonstrations and imposed safety constraints. This property allows for the design of uncertainty-aware, state-dependent exploration noise, which is essential for efficient learning in complex, contact-rich tasks.

Collectively, these components enable safe and sample-efficient reinforcement learning while simplifying reward function design through the use of partial task completion strategy extracted from human demonstrations. We validate the proposed framework on an insertion task using a torque-controlled 7-DoF robotic manipulator, demonstrating its effectiveness in achieving safe and efficient real-world learning.

KGRL achieves the aim of using human demonstrations as a source of information about the demonstrated task, extracting task completion strategies, implicit constraints, and uncertainty metrics from demonstrations that can be exploited to guide RL and inform exploration strategies.

The remainder of this chapter has been published as [34] Abhishek Padalkar et al. "Towards safe and efficient learning in the wild: guiding RL with constrained uncertainty-aware movement primitives." In: *IEEE Robotics and Automation Letters* (2025).

4.1 INTRODUCTION

Learning from Demonstration (LfD) [13] has proven to be an effective method for motion generation, enabling a robot to imitate and adapt demonstrated motions. Various LfD frameworks have been developed,

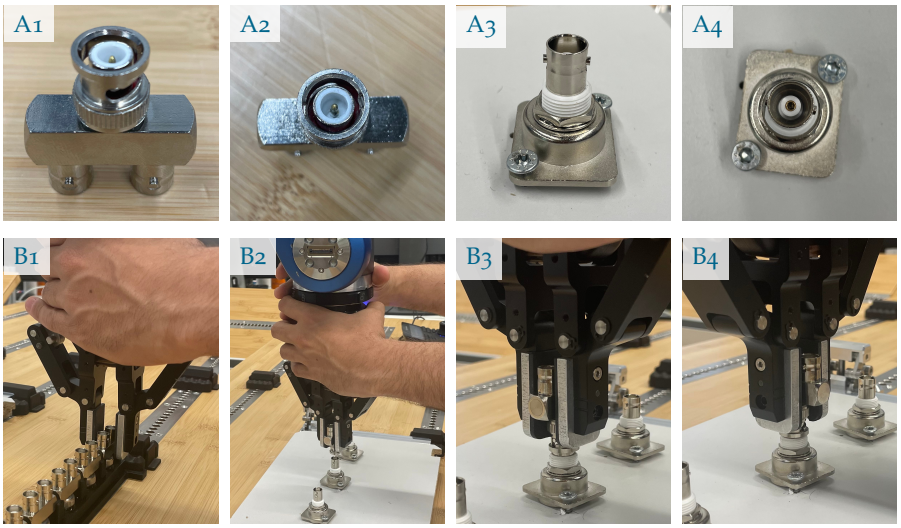


Figure 4.1: BNC connector assembly task from the NIST assembly benchmark [35]. A1 to A4 show the male and female BNC connectors. B1 to B4 show a human demonstrating the task by hand-guiding the DLR SARA robot [2]. An LfD trajectory learned from the demonstrations was not able to solve the task as it does not model the contact dynamics and uncertainties in the kinematics.

including Dynamic Movement Primitives (DMPs) [14], Probabilistic Movement Primitives (ProMPs) [15], and Kernelized Movement Primitives (KMPs) [16], which effectively address common real-world challenges such as generalizing to new situations and avoiding obstacles. However, these methods often struggle in dynamic environments where demonstrations inadequately represent task dynamics, particularly during in-contact tasks. In such tasks, the policy often receives out-of-distribution states as an input, resulting in failures. Collaborative robots aim to mitigate these challenges by employing impedance control to remain compliant while in contact, thus reacting to the inaccuracies caused by kinematics and dynamics. However, learning a robust LfD policy that can adapt to such uncertainties remains a significant challenge.

Reinforcement Learning (RL) addresses this challenge by training a reactive policy that considers the current state of both the robot and its environment. However, the necessity of a large number of trials, coupled with concerns about robot safety, presents a significant challenge for RL to be directly applicable on real robots. Transfer learning attempts to overcome this by learning a policy in a simulation and then applying it to the robot, yet it is still limited by the sim-to-real gap [21]. Learning skills directly on real robots eliminates the need for meticulously modelling tasks in simulation. In [33] we propose a guided RL approach enabling tasks to be learned directly on the real robot, where available task knowledge, represented as constraints, facilitates effective policy search (see Section 4.2 for an overview of

related work). Despite the promising results, the manual modeling of inductive biases (e.g. constraints, exploration strategies) can be challenging, particularly in complex tasks that involve contacts.

To address the above-mentioned challenges, we propose to learn a nominal policy together with a state-dependent exploration strategy from human demonstrations. Specifically, we introduce a novel movement primitive representation, *Linearly Constrained Null-Space Kernelized Movement Primitives* (LC-NS-KMP), where a non-parametric LfD framework generates motions while adhering to linear inequality constraints on the robot state. Simultaneously, LC-NS-KMP provides a *null-space* projector that allows actions generated by RL policies to modify the mean behavior of the LfD policy. The derived projector modifies the mean behavior of the LfD policy in accordance with the variance and correlations in the demonstrations. Consequently, the same null-space action will result in larger/smaller deviations in states where the variance in the demonstrations is higher/lower. This uncertainty-awareness facilitates the design of state-dependent exploration noise. We leverage this property to enable state-based exploration in RL while ensuring safety by enforcing state space constraints. The main contribution of the work presented in this chapter is thus a new RL framework, *Kernelized Guided Reinforcement Learning (KGRL)*, described in Section 4.3, which leverages the properties of LC-NS-KMP to facilitate RL in the wild.

To fully demonstrate the capabilities of KGRL, we selected the BNC connector assembly task from the NIST assembly task board 1 [35], illustrated in Figure 4.1. This task presents significant challenges, as it requires precise insertion of the connector while maintaining compliance to prevent damage to the components. Following the insertion, a complex series of translations and rotations are necessary to lock the connector in position. Our method is well-suited for such tasks because it 1) allows for the specification of constraints that ensure safe operation during state space exploration, and 2) guarantees uncertainty-aware, state-dependent exploration for reinforcement learning, which helps avoid unnecessary exploration in the low-variance regions of the motion (Section 4.4).

4.2 RELATED WORK

Learning from Demonstration is a widely used approach for learning robot motions from humans. Ravichandar *et al.* [13] present a comprehensive survey on recent advances in LfD. One of the main paradigms in LfD is behavior cloning (BC) which uses supervised learning frameworks such as Dynamic Motion Primitives (DMPs) [14], Probabilistic Movement Primitives (ProMPs) [15] and Kernelized Movement Primitives (KMPs) [16] to teach new skills to robots with only a few demonstrations. BC often generates brittle policies that

fail when the robot encounters situations outside the distribution of the demonstrations [42]. Another important paradigm is Inverse Reinforcement Learning (IRL) where a reward function is extracted from human demonstrations to guide RL. Extracting reward functions in IRL is also subject to the coverage of optimal behaviors in the used demonstrations [43, 44].

The limitations of LfD, particularly in tasks involving contacts, can be mitigated by the use of RL in combination with demonstrations, to obtain more robust policies [118] than with LfD alone. This has been done in four main ways: (1) by populating the replay buffer of off-policy RL algorithms with demonstrations and associated reward [119, 120], (2) by extracting a reward function from demonstrations and augmenting it with task specific rewards for further generalization while simultaneously learning an RL policy (GAIL [43], AIRL [121]), (3) by using a behavior cloning policy to regularize the RL policy during learning [122, 123], and (4) by learning a residual RL policy to support an LfD one [106, 124]. Populating the replay buffer with demonstrations provides initial experience from the demonstrations, but the RL policy is initialized randomly and trained from scratch which is an inefficient use of the demonstrations. Learning a reward function and an RL policy simultaneously from the demonstrations can result in unstable learning [42].

Work presented in [125] leverages trajectories produced by a trajectory optimization-based controller to initialize an RL policy and then learns robust behaviors with RL in simulation using domain randomization. They propose to learn adaptive task phase dynamics to facilitate learning policies which are robust against failures. Work presented in [106] proposes multiple strategies for correcting a DMP policy with RL residual policy for solving contact-rich tasks. None of the above methods extract exploration strategies from demonstrations or account for state-dependent noise, instead often assuming isotropic exploration noise. Moreover, in these approaches, the exploration is unconstrained, hence potentially unsafe exploratory actions can cause damage to the environment as well as the robot. Both of these shortcomings are addressed in our proposed solution.

A holistic review of works on safe reinforcement learning is presented in [32]. Approaches such as [126] and [127] present an optimization layer to optimize the actions generated by the RL policy based on safety constraints. Others, e.g. [128, 129] use Gaussian Processes for conducting safe exploration in the proximity of already safe states. Our approach has similarities with these works as we propose to introduce modifications in the trajectories based on variance in the already safe demonstrations. Additionally, we provide a mechanism to enforce hard constraints on the state of the robot.

More recently, Chi *et al.* [130] introduced diffusion policies for LfD which leverage a conditional denoising diffusion process for

generating robot motions. An overview of diffusion policies for RL is provided in [131]. Zheng et al. introduce diffusion-based, safe RL [132] by defining a feasibility-dependent objective, which depends on an offline dataset to predefine a safe region. To the best of our knowledge, current diffusion-based methods in robot manipulation lack inherent constraint enforcement mechanisms and uncertainty-aware exploration.

Kernelized Movement Primitives (KMPs) are an LfD framework originally formulated by Huang *et al.* [16]. Silvério and Huang [50] extended this framework such that modulations in the trajectories are possible with a *null-space* modifier, without re-parameterizing the whole KMP. Later, Huang and Caldwell [52] introduced a way of enforcing linear constraints on the predicted trajectories without re-parameterization. In our work, we build on top of the above-mentioned works by deriving a unified framework for enforcing linear inequality constraints and modifying trajectories with a null-space action. We leverage the resulting framework in combination with RL, to efficiently and safely learn in-contact tasks bootstrapped by human demonstrations.

4.3 METHODOLOGY

4.3.1 *Background*

Recalling from Section 2.2.7, Kernelized Movement Primitives (KMP) learn probabilistic trajectories from demonstrations D, defined in Section 2.2.2, by minimizing the KL divergence between the reference trajectory distribution $T_r = \{\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\}_{n=1}^{N}$, obtained via GMR and GMM in Equation (2.12), and the probabilistic representation of demonstrations defined in Equation (2.14). Huang and Caldwell [52], and Silvério and Huang [50] further enhance KMPs by introducing variants, namely the Linearly Constrained KMP (LC-KMP) and Null Space KMP (NS-KMP), respectively.

The remainder of this subsection provides an overview of LC-KMP and NS-KMP. In the discussions of this section (Section 4.3), we reuse the notations and mathematical foundation of KMP presented in Section 2.2.7. For further details, please refer to Section 2.2.7. Using these variants as building blocks, we derive our proposed LC-NS-KMP in Section 4.3.2 by presenting a unified framework that incorporates both linear inequality constraints and null-space modifications.

Linearly Constrained KMP

Huang and Caldwell [52] formulated a linearly constrained imitation learning framework which incorporates linear inequality constraints on the state of the robot, and applied the same method to minimize the KL-divergence between two distributions represented by T_r and

Equation (2.14) as described in Section 2.2.7, to obtain a *constrained* mean minimization subproblem (similar to Equation (2.15)),

$$\begin{aligned} \underset{\boldsymbol{\mu}_w}{\textbf{argmin}} \quad & \sum_{n=1}^{N} \frac{1}{2} (\boldsymbol{\Theta}^\top (\boldsymbol{\xi}_n^{\mathcal{I}}) \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n)^\top \hat{\boldsymbol{\Sigma}}_n^{-1} (\boldsymbol{\Theta}^\top (\boldsymbol{\xi}_n^{\mathcal{I}}) \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n) \\ & + \frac{1}{2} \lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w \\ \textbf{s.t.} \quad & \boldsymbol{g}_{n,f}^\top \boldsymbol{\xi}^{\mathcal{O}} \geq c_{n,f}, \quad \forall f \in \{1, 2, \ldots, F\}, \\ & \forall n \in \{1, 2, \ldots, N\}, \end{aligned} \tag{4.1}$$

where F is the number of constraints imposed on the output, $\boldsymbol{g}_{n,f}$ and $c_{n,f}$ parameterize the constraint hyperplanes, and λ is a regularization factor. The solution of Equation (4.1) leads to the formulation of Linearly Constrained KMP (LC-KMP). The mean prediction of LC-KMP is given by

$$\mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}) = \boldsymbol{k}^*(\boldsymbol{K} + \lambda \boldsymbol{\Sigma})^{-1} \boldsymbol{\mu} + \boldsymbol{k}^*(\boldsymbol{K} + \lambda \boldsymbol{\Sigma})^{-1} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \boldsymbol{\alpha}, \tag{4.2}$$

where,

$$\begin{aligned} \boldsymbol{\mu} &= [\hat{\boldsymbol{\mu}}_1^\top, \hat{\boldsymbol{\mu}}_2^\top, \ldots, \hat{\boldsymbol{\mu}}_N^\top]^\top, \\ \boldsymbol{\Sigma} &= \text{blockdiag}(\hat{\boldsymbol{\Sigma}}_1, \hat{\boldsymbol{\Sigma}}_2, \ldots, \hat{\boldsymbol{\Sigma}}_N), \\ \boldsymbol{G}_n &= [\boldsymbol{g}_{n,1}\ \boldsymbol{g}_{n,2}\ \boldsymbol{g}_{n,3}\ \ldots\ \boldsymbol{g}_{n,F}], \forall n \in \{1, 2, 3, \ldots, N\}, \\ \bar{\boldsymbol{G}} &= \text{blockdiag}(\boldsymbol{G}_1,\ \boldsymbol{G}_2,\ \boldsymbol{G}_3,\ \ldots,\ \boldsymbol{G}_N), \\ \boldsymbol{\alpha} &= [\alpha_{1,1},\ \alpha_{1,2},\ \ldots,\ \alpha_{1,F},\ \ldots,\ \alpha_{N,1},\ \ldots,\ \alpha_{N,F}], \end{aligned}$$

$\boldsymbol{k}^*$ and $\boldsymbol{K}$ are kernel matrices obtained after applying kernel treatment to the basis functions. The Lagrange multiplier vector $\boldsymbol{\alpha}$ is obtained by solving a convex optimization problem [52]. Note that the prediction given by Equation (4.2) respects the constraints defined in Equation (4.1).

Null Space KMP

Silvério and Huang [50] formulated Null Space KMP which allows modification in the learned trajectory in a variance informed manner by introducing an additional cost term $\frac{1}{2}\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)^\top(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)$ in the original mean minimization subproblem defined in Equation (2.15) which led to the formulation of KMP, to obtain a new mean minimization subproblem

$$\begin{aligned} \underset{\boldsymbol{\mu}_w}{\textbf{argmin}} \quad & \sum_{n=1}^{N} \frac{1}{2} (\boldsymbol{\Theta}^\top (\boldsymbol{\xi}_n^{\mathcal{I}}) \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n)^\top \hat{\boldsymbol{\Sigma}}_n^{-1} (\boldsymbol{\Theta}^\top (\boldsymbol{\xi}_n^{\mathcal{I}}) \boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n) \\ & + \frac{1}{2} \lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w + \frac{1}{2} \beta (\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)^\top (\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w). \end{aligned} \tag{4.3}$$

The solution of the above optimization problem leads to the formulation of Null Space KMP (NS-KMP). The mean prediction of NS-KMP is given by

$$\mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}) = \boldsymbol{k}^*(\boldsymbol{K} + \hbar \boldsymbol{\Sigma})^{-1} \boldsymbol{\mu} + \frac{\beta}{\hbar} \left(\hat{\boldsymbol{k}}^* - \boldsymbol{k}^*(\boldsymbol{K} + \hbar \boldsymbol{\Sigma})^{-1} \hat{\boldsymbol{K}} \right) \underline{\boldsymbol{K}}^{-1} \hat{\boldsymbol{\xi}} \tag{4.4}$$

where $\hat{\boldsymbol{k}}^*$, $\hat{\boldsymbol{K}}$, and $\underline{\boldsymbol{K}}^{-1}$ are kernel matrices obtained after applying kernel treatment to the basis functions, $\hbar = \lambda + \beta$, and $\hat{\boldsymbol{\xi}}$ is the null-space action.

4.3.2 *LC-NS-KMP Formulation*

In this chapter, we derive a unified method which combines null-space modifier for KMPs proposed by [50] and linear inequality constraints proposed by [52]. Combining the desirable properties of these methods, our framework allows RL to modulate a mean trajectory predicted by KMPs adhering to linear inequality constraints and covariance in the demonstrations. It helps RL conduct an effective search by modulating the exploration noise in accordance with the variance and constraints.

We start from the same constrained mean optimization problem defined by Equation (4.1), and introduce an additional cost term $\frac{1}{2}\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)^\top(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)$ which results in a *soft* null-space projector that modifies the mean trajectory (see [50] for details), to obtain,

$$\begin{aligned}
\underset{\boldsymbol{\mu}_w}{\textbf{argmin}} \quad & \sum_{n=1}^{N} \frac{1}{2}(\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n)^\top \hat{\boldsymbol{\Sigma}}^{-1}(\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n) \\
& + \frac{1}{2}\lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w + \frac{1}{2}\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)^\top(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w), \qquad (4.5) \\
\textbf{s.t.} \quad & \boldsymbol{g}_{n,f}^\top \boldsymbol{\xi}^{\mathcal{O}} \geq c_{n,f}, \quad \forall f \in \{1, 2, \ldots, F\}, \\
& \forall n \in \{1, 2, \ldots, N\}.
\end{aligned}$$

Here, the cost term $\frac{1}{2}\lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w$ regularizes the solution and the cost term $\frac{1}{2}\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)^\top(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)$ inspired from [50] keeps the solution close to a desired one $\hat{\boldsymbol{\mu}}_w$.

Similarly to [52], we propose to solve Equation (4.5) by introducing Lagrange multipliers $\alpha_{n,f} \geq 0$, with the Lagrange function

$$\begin{aligned}
L(\boldsymbol{\mu}_w, \alpha) = & \sum_{n=1}^{N} \frac{1}{2}(\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n)^\top \hat{\boldsymbol{\Sigma}}_n^{-1}(\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n) \\
& + \frac{1}{2}\lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w + \frac{1}{2}\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)^\top(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w) \qquad (4.6) \\
& - \sum_{n=1}^{N} \sum_{f=1}^{F} \alpha_{n,f}(\mathbf{g}_{n,f}^\top \boldsymbol{\Theta}(\boldsymbol{\xi}_n^{\mathcal{I}})^\top \boldsymbol{\mu}_w - c_{n,f}),
\end{aligned}$$

which can be re-written using matrix notation as

$$\begin{aligned}
L(\boldsymbol{\mu}_w, \boldsymbol{\alpha}) = & \frac{1}{2}(\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\mu}) \\
& + \frac{1}{2}\lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w + \frac{1}{2}\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)^\top(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w) \qquad (4.7) \\
& - \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\alpha}^\top \bar{\boldsymbol{C}},
\end{aligned}$$

where

$$\begin{aligned}
&\boldsymbol{\Phi} = [\boldsymbol{\Theta}(\boldsymbol{\xi}_1^{\mathcal{I}})\ \boldsymbol{\Theta}(\boldsymbol{\xi}_2^{\mathcal{I}})\ \dots\ \boldsymbol{\Theta}(\boldsymbol{\xi}_N^{\mathcal{I}})],\\
&\bar{\boldsymbol{C}} = [\boldsymbol{C}_1^\top\ \boldsymbol{C}_2^\top\ \dots\ \boldsymbol{C}_N^\top]^\top,\\
&\boldsymbol{C}_n = [c_{n,1}\ c_{n,2}\ \dots\ c_{n,F}]^\top, \forall n \in \{1,2,\dots,N\}.
\end{aligned}$$

The objective function is then minimized with respect to $\boldsymbol{\mu}_w$, by calculating the derivative with respect to $\boldsymbol{\mu}_w$ as

$$\begin{aligned}
\frac{\partial \mathcal{L}(\boldsymbol{\mu}_w, \boldsymbol{\alpha})}{\partial \boldsymbol{\mu}_w} &= \frac{1}{2}(2\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w - \boldsymbol{\mu})) + \frac{1}{2}(2\lambda\boldsymbol{\mu}_w)\\
&\quad + \frac{1}{2}(2\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)) - \boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top, \qquad (4.8)\\
&= \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w - \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + (\lambda+\beta)\boldsymbol{\mu}_w\\
&\quad - \beta\hat{\boldsymbol{\mu}}_w - \boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top,\\
&= \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w - \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \hbar\boldsymbol{\mu}_w - \beta\hat{\boldsymbol{\mu}}_w - \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha},\\
&= (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top + \hbar\boldsymbol{I})\boldsymbol{\mu}_w - (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \beta\hat{\boldsymbol{\mu}}_w + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}). \qquad (4.9)
\end{aligned}$$

By setting the derivative $\frac{\partial L(\boldsymbol{\mu}_w, \boldsymbol{\alpha})}{\partial \boldsymbol{\mu}_w} = 0$,

$$(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top + \hbar\boldsymbol{I})\boldsymbol{\mu}_w^* = (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \beta\hat{\boldsymbol{\mu}}_w + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) \qquad (4.10)$$

resulting in

$$\begin{aligned}
\boldsymbol{\mu}_w^* &= (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top + \hbar\boldsymbol{I})^{-1}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \beta\hat{\boldsymbol{\mu}}_w + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}), \qquad (4.11)\\
&= \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w, \qquad (4.12)
\end{aligned}$$

where, $\boldsymbol{A} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \hbar\boldsymbol{\Sigma})^{-1}$ and $\hbar = \lambda + \beta$. Equation (4.11) is further simplified into Equation (4.12) using the Woodbury identity[1].

By substituting $\boldsymbol{\mu}_w^*$ in Equation (4.7) and Equation (2.13), we get

$$\begin{aligned}
\tilde{L}(\boldsymbol{\alpha}) &= \boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Sigma}\boldsymbol{A}\mathcal{K}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + (2\boldsymbol{\mu}^\top\boldsymbol{A}\mathcal{K}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\\
&\quad - \beta\hat{\boldsymbol{\mu}}_w^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}} + \bar{\boldsymbol{C}}^\top)\boldsymbol{\alpha} + \text{const}, \qquad (4.13)
\end{aligned}$$

and

$$\begin{aligned}
\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}}) &= \boldsymbol{\Theta}(\boldsymbol{\xi}_*^{\mathcal{I}})\boldsymbol{\mu}_w^*\\
&= \boldsymbol{\Theta}(\boldsymbol{\xi}_*^{\mathcal{I}})\Big(\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w\Big), \qquad (4.14)
\end{aligned}$$

respectively, where $\boldsymbol{\xi}_*^{\mathcal{O}} = f(\boldsymbol{\xi}_*^{\mathcal{I}})$, and $\mathcal{K} = -\frac{1}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \frac{\hbar}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$.

For a desired output $\hat{\boldsymbol{\xi}} = \hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\mu}}_w$, we can estimate the optimal weight vector $\hat{\boldsymbol{\mu}}_w$ given the target trajectory $\hat{\boldsymbol{\xi}}$, using the right pseudo-inverse of $\hat{\boldsymbol{\Phi}}^\top$, similarly to [50], hence,

$$\hat{\boldsymbol{\mu}}_w = \hat{\boldsymbol{\Phi}}(\hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\Phi}})^{-1}\hat{\boldsymbol{\xi}}.$$

1 Woodbury identity: if $\boldsymbol{P} \succ 0$ and $\boldsymbol{R} \succ 0$, $(\boldsymbol{P}^{-1} + \boldsymbol{B}^\top\boldsymbol{R}^{-1}\boldsymbol{B})^{-1}\boldsymbol{B}^\top\boldsymbol{R}^{-1} = \boldsymbol{P}\boldsymbol{B}^\top(\boldsymbol{B}\boldsymbol{P}\boldsymbol{B}^\top + \boldsymbol{R})^{-1}$.

By further replacing $\hat{\boldsymbol{\mu}}_w$ in Equation (4.13) and Equation (4.14), we obtain, respectively,

$$\begin{aligned}\tilde{L}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Sigma} \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \boldsymbol{\alpha} + \Big(2\boldsymbol{\mu}^\top \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \\ - \beta \hat{\boldsymbol{\xi}}^\top (\hat{\boldsymbol{\Phi}}^\top \hat{\boldsymbol{\Phi}})^{-1} \hat{\boldsymbol{\Phi}}^\top \boldsymbol{\Phi} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} + \bar{\boldsymbol{C}}^\top \Big) \boldsymbol{\alpha} + \text{const},\end{aligned} \tag{4.15}$$

and

$$\begin{aligned}\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}}) = \boldsymbol{\Theta}(\boldsymbol{\xi}_*^{\mathcal{I}}) \Big(\boldsymbol{\Phi} \boldsymbol{A} \boldsymbol{\mu} + \boldsymbol{\Phi} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \boldsymbol{\alpha} \\ + \frac{\beta}{\hbar} (\boldsymbol{I} - \boldsymbol{\Phi} \boldsymbol{A} \boldsymbol{\Phi}^\top) \hat{\boldsymbol{\Phi}} (\hat{\boldsymbol{\Phi}}^\top \hat{\boldsymbol{\Phi}})^{-1} \hat{\boldsymbol{\xi}} \Big).\end{aligned} \tag{4.16}$$

Similarly to [16], we propose to kernelize the above equation using the kernel treatment, i.e. the inner product of basis functions $\varphi(\boldsymbol{\xi}_i^{\mathcal{I}})$ and $\varphi(\boldsymbol{\xi}_j^{\mathcal{I}})$ defined as

$$\varphi(\boldsymbol{\xi}_i^{\mathcal{I}})^\top \varphi(\boldsymbol{\xi}_j^{\mathcal{I}}) = k(\boldsymbol{\xi}_i^{\mathcal{I}}, \boldsymbol{\xi}_j^{\mathcal{I}}),$$

where $k(.,.)$ is a kernel function. With the kernel treatment, we can write

$$\begin{aligned}\tilde{L}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Sigma} \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \boldsymbol{\alpha} + \Big(2\boldsymbol{\mu}^\top \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \\ - \beta \hat{\boldsymbol{\xi}}^\top \underline{\boldsymbol{K}}^{-1} \hat{\boldsymbol{K}} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} + \bar{\boldsymbol{C}}^\top \Big) \boldsymbol{\alpha} + \text{const},\end{aligned} \tag{4.17}$$

and

$$\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}}) = \boldsymbol{k}^* \boldsymbol{A} \boldsymbol{\mu} + \boldsymbol{k}^* \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \boldsymbol{\alpha} + \frac{\beta}{\hbar} (\hat{\boldsymbol{k}}^* - \boldsymbol{k}^* \boldsymbol{A} \hat{\boldsymbol{K}}) \underline{\boldsymbol{K}}^{-1} \hat{\boldsymbol{\xi}}, \tag{4.18}$$

with $\boldsymbol{A} = (\boldsymbol{K} + \hbar \boldsymbol{\Sigma})^{-1}$, and $\mathcal{K} = -\frac{1}{2} \boldsymbol{K} \boldsymbol{\Sigma}^{-1} \boldsymbol{K} - \frac{\hbar}{2} \boldsymbol{K}$, where,

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & k(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \dots & k(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \\ k(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & k(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \dots & k(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & k(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \dots & k(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \end{bmatrix},$$

$$\begin{aligned} &k(\boldsymbol{\xi}_i^{\mathcal{I}}, \boldsymbol{\xi}_j^{\mathcal{I}}) = k(\boldsymbol{\xi}_i^{\mathcal{I}}, \boldsymbol{\xi}_j^{\mathcal{I}}) \boldsymbol{I}, \\ &\boldsymbol{k}^* = [k(\boldsymbol{\xi}_*^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}), \ \dots, \ k(\boldsymbol{\xi}_*^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}})], \\ &\underline{\boldsymbol{K}} = \hat{\boldsymbol{\Phi}}^\top \hat{\boldsymbol{\Phi}}, \\ &\hat{\boldsymbol{K}} = \boldsymbol{\Phi}^\top \hat{\boldsymbol{\Phi}}, \\ &\hat{\boldsymbol{k}}^* = \boldsymbol{\Phi}(\boldsymbol{\xi}_*^{\mathcal{I}})^\top \hat{\boldsymbol{\Phi}}. \end{aligned}$$

We substitute

$$\begin{aligned} &\mathcal{B}_1 = \bar{\boldsymbol{G}}^\top \boldsymbol{\Sigma} \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}}, \\ &\mathcal{B}_2 = 2\boldsymbol{\mu}^\top \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} + \beta \hat{\boldsymbol{\xi}}^\top \underline{\boldsymbol{K}}^{-1} \hat{\boldsymbol{K}} (-\boldsymbol{A}) \boldsymbol{\Sigma} \bar{\boldsymbol{G}} + \bar{\boldsymbol{C}}^\top \end{aligned} \tag{4.19}$$

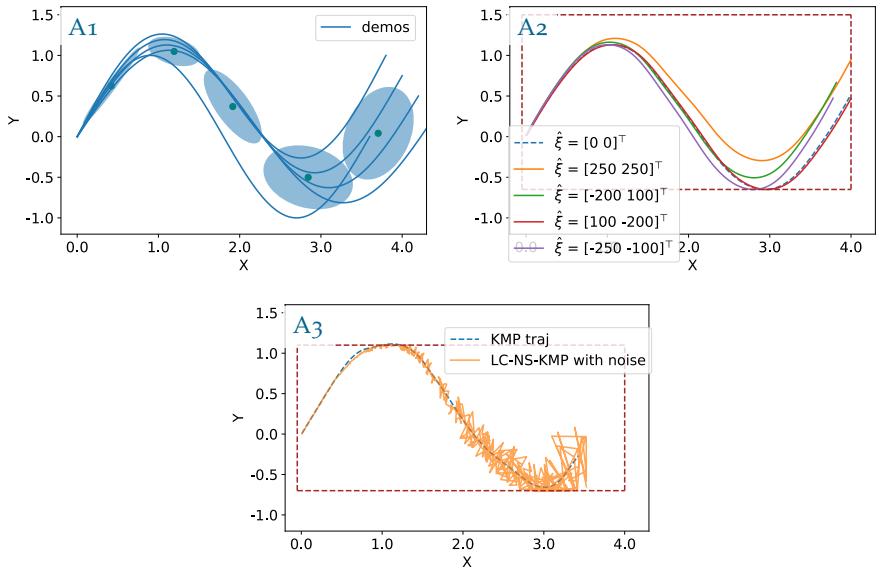


Figure 4.2: LC-NS-KMP properties: (A1) shows the demonstrations and the learned GMM; (A2) shows the modulations due to different $\hat{\boldsymbol{\xi}}$ applied at $t = 3.2s$, adhering to the constraints; (A3) shows the effect of randomly sampled $\hat{\boldsymbol{\xi}}$. In (A2) and (A3), the modifications introduced by $\hat{\boldsymbol{\xi}}$ respect the linear inequality constraints, which are shown by dashed red rectangle.

in Equation (4.17), which results in a quadratic function

$$\tilde{L}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \mathcal{B}_1 \boldsymbol{\alpha} + \mathcal{B}_2 \boldsymbol{\alpha}.$$

Thus, we can tackle the problem of finding optimal Lagrange multipliers $\boldsymbol{\alpha}$ by solving

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\textbf{argmax}} \quad & \boldsymbol{\alpha}^\top \mathcal{B}_1 \boldsymbol{\alpha} + \mathcal{B}_2 \boldsymbol{\alpha}, \\ \textbf{s.t.} \quad & \boldsymbol{\alpha} \geq 0. \end{aligned} \tag{4.20}$$

Since $\boldsymbol{A}\mathcal{K}\boldsymbol{A} = (\boldsymbol{A}\mathcal{K}\boldsymbol{A})^\top \preccurlyeq 0$ and $-\boldsymbol{A} = -\boldsymbol{A}^\top \preccurlyeq 0$, Eq. (4.20) defines a quadratic program with linear inequality constraints. After solving for $\boldsymbol{\alpha}$, Eq. (4.18) enables constrained predictions incorporating modulations from $\hat{\boldsymbol{\xi}}$. See Section A.1 for more details on the derivation.

4.3.3 *Properties of LC-NS-KMP*

We evaluated the properties of LC-NS-KMP using synthetically generated 2D time trajectories, shown in Figure 4.2 (A1), alongside the learned GMM. We chose the squared exponential kernel $k(t_i, t_j) = \exp(-l(t_i - t_j)^2)$, with hyperparameter $l = 2$. We use GMR for generating the reference trajectories and covariances, and the same kernel in all our experiments, including those in Section 4.4. The KMP input is $\boldsymbol{\xi}^{\mathcal{I}} = t$ and the 2D outcome is $\boldsymbol{\xi}^{\mathcal{O}} = [x \quad y]^\top$.

Figure 4.2 (A2) illustrates the impact of various null-space actions $\hat{\boldsymbol{\xi}}$ applied at $t = 3.2s$ on the resultant trajectories $\{\boldsymbol{\xi}_i^{\mathcal{O}}\}_{i=1}^{N}$. Despite the local modulation in the trajectory, smoothness is preserved while respecting the linear inequality constraints defined in LC-NS-KMP. Finally, Figure 4.2 (A3) shows trajectories generated using Equation (4.18), where $\hat{\boldsymbol{\xi}}$ is randomly sampled from a normal distribution at each time step. $\hat{\boldsymbol{\xi}}$ modulates the trajectory in accordance with the variance in the demonstrations and hence it demonstrates the uncertainty-aware exploration through null-space actions. The modulated trajectory also satisfies the constraints despite the noise amplitude. This property paves the way for safe exploration in RL.

4.3.4 *Kernelized Guided RL (KGRL)*

We propose to use null-space actions $\hat{\boldsymbol{\xi}}$ obtained from an RL policy $\pi(\hat{\boldsymbol{\xi}}|\boldsymbol{s})$, to introduce modulations in the LfD trajectory learned from the demonstrations. LC-KMP [52] in Equation (4.2) predicts a trajectory which respects the linear inequality constraints defined in Equation (4.1). Our proposed method LC-NS-KMP in Equation (4.18) allows modifications in the prediction using null-space action $\hat{\boldsymbol{\xi}}$, whose magnitude depends on the variance in the demonstrations, while respecting the constraints in Equation (4.5). This important property allows us to conduct efficient and safe RL search using null-space actions. Particularly, we obtain null-space actions from a RL policy $\pi(\hat{\boldsymbol{\xi}}|\boldsymbol{s})$ modifying the prediction for further refinement as

$$\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}}) = \boldsymbol{k}^* \boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{k}^* \boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\beta}{\hbar}(\hat{\boldsymbol{k}}^* - \boldsymbol{k}^* \boldsymbol{A}\hat{\boldsymbol{K}})\hat{\boldsymbol{\xi}}. \tag{4.21}$$

$\underline{\boldsymbol{K}}$ becomes identity when modification is applied to a single point in the trajectory. It should be noted that $\boldsymbol{\alpha}$ is a function of $\hat{\boldsymbol{\xi}}$, as shown in Equation (4.19). To compute the KGRL prediction using Equation (4.21), we first sample $\hat{\boldsymbol{\xi}}$ and then, using this sampled value, compute $\boldsymbol{\alpha}$ via Equation (4.20). Our complete approach, termed Kernelized Guided Reinforcement Learning (KGRL), is summarized in Algorithm 2 for one RL episode.

4.4 EVALUATION

4.4.1 *Experiments in Simulation*

We evaluate the performance of our proposed framework against two baselines: 1) a residual RL policy and 2) a safe residual RL policy using predictive safety filters [32]. Both of these residual policies learn to adapt the mean LfD trajectory. For this evaluation, we developed a simulation involving a robot that navigates a 2D environment with the primary objective of reaching a goal position while passing through a

Algorithm 2 Kernelized Guided Reinforcement Learning (KGRL)

Collect demonstrations $D \leftarrow \{\{\boldsymbol{\xi}^{\mathcal{I}}_{n,m}, \boldsymbol{\xi}^{\mathcal{O}}_{n,m}\}_{n=1}^{N}\}_{m=1}^{M}$
Set λ, β and define $k(.,.)$
Set horizon h
Model joint probability distribution $\mathcal{P}(\boldsymbol{\xi}^{\mathcal{I}}, \boldsymbol{\xi}^{\mathcal{O}})$
Define set of constraints $O = \{\{\boldsymbol{g}^{\top}_{n,f}, c_{n,f}\}_{n=1}^{N}\}_{f=1}^{F}$
Initialize a RL policy $\pi(\hat{\boldsymbol{\xi}}|\boldsymbol{s})$ and policy parameters
loop for each $n = 1, \ldots, N$
    In current state $\boldsymbol{\xi}^{\mathcal{I}}_{n}$, retrieve the reference trajectory distribution $T_r \leftarrow \{\hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\Sigma}}_i\}_{i=n}^{n+h}$
    Choose constraints for T_r from O
    Sample RL action $\hat{\boldsymbol{\xi}}$ from RL policy π
    Compute $\boldsymbol{\alpha}$ with Equation (4.20)
    Compute $\mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}_{*})$ with Equation (4.21)
    Apply $\mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}_{*})$ to robot
    Compute reward
    Update RL policy π
end loop

narrow passage, see Figure 4.3. As illustrated in Figure 4.3, demonstrations $D = \{\{t_{n,m}, \boldsymbol{p}_{n,m}\}_{n=1}^{400}\}_{m=1}^{9}$ were given to the robot, where $\boldsymbol{p} = [x\ y]^{\top}$ is the position of the robot. However, after demonstrating the trajectories, an obstacle is added along the path of the robot so that the mean trajectory consistently intersects with the obstacle. Moreover, we define a *Restricted Zone* ($x \geq 0.6$) for the robot, as shown in Figure 4.3, after the demonstrations were collected. The robot must not go into the Restricted Zone to ensure safety during learning and execution. Consequently, an RL agent must learn to avoid the obstacle and not to enter the restricted zone, while still successfully reaching the goal.

We selected KMP, described in Section 2.2.7, as the baseline LfD method, with $\boldsymbol{\xi}^{\mathcal{I}} = t$ and generated the necessary time-based trajectory $\boldsymbol{p}_t^{\text{kmp}} = \mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}(t))$ to reach the goal. The residual RL policy $\pi_{\text{res}}(\Delta\boldsymbol{p}_t|t)$ modifies the mean trajectory $\boldsymbol{p}_t^{\text{kmp}}$ for avoiding the obstacle. The robot follows the resultant 2D position $\hat{\boldsymbol{p}}_t = \boldsymbol{p}_t^{\text{kmp}} + \Delta\boldsymbol{p}_t$.

We then implemented predictive safety filters [32] in the form of Active Constraints (AC) similar to our earlier work [33] in Chapter 3, which keep the robot in the safe zone and avoid wall collisions by filtering the unsafe $\hat{\boldsymbol{p}}_t$ commands. This resulted in the safe residual RL policy $\pi_{\text{safe}}(\Delta\boldsymbol{p}_t|t)$. As illustrated in Figure 4.4b, ACs stop the robot 0.02 units before the restricted zone and the walls. ACs implement projection functions which project an unsafe robot position back to the safe zone before sending it to the robot for execution.

We then compare these baselines to our KGRL algorithm outlined in Equation (4.21), where an RL policy $\pi(\hat{\boldsymbol{\xi}}|t)$ generates null space

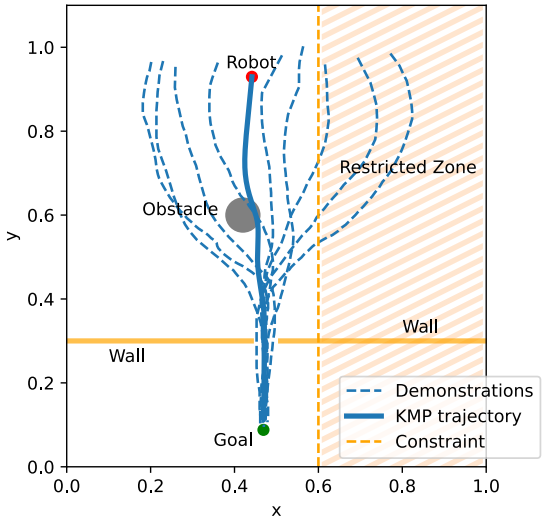


Figure 4.3: Simulated 2D environment where the robot navigates through a narrow passage to reach the goal. A trajectory for navigation can be learned from demonstration. Then an RL policy learns to avoid the obstacle in the path of the robot. The robot must not cross into the restricted zone.

actions $\hat{\boldsymbol{\xi}}$ that modify the trajectory using the null-space projector in LC-NS-KMP. For KGRL, the KMP input is $\boldsymbol{\xi}^{\mathcal{I}} = t$ and outcome is $\boldsymbol{\xi}^{\mathcal{O}} = \boldsymbol{p}_t$. The robot follows the resultant 2D position $\hat{\boldsymbol{p}}_t = \mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}(t))$, derived from Equation (4.21). In KGRL, safety is ensured by defining a boundary constraint on robot state as

$$\boldsymbol{g}_{n,1} = [-1, 0], \quad c_{n,1} = -0.6, \quad \forall n \in \{1, 2, \ldots, N\},$$

so that the robot does not enter into the Restricted Zone.

In all cases, the reward function for the robot is given by

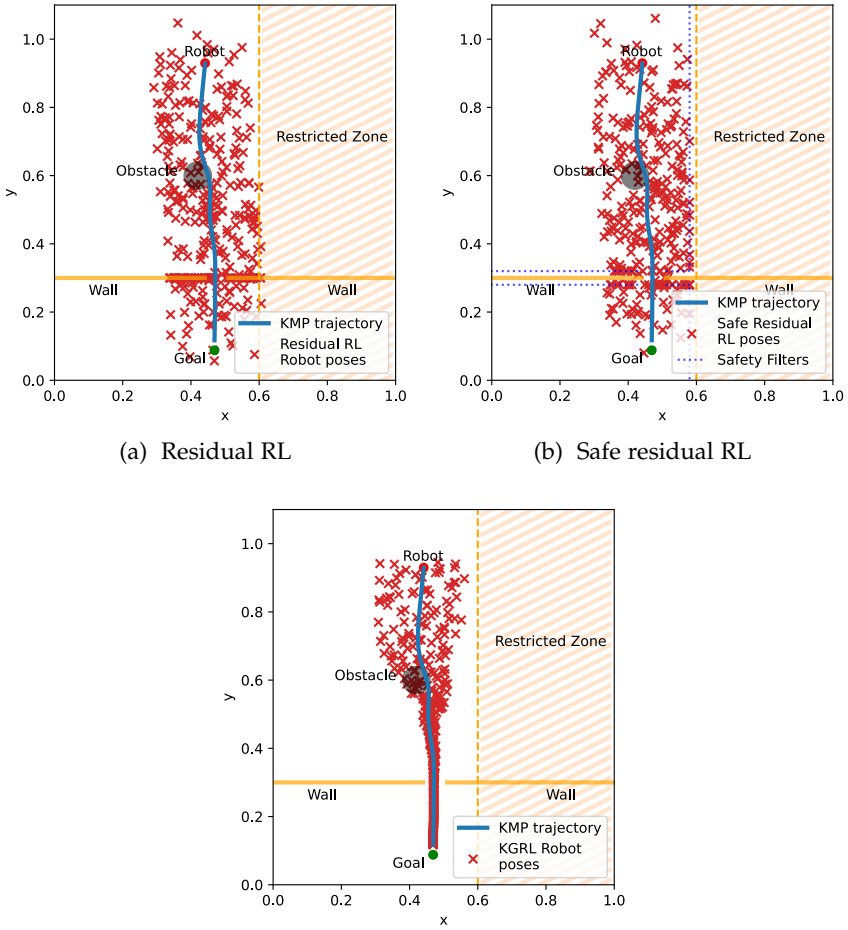
$$r_t = r_a + r_o + r_T, \tag{4.22}$$

$$r_a = -10\delta\boldsymbol{p}_t^\top \delta\boldsymbol{p}_t, \tag{4.23}$$

$$r_o = \begin{cases} -100(0.04 - \boldsymbol{d}_t), & \text{if } \boldsymbol{d}_t \leq 0.04 \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

$$r_T = \begin{cases} 200 & \text{at terminal step } T\text{, if successful} \\ 0 & \text{otherwise} \end{cases} \tag{4.25}$$

where $\delta\boldsymbol{p}_t$ is the displacement of the robot, which can be different from the residual action $\Delta\boldsymbol{p}_t$, e.g. due to collisions with the environment, $\boldsymbol{d}_t$ is the distance of the robot from the obstacle, the terminal reward r_T is given if the episode terminates successfully, r_o is the obstacle avoidance cost, and r_a is the action cost. The episode is considered successful if the robot reaches the goal within 400 time steps. Conversely, it is deemed unsuccessful if the robot becomes blocked in the narrow



(a) Residual RL

(b) Safe residual RL

(c) KGRL (ours)

Figure 4.4: Comparison of the robot poses resulting from isotropic exploration in residual RL approaches, (a)–(b), and uncertainty-aware exploration in KGRL, (c). For comparison purposes, manually-defined safety filters for safe residual RL keep the robot in the safe zone and prevent wall collisions in (b).

passage for 20 or more time steps, or if the maximum limit of 400 time steps is reached without achieving the goal.

In all cases, the RL policy is learned using a DNN with 2 hidden layers with 256 neurons each. To train the RL policies, we used the implementation of Truncated Quantile Critics (TQC) [67] from Stable-baselines3 [111]. Key TQC hyperparameters used in the experiment are summarized in Table 4.1.

The performance comparison between residual RL, safe residual RL and the KGRL framework is shown in Figure 4.5. KGRL quickly achieves the primary objective while minimizing both obstacle avoidance and action costs. In contrast, residual learning approaches take much longer due to isotropic noise (see Figure 4.4a and Figure 4.4b) used for exploration, which often causes the robot to become stuck in the narrow passage.

Hyperparameter	Simulation	Real Robot
Learning rate	0.001	0.001
Soft update coefficient	0.02	0.01
Discount factor	0.99	0.995
Training frequency	8	8
Gradient steps	8	8
Entropy regularization coefficient	auto	auto (initial 0.1)

Table 4.1: TQC hyperparameters used in simulation and on the real robot.

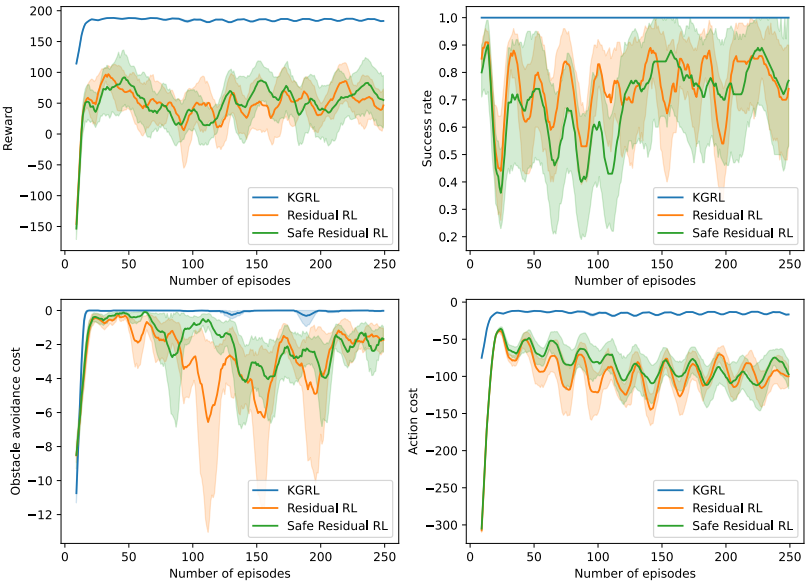


Figure 4.5: Comparison of the performance of residual RL and safe residual RL with LfD to KGRL. KGRL achieves success rate of 1 from the beginning while optimizing secondary costs. Both residual RL approaches show unstable learning behavior as the robot often get stuck at the narrow passage due to inefficient exploration.

Conversely, KGRL modifies trajectories based on the variance in demonstrations, reducing unnecessary exploration in the low-variance region near the narrow passage (see Figure 4.4c). Additionally, hard constraints in KGRL keep the robot within a safe zone, enhancing its overall performance. Interestingly, the walls defining the narrow passage are not modeled as hard constraints. Instead, collisions in that region are avoided through soft constraints on exploration, informed by the low variance in demonstrations where the robot remained at a safe distance from the walls.

Figure 4.6 compares the number of collisions with the wall and the constraint violations in residual RL, safe residual RL and KGRL during learning. Since demonstrations exhibit low variance while passing through the narrow passage, our approach effectively shows

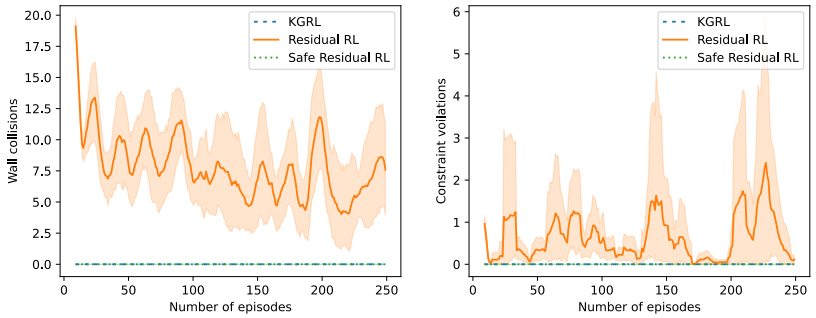


Figure 4.6: Comparison of the number of wall collisions and constraint violations per episode. KGRL avoids wall collisions as the exploration is guided by low-variance in the motion near the narrow passage, while linear constraints prevent any constraint violations. Residual RL shows high number of wall collisions as well as constraint violations. Safety filters used in safe residual RL prevent wall collisions and constraint violations.

no collisions with the wall due to the guided exploration. As seen in Figure 4.4, the magnitude of the exploration actions generated in KGRL is low in this region as it adheres to the demonstrated variance. On the other hand, the isotropic exploration noise in residual RL leads to more collisions with walls. Constraints enforced in KGRL keep the robot out of the restricted zone, effectively limiting the constraint violations to zero. Safe residual RL also shows no constraint violations and collisions with the wall due to safety filters. In this case, the safety filters are needed to be explicitly implemented near the walls, unlike KGRL where collisions with the walls are avoided due to the state-dependent uncertainty-aware exploration. With no constraint enforcement mechanism in residual RL, we observe a high number of constraint violations and wall collisions during learning.

The state of the RL agent, denoted by $\boldsymbol{s}$ in KGRL, can accommodate different input modalities beyond pose and wrench vectors, such as image-based observations of the environment. To demonstrate this capability, we conducted additional experiments in the simulation shown in Figure 4.3, where the RL state $\boldsymbol{s}$ was derived from rendered images of the environment. Specifically, the state at time step t is obtained by flattening the image into a one-dimensional vector as

$$\boldsymbol{s}_t = \mathrm{vec}(\mathrm{Img}_t) \in \{0, 1, \ldots, 255\}^{47520},$$

where $\mathrm{vec}(\cdot)$ denotes the vectorization (flattening) operation concatenating all pixel values across spatial and channel dimensions, and

$$\mathrm{Img}_t \in \{0, 1, \ldots, 255\}^{110 \times 144 \times 3}$$

represents the RGB image observation at time step t. Figure 4.7 illustrates that, even with high-dimensional image inputs, KGRL successfully learns obstacle avoidance within 30 episodes.

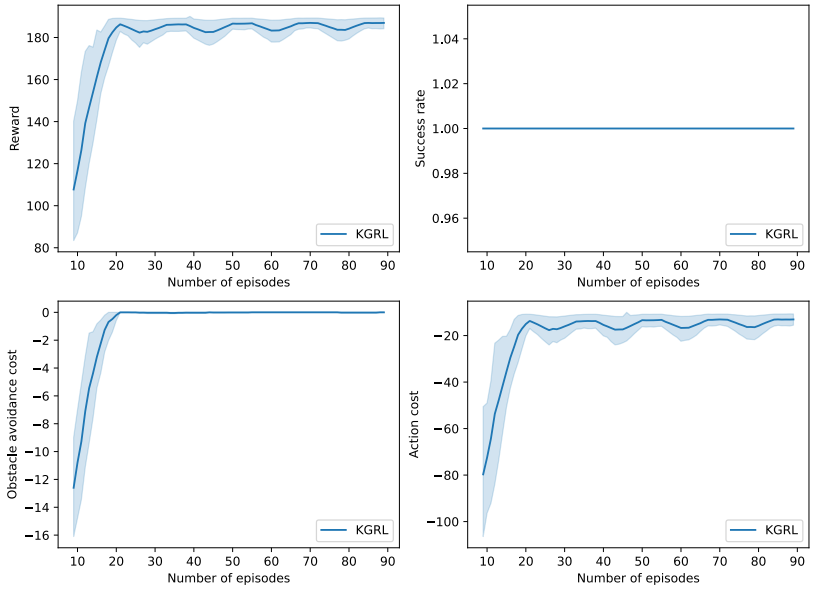


Figure 4.7: Performance of KGRL in learning obstacle avoidance task in simulation with RL state $\boldsymbol{s}$ as an image.

4.4.2 *Experiments on Real Robot*

To evaluate our framework on a real robot, we selected a task from the NIST assembly benchmark 1 [35] involving the plugging of a BNC connector. This task is particularly challenging and requires multiple manipulation strategies for different phases: inserting, aligning, and locking the connector. The strategy learned from demonstration alone is insufficient to complete the task, and relying purely on RL would necessitate an impractically large number of trials. Our approach utilizes state-dependent guided exploration, allowing the robot to explore the state-action space selectively, where necessary. Additionally, linear inequality constraints reduce the state space and ensure the robot's safety. We use the DLR SARA robot in our experiment for learning the task of inserting a BNC connector. Figure 4.1 illustrates the experimental setup. Images (A1) and (A2) present side and top views of the BNC male connector, while (A3) and (A4) show the respective views for the BNC female connector. Images (B1) to (B4) depict the stages of picking, aligning, inserting, and locking the BNC male connector, respectively, while the human is demonstrating the task on the DLR SARA robot.

Demonstrations $D = \{\{t_{n,m}, \boldsymbol{p}_{n,m}\}_{n=1}^{400}\}_{m=1}^{5}$ were provided for the aligning, inserting, and locking phases. The 6D pose of the robot end-effector $\boldsymbol{p} = [x\ y\ z\ \psi^z\ \psi^y\ \psi^x]^\top$ (where ψ^x, ψ^y and ψ^z represent the Euler angles) is measured in the target frame, which is given by the end-effector pose when the connector is locked. For practical purposes, this target frame is assumed to be the last frame of each successful demonstration. We then learned a *vanilla* KMP [16] from D

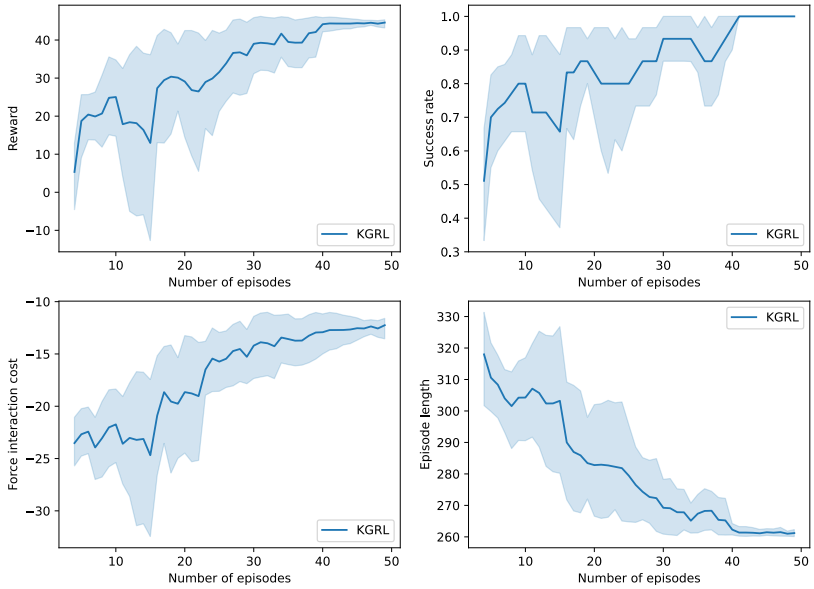


Figure 4.8: Performance of KGRL on BNC connector assembly task. The robot learns to solve the task in 45 episodes on an average, simultaneously minimizing the interaction force.

with $\boldsymbol{\xi}^{\mathcal{I}} = t$ and $\boldsymbol{\xi}^{\mathcal{O}} = \boldsymbol{p}_t$. This KMP was tested but found inadequate for task completion due to kinematic and dynamic uncertainties as well as the KMP's inability to effectively capture the contact dynamics involved in the task.

We then formulated a KGRL problem and a RL policy $\pi(\hat{\boldsymbol{\xi}}|\boldsymbol{s}_t)$ was learned to complete the task, where the state for the RL policy $\boldsymbol{s}_t = [t \ \boldsymbol{p}_t^\top \ \boldsymbol{f}_t^\top]^\top$ with $\boldsymbol{f}_t$ being the 6D wrench measured at the center of compliance, $\boldsymbol{\xi}^{\mathcal{I}} = t$, and $\boldsymbol{\xi}^{\mathcal{O}} = \boldsymbol{p}$. We defined the linear inequality constraints in XY-plane so that exploration does not deviate too far from the alignment pose,

$$\begin{aligned}
\boldsymbol{g}_{n,1}^\top &= [1\ 0\ 0\ 0\ 0\ 0], & c_{n,1} &= -0.002, \\
\boldsymbol{g}_{n,2}^\top &= [-1\ 0\ 0\ 0\ 0\ 0], & c_{n,2} &= -0.002, \\
\boldsymbol{g}_{n,3}^\top &= [0\ 1\ 0\ 0\ 0\ 0], & c_{n,3} &= -0.002, \\
\boldsymbol{g}_{n,4}^\top &= [0\ -1\ 0\ 0\ 0\ 0], & c_{n,4} &= -0.002, \quad \forall n = 1 \dots N.
\end{aligned} \tag{4.26}$$

Using $\boldsymbol{g}_{n,f}$ and $c_{n,f}$ formulated in eq. (4.26), matrices $\bar{\boldsymbol{G}}$ and $\bar{\boldsymbol{C}}$ are constructed.

The reward function for the RL agent is given by,

$$r_t = -0.01\hat{\boldsymbol{\xi}}_t^\top \hat{\boldsymbol{\xi}}_t - 0.01\boldsymbol{f}_t^\top \boldsymbol{f}_t + r_T, \tag{4.27}$$

$$r_T = \begin{cases} 60 & \text{at terminal step } T \text{ if successful,} \\ -50 & \text{if robot detects collision} \\ 0 & \text{otherwise.} \end{cases} \tag{4.28}$$

Since the DLR SARA robot can detect collisions observing anomalous joint torques, we use this signal in the reward function. The RL policy is learned using a DNN with 2 hidden layers with 256 neurons each and TQC for training, similarly to the simulation experiments. The TQC hyperparameters used in the experiment are summarized in Table 4.1.

Figure 4.8 illustrates the overall performance of KGRL. The robot successfully learned to insert and lock the connector in under 45 episodes while significantly reducing force interactions with the environment—an important factor for ensuring the robot's long-term safe operation. The robot achieved a success rate of 1 in 45 episodes, along with a decrease in episode length, which results in faster task completion.

4.5 DISCUSSION

With evaluations in simulation and on the real robot, we demonstrated that a complex task can be learned using KGRL even with a sparse reward function in a sample-efficient and safe manner. With KGRL, a simulated robot learns to achieve the primary goal of reaching the target while minimizing secondary costs, showing success rate of 1 from the beginning. Meanwhile, both residual RL approaches show unstable learning behavior with the same reward function, as shown Figure 4.5. In both residual RL approaches, isotropic exploration noise leads to counter-productive exploration near the narrow passage. Also, despite improving safety compared to the π_{res} baseline, the definition of π_{safe} comes at the cost of having to manually define wall constraints for the task to succeed, which is not required in KGRL, since the exploration behavior is extracted from the data.

As discussed above, KGRL is capable of learning complex contact tasks directly on the real robot. The safety of both robot and environment is facilitated by the hard constraints defined in the framework. For the task of inserting a BNC connector, a policy learned from demonstrations using KMP alone was not sufficient. With KGRL, we showed that the demonstrations can be used for accelerating RL by extracting the task completion strategy along with the exploration strategy. With the possibility of learning the task on a real robot safely and in a time-efficient manner, we largely alleviate the need of modeling the task meticulously in simulation. Furthermore, we use an off-the-shelf off-policy algorithm for learning the tasks. Meticulous hyper-parameter tuning was not necessary in our approach for learning the tasks successfully.

In Section 4.3.3, we discussed the properties of LC-NS-KMP. Given its ability to adapt based on demonstration variance while respecting constraints, LC-NS-KMP is not limited to RL and also offers desirable properties for LfD, especially on real robots. Also note that we pri-

oritized success rate and learning efficiency in our evaluations, and therefore did not explicitly focus on smooth exploration. Nevertheless, the framework accommodates additional velocity constraints–such as those proposed in [52] in the context of LfD–which can be defined to promote smoother exploratory behavior.

We would also like to highlight some limitations of our framework. The safety constraints in this work are manually defined and require expert knowledge during design; however, such constraints could also be extracted from demonstration data, e.g., upper and lower limits of motion in each degree of freedom. Another possible limitation is that we assume that the demonstrations encode an appropriate exploration strategy, which might not hold true in some cases. In such scenarios, a potential mitigation strategy is to manually tune the covariance matrix $\mathbf{\Sigma}$, as it is human-interpretable, leveraging the flexibility inherent in our approach.

4.6 Conclusion

In this chapter, we presented a novel movement primitive representation called Linearly Constrained Null-space Kernelized Movement Primitives (LC-NS-KMP), which can learn movement primitives from demonstrations allowing modifications through null-space actions, while respecting the linear inequality constraints. We leverage this movement primitive representation to deliver a novel constrained and guided RL method called Kernelized Guided Reinforcement Learning (KGRL). We evaluated our approach to highlight the effectiveness of KGRL in learning challenging manipulation tasks involving complex contacts directly on real robot. By integrating state-dependent guided exploration and linear inequality constraints, we were able to facilitate efficient learning and enhance the robot's operational safety. Our approach enables the robot to master connector insertion and locking in under 45 episodes, significantly reducing learning time and effort. Additionally, the reduction in force interactions with the environment indicates a pathway toward long-term reliability and safety in robot manipulation. Future work includes automating constraint extraction and incorporating non-linear constraints to extend applicability to broader assembly challenges.

5

KERNELIZED GUIDED REINFORCEMENT LEARNING WITH SMOOTH EXPLORATION

We present *Smooth Kernelized Guided Reinforcement Learning (sKGRL)* in this chapter which ensures smooth RL exploration on real robots.

RL enables the acquisition of complex robotic behaviors through exploration and exploitation. In classical RL approaches, exploration is achieved by adding unstructured, step-wise noise to policy actions, executing these actions in the environment, and collecting rewards to update the policy. While effective in simulation, this strategy often results in jerky, unstable motions and high accelerations when applied to real robotic systems, potentially damaging the robot or its environment. As discussed in Chapter 1, smooth exploration, characterized by structured, temporally correlated noise is crucial for ensuring the long-term safe operation of robots.

The safety-focused, guided RL framework Kernelized Guided Reinforcement Learning (KGRL), presented in Chapter 4, provides an RL method for learning robotic tasks while enforcing constraints to ensure safety. However, the exploration strategy in KGRL, which relies on unstructured noise, can still generate abrupt actions, as seen in Figure 4.4c. To address this limitation, in this chapter, we introduce *Smooth Kernelized Guided Reinforcement Learning (sKGRL)*, an extension of KGRL that incorporates a smooth exploration strategy. sKGRL leverages the continuity-inducing properties of kernel methods and incorporates the history of previous actions and robot states to limit acceleration during exploration, promoting both safe and efficient policy learning.

We evaluate sKGRL in simulation as well as on a real robotic system performing a BNC connector insertion task. By integrating smooth, state-dependent, and uncertainty-aware exploration into the guided RL framework, sKGRL demonstrates comparable real-world performance to KGRL with substantially reduced jerks in the motion.

The remainder of this chapter has been submitted as [36] Abhishek Padalkar et al. "Kernelized Guided Reinforcement Learning with Smooth Exploration." Manuscript under review at the IEEE International Conference on Robotics and Automation (ICRA), 2026. -.

5.1 INTRODUCTION

Reinforcement Learning (RL) has shown immense potential in enabling robots to learn complex tasks through interaction with their environments, holding significant promise for applications in au-

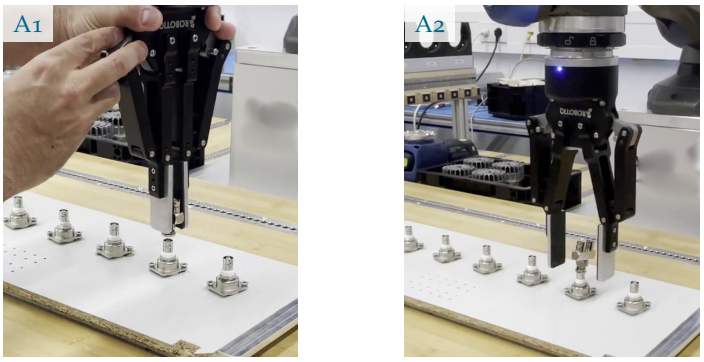


Figure 5.1: BNC connector assembly task from the NIST benchmark [35]. A1: human demonstration via hand-guided manipulation of a 7-DoF torque-controlled arm. A2: policy learned from a few demonstrations fails due to unmodelled contacts and kinematic uncertainties. We address this problem using kernel-based guided RL with smooth, constraint-respecting exploration in the vicinity of the demonstrations.

tonomous navigation, manipulation, and human-robot collaboration — particularly in scenarios where human demonstrations alone fail to capture the full task complexity (as shown, e.g., in Figure 5.1). However, efficient and safe exploration remains a critical challenge in applying RL to robotics, particularly in high-dimensional, continuous action spaces. Smooth exploration — characterized by controlled, gradual, and informed action selection — is a desirable property, as it promotes safe and efficient learning. Yet, in practice, RL often relies on unstructured exploration, as discussed in [19, 20]. Existing exploration methods, such as random action perturbations (e.g., ϵ-greedy or Gaussian noise), add randomly sampled unstructured exploration noise to the policy action. This often leads to erratic, abrupt robot behavior and lack of temporal consistency, increasing the risk of hardware damage and slowing convergence. Although highly effective in simulation environments, such unstructured, random, step-based exploration noise in RL leads to high accelerations.

A variety of recent research has focused on ensuring safe RL on real robots [32] (see Section 5.2 for a review of related work). One class of approaches focuses on imposing constraints (both soft and hard) on the robot's state space during learning, preventing exploration of unsafe regions and favoring solutions close to prior demonstrations [33, 34], presented in Chapters 3 and 4. Another class addresses safety by focusing on the optimal design of exploration noise [20]. To the best of our knowledge, the simultaneous consideration of constraints and smooth exploration strategies remains largely unaddressed in the context of real-world robot RL. To address this gap, we build on a constrained, guided RL approach that enables state-dependent, uncertainty-aware exploration — Kernelized Guided Reinforcement Learning (KGRL) [34] — and extend it with mechanisms that promote

continuity (Section 5.3). KGRL uses demonstrations to initialize a nominal policy distribution and derives the exploration strategy from its covariance. It also allows the imposition of linear inequality constraints to keep the robot within safe regions of the state space. Our solution combines the strong continuity priors imposed by smooth kernels — such as the radial basis function (RBF) kernel — with KGRL's state-dependent exploration to encourage future states to remain close to previously visited ones, thereby promoting smoothness in the resulting trajectories. Specifically, we prepend the distribution over future states with the history of measured states, assigning low uncertainty to the latter since they have been observed, promoting continuity within each training episode.

We evaluate our method, Smooth Kernelized Guided Reinforcement Learning (sKGRL) in an illustrative example in 2D and on a real robot performing a BNC insertion task (Figure 5.1); see Section 5.4.

5.2 Related Work

Entropy-based reinforcement learning (RL) methods, such as Soft Actor-Critic (SAC), are widely used for exploration in continuous action spaces by balancing exploration and exploitation through entropy regularization [62]. However, their tendency to produce abrupt action changes limits their applicability in robotics. Parameter-space exploration methods, such as Natural Evolution Strategies [133], Black-Box Optimization [134], and Policy Gradients with Parameter-Based Exploration [135], add noise to policy parameters at the start of each episode [136]. [137] and [138] learn the weights of a motion primitive (MP) using a deep neural network policy. The policy generates a set of weights that parameterize the MP, which in turn predicts a desired trajectory given the weights and initial conditions. [139] performs exploration in parameter space, but updates the policy using segments of trajectories rather than treating the entire trajectory as a single data point. This infrequent noise injection necessitates a large number of episodes to estimate policy gradients, leading to sample inefficiency.

State-Dependent Exploration (SDE) [19] generates noise using a function of the robot's state and randomly sampled parameters, ensuring consistent actions for identical states within an episode. However, parameter sampling occurs at the episode level, resulting in sample inefficiency. Generalized State-Dependent Exploration (gSDE) [20] extends SDE by sampling noise every n steps instead of per episode and using policy features (e.g., the output of the last layer of the RL policy) as the noise function's input, instead of the robot state. Despite this improvement, gSDE produces abrupt action changes every n steps. Similarly, the method in [140] injects temporally correlated noise into the latent state of the policy network, using two learned Gaussian distributions to generate noise matrices for latent and action noise,

applied every n steps. Like gSDE, this approach results in abrupt, unstructured actions at these intervals.

Exploration using pink noise, which combines temporally uncorrelated white noise and temporally correlated colored noise, is explored in [141]. While colored noise has proven effective for learning policies in continuous action spaces with deep RL [68], its combination with white noise disrupts action continuity during exploration. [34] presents a state-dependent uncertainty-aware exploration method called Kernelized Guided Reinforcement Learning (KGRL) which leverages demonstrations for guiding RL. Additionally, KGRL enables the imposition of linear inequality constraints on the robot's state, enhancing safety for robotic applications. Our method, sKGRL, extends this method further for smooth RL exploration by considering history of the robot state in the reference trajectories used in KGRL. It preserves all desirable properties of KGRL namely state-based uncertainty-aware exploration and constraints enforcement.

Low-pass filters are commonly used in reinforcement learning to smooth actions and enhance stability in robotic control. [142] applied them to quadruped locomotion, [143] to humanoid robots, and Narang et al. [23] to contact-rich manipulation in simulated environments. Similarly, Neunert et al. [144] employed action repetition for temporal smoothing in continuous control tasks. These techniques improve policy stability by mitigating high-frequency noise. In contrast, our method induces smoothness without requiring low-pass filters in the robot's low-level control loop. Consequently, it integrates seamlessly with off-the-shelf robotic control architectures while leveraging state-based, uncertainty-aware exploration for data-efficient learning.

5.3 METHODOLOGY

We introduce an RL framework that builds on the Kernelized Guided Reinforcement Learning (KGRL) approach presented in Section 4.3.4. Our framework extends KGRL with a smooth exploration strategy, leveraging the continuity-inducing properties of kernel-based methods to promote safe and efficient policy learning. In Section 5.3.1, we briefly revisit the KGRL formulation. We then present our novel exploration approach in Section 5.3.2 and discuss its key properties in Section 5.3.3.

5.3.1 *Kernelized Guided Reinforcement Learning (KGRL)*

KGRL [34] aims to derive a model of the demonstrations that quantifies the variance and correlations between datapoints while simultaneously adhering to hard constraints. Additionally, it derives an uncertainty-aware term that enables exploration informed by demonstration data. KGRL proposes to obtain *null space actions* $\hat{\boldsymbol{\xi}}$ from an

RL policy $\pi(\hat{\boldsymbol{\xi}}|\boldsymbol{s})$, to modulate the LfD trajectory learned from the demonstrations. Recalling from Section 4.3.4, these null-space actions modify the prediction for further refinement as

$$\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}}) = \boldsymbol{k}^* \boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{k}^* \boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\beta}{\hbar}(\hat{\boldsymbol{k}}^* - \boldsymbol{k}^* \boldsymbol{A}\hat{\boldsymbol{K}})\hat{\boldsymbol{\xi}}, \tag{5.1}$$

with $\boldsymbol{A} = (\boldsymbol{K} + \hbar\boldsymbol{\Sigma})^{-1}$, $\mathcal{K} = -\frac{1}{2}\boldsymbol{K}\boldsymbol{\Sigma}^{-1}\boldsymbol{K} - \frac{\hbar}{2}\boldsymbol{K}$, and scalar parameters β and $\hbar$. Lagrange multipliers $\boldsymbol{\alpha}$ are obtained by solving a quadratic program with linear inequality constraints,

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\textbf{argmax}} \quad & \boldsymbol{\alpha}^\top \mathcal{B}_1 \boldsymbol{\alpha} + \mathcal{B}_2 \boldsymbol{\alpha}, \\ \text{s.t.} \quad & \boldsymbol{\alpha} \geq 0, \end{aligned} \tag{5.2}$$

where,

$$\mathcal{B}_1 = \bar{\boldsymbol{G}}^\top \boldsymbol{\Sigma} \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}}, \quad \mathcal{B}_2 = 2\boldsymbol{\mu}^\top \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} - \beta \hat{\boldsymbol{\xi}}^\top \underline{\boldsymbol{K}}^{-1} \hat{\boldsymbol{K}} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} + \bar{\boldsymbol{C}}^\top.$$

Matrices $\bar{\boldsymbol{G}}$ and $\bar{\boldsymbol{C}}$ parameterize the linear inequality constraints imposed on the output. Please visit Section 4.3.4 for more details and for the definitions of the matrices $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, $\boldsymbol{k}^*$, $\bar{\boldsymbol{G}}$, $\bar{\boldsymbol{C}}$, $\boldsymbol{\alpha}$, $\hat{\boldsymbol{k}}^*$, and $\hat{\boldsymbol{K}}$. As shown in [34], $\frac{\beta}{\hbar}(\hat{\boldsymbol{k}} - \boldsymbol{k}\boldsymbol{A}\hat{\boldsymbol{K}})\underline{\boldsymbol{K}}^{-1}$ acts as a *soft null space projector*, adapting the trajectory expectation in line with the data variance—with larger variance allowing stronger deformations and smaller variance constraining them. After solving for $\boldsymbol{\alpha}$, Equation (5.1) enables constrained predictions incorporating modulations from $\hat{\boldsymbol{\xi}}$.

5.3.2 *A Smooth Exploration Approach for KGRL (sKGRL)*

To enable smooth exploration in KGRL, we leverage its inherited smoothness bias from KMPs [16], which uses smooth kernels imposing continuity priors on generated trajectories and adjusts them based on demonstration variance. Specifically, KGRL's state-dependent, uncertainty-aware exploration employs null-space actions to induce greater modifications in high-variance regions than in low-variance ones. However, standard KGRL exploration: 1) generates a reference trajectory distribution of horizon H from the GMM at each time step, 2) applies a noisy *soft null space* action to its first point to create a modification, 3) executes only that first modified point on the robot, and 4) discards the rest before resampling a new reference trajectory distribution. This causes the robot to jump between unrelated modified trajectories at each step, resulting in jerky motion, as illustrated in Figure 5.2a. To achieve smooth exploration, we propose prepending the history of robot states to the newly sampled reference distribution T_r with very low variance, realistically assuming that these states are known precisely from robot sensors. The resulting modified distribution is

$$T_r' = \left\{\boldsymbol{\mu}_n', \boldsymbol{\Sigma}_n'\right\}_{n=-P}^{0} \cup \left\{\hat{\boldsymbol{\mu}}_n, \hat{\boldsymbol{\Sigma}}_n\right\}_{n=1}^{H}, \tag{5.3}$$

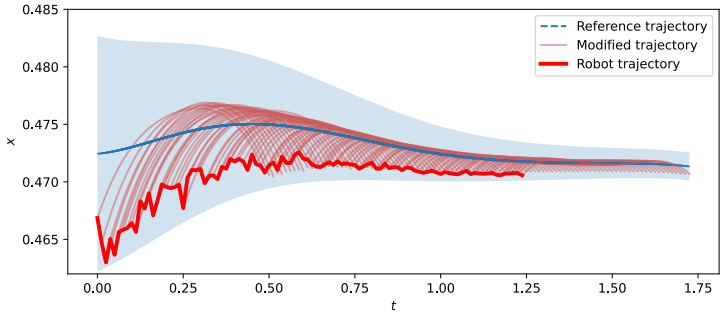
Algorithm 3 Smooth Kernelized Guided Reinforcement Learning (sKGRL)

Collect demonstrations $D \leftarrow \{\{\boldsymbol{\xi}^{\mathcal{I}}_{n,m}, \boldsymbol{\xi}^{\mathcal{O}}_{n,m}\}_{n=1}^{N}\}_{m=1}^{M}$
Set λ, β, ρ and define $k(.,.)$
Set horizon H, history window size P
Initialize historical trajectory distribution $T_p = \{\}$
Model joint probability distribution $\mathcal{P}(\boldsymbol{\xi}^{\mathcal{I}}, \boldsymbol{\xi}^{\mathcal{O}})$
Define set of constraints $O = \{\{\boldsymbol{g}^{\top}_{n,f}, c_{n,f}\}_{n=1}^{N}\}_{f=1}^{F}$
Initialize a RL policy $\pi(\hat{\boldsymbol{\xi}}|\boldsymbol{s})$ and policy parameters
loop for each $n = 1, \ldots, N$
    In current state $\boldsymbol{\xi}^{\mathcal{I}}_n$, retrieve the reference trajectory distribution $T_r \leftarrow \{\hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\Sigma}}_i\}_{i=n}^{n+H}$
    Prepend T_p to T_r to obtain $T_r^{'}$ as per Equation (5.3)
    Choose constraints for $T_r^{'}$ from O
    Sample RL action $\hat{\boldsymbol{\xi}}$ from RL policy π
    Compute $\boldsymbol{\alpha}$ using Equation (5.2)
    Compute $\mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}_n))$ with Equation (5.1)
    Apply $\mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}(\boldsymbol{\xi}^{\mathcal{I}}_n))$ to robot
    Compute reward
    Update RL policy π
    Update T_p with new measured robot state $\boldsymbol{\mu}^{'}$ and $\boldsymbol{\Sigma}^{'} = \rho I$
end loop

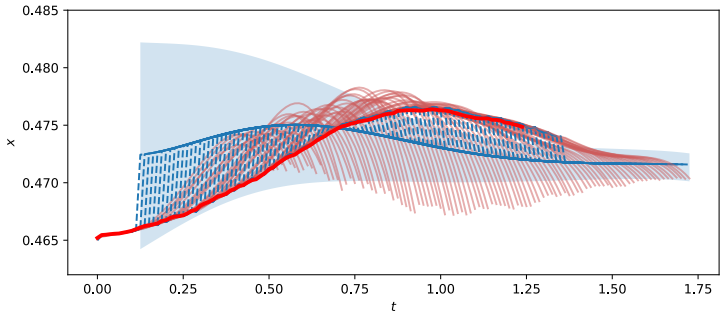
where P is the number of history points and the exploration action is applied at $n = 1$. $\boldsymbol{\mu}^{'}$ is the measured robot state and the covariance is defined as $\boldsymbol{\Sigma}^{'} = \rho \boldsymbol{I}$ where ρ is a small scalar factor. Unlike standard KGRL, this combines a rolling history with a future reference trajectory distribution sampled from the probabilistic model (GMM modelling demonstrations D), ensuring the robot maintains a continuous moving reference distribution rather than jumping between unrelated distributions. Algorithm 3 details the sKGRL algorithm for one RL episode. The steps shown in the blue text differentiate the sKGRL algorithm from KGRL algorithm presented in Algorithm 2. Notably, while the underlying offline RL policy still generates random, unstructured noisy actions, the sKGRL framework produces principled trajectory modifications leveraging the aforementioned properties.

5.3.3 *Smoothness Properties of sKGRL*

Figure 5.2a and Figure 5.2b illustrate the differences in exploration between KGRL and sKGRL, highlighting the smooth exploration by sKGRL. In KGRL, a reference trajectory distribution is sampled at each time step, a null-space action is applied to yield a modified trajectory, and only the first command is executed by the robot. This process, depicted in Figure 5.2a, produces jerky motion, with dotted blue lines



(a) KGLR exploration



(b) sKGLR exploration

Figure 5.2: Comparison of the reference trajectories, modified trajectories and robot trajectory in KGRL and sKGRL. Blue dotted lines show the reference trajectories. Red lines denote the trajectories modified by RL exploration. Red solid line indicates the resultant trajectory of the robot. Shaded blue region shows the variance in the reference trajectory distribution.

denoting mean reference trajectories with $H = 40$, red lines indicating modified trajectories, and the solid red line tracing the robot's executed trajectory. In contrast, sKGRL (Figure 5.2b) prepends measured robot states with low variance to the reference trajectory distribution, ensuring smooth exploration. As shown in Figure 5.2b, dotted blue lines represent means of these augmented reference trajectory distributions with $H = 40$ and $P = 10$, red lines denote sKGRL-modified trajectories, and the solid red line marks the robot's executed trajectory. In both figures, the blue shaded region shows the relative variance of the reference trajectories sampled from probabilistic model.

In both KGRL and sKGRL, without perturbation, the robot would continue along the modified (red) path from each time step onward; however, to enable exploration, a new modified path is generated at every time step using a reference trajectory distribution and a new null-space action. In KGRL, this involves sampling a new reference trajectory distribution at each step, yielding unrelated modified trajectories after applying the action. In contrast, sKGRL maintains a moving reference trajectory distribution that incorporates the robot's historical states including current state with low variance, resulting in a smooth modified trajectory passing through the current robot

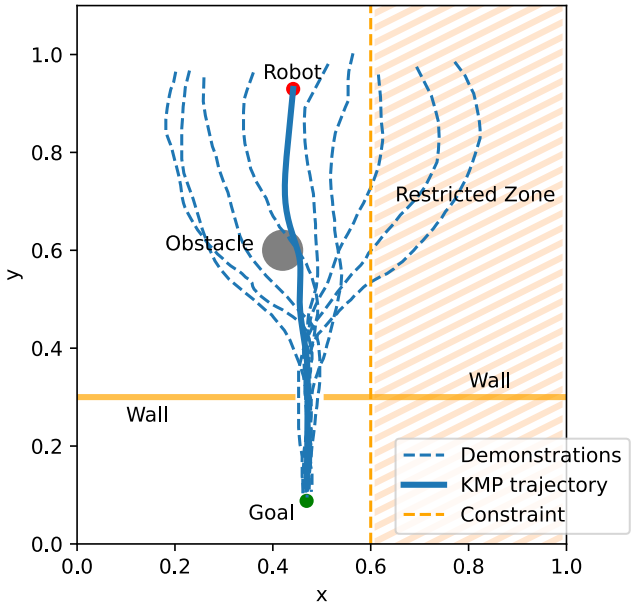


Figure 5.3: Simulated 2D environment where the robot navigates through a narrow passage to reach the goal. A trajectory for navigation can be learned from demonstration. Then an RL policy learns to avoid the obstacle in the path of the robot. The robot must not cross into the restricted zone.

pose. Uncertainty-aware exploration enforces passage through the current pose (due to its minimal variance), while smoothness bias from smooth kernels ensures continuity to the next pose in the modified trajectory which will be executed on the robot.

5.3.4 *Extending sKGRL for Handling of External Perturbations*

Our proposed approach, sKGRL, can be extended to handle external perturbations in the Learning from Demonstrations context. sKGRL inherently replans the robot trajectory at every time step using the modified reference trajectory distribution $T_r^{'}$, which contains the history of measured robot states. With its smoothness properties, sKGRL robustly handles external perturbations without generating discontinuities in the replanned trajectories, even when perturbations affect the current robot state.

These perturbations may arise from external influences, for example, human interactions during execution. By setting $P = 1$, sKGRL replans the future trajectory to pass through the current perturbed state of the robot. Sampling the reference trajectory distribution at every time step and prepending the current state with low variance in the reference distribution ensures trajectory continuity even under external perturbations.

The constraint enforcement mechanism further ensures that the replanned trajectory respects linear inequality constraints in the pres-

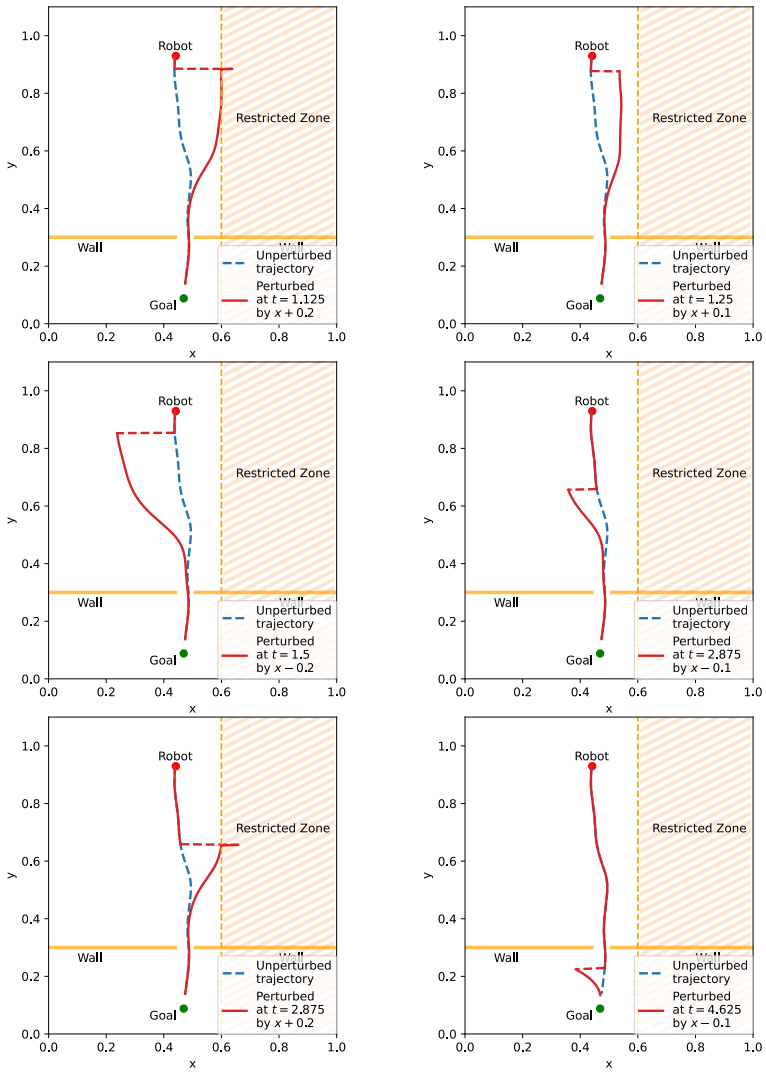
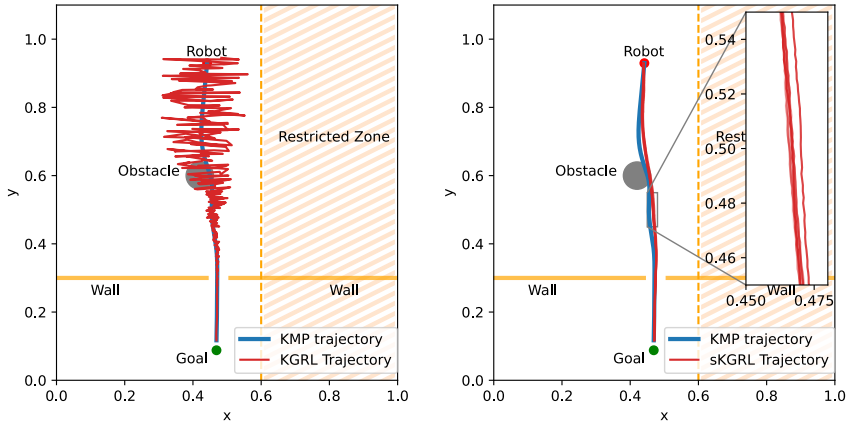


Figure 5.4: Figure shows the robust handling of external perturbations (shown by dashed red lines) introduced at various time instances. Variance informed recovery behavior respects the imposed lineal inequality constraints.

ence of perturbations. Recovery from a perturbed state is variance-informed: in regions of high demonstration variance, recovery is less aggressive and takes longer to merge with the nominal (unperturbed) behavior, allowing greater deviation over a longer time horizon. Conversely, in regions of low demonstration variance, the replanned trajectory merges more aggressively with the nominal trajectory while maintaining smoothness. It should be noted that null-space actions can still introduce additional, desired modifications to the trajectory.

Using simulated 2D toy example in Figure 5.3, Figure 5.4 illustrates the effect of perturbations at different time instances on the robot trajectory. The discussed properties—variance-informed perturbation handling and replanning while respecting constraints—are clearly evident from the robot's behavior in the plots.



(a) KGLR exploration (b) sKGRL exploration

Figure 5.5: Comparison of KGRL and sKGRL exploration. Figure 5.5a shows 1 KGRL trajectory and Figure 5.5b shows 5 different sKGRL trajectories to highlight gradual, structured evolution of trajectories. sKGRL maintains single reference trajectory distribution by prepending historical state observations to newly sampled reference trajectory distribution. By not keeping history of visited states, KGRL makes robot jump between different trajectories.

5.4 EVALUATION

5.4.1 *Two-dimensional Toy Examples*

We evaluate the performance of our proposed framework, sKGRL against KGRL. For this evaluation, we used a simulation involving a robot that navigates a 2D environment with the primary objective of reaching a goal position while passing through a narrow passage, see Figure 5.3, similar to Chapter 4. As illustrated in Figure 5.3, demonstrations $D = \{\{t_{n,m}, \boldsymbol{p}_{n,m}\}_{n=1}^{400}\}_{m=1}^{9}$ were given to the robot. However, after demonstrating the trajectories, an obstacle is added along the path of the robot so that the mean trajectory consistently intersects with the obstacle. Moreover, we define a *Restricted Zone* ($x \geq 0.6$) for the robot, as shown in Figure 5.3, after the demonstrations were collected. The robot must not go into the Restricted Zone to ensure safety during learning and execution. Consequently, an RL agent must learn to avoid the obstacle and not to enter the restricted zone, while still successfully reaching the goal. The primary objective of this analysis is to compare the ability to conduct smooth exploration of these algorithms and to compare the learning performance. Both sKGRL and KGRL conduct state-dependent uncertainty-aware exploration while respecting imposed linear inequality constraints, ensuring the robot does not enter the restricted zone.

To learn an RL policy for avoiding obstacle and reaching goal, we used the same reward function and Truncated Quantile Critics (TQC)

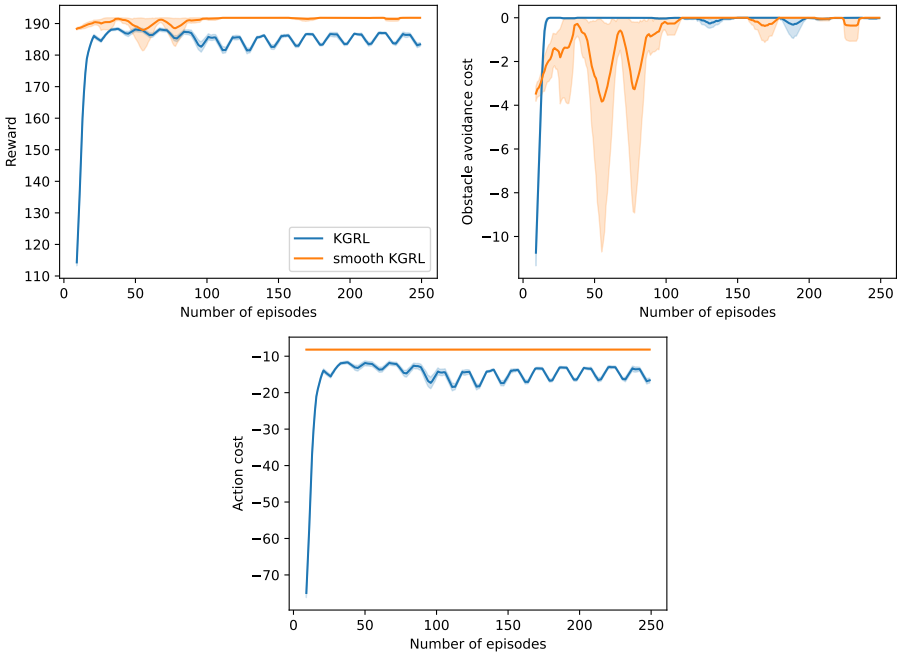


Figure 5.6: Comparison of the learning performance of KGRL and sKGRL. sKGRL shows overall superior performance in terms of total rewards achieved with structured, smooth exploration which results in lower action cost. sKGRL takes comparatively more time to learn to avoid obstacles due to slower and gradual exploration in the state-space. The figure shows the average learning performance for 10 learning trials for each approach.

algorithm [111] for KGRL as well sKGRL. The reward function for the robot is given by

$$r_t = r_a + r_o + r_T, \tag{5.4}$$

$$r_a = -10\delta \boldsymbol{p}_t^\top \delta \boldsymbol{p}_t, \tag{5.5}$$

$$r_o = \begin{cases} -100(0.04 - \boldsymbol{d}_t), & \text{if } \boldsymbol{d}_t \leq 0.04 \\ 0 & \text{otherwise} \end{cases} \tag{5.6}$$

$$r_T = \begin{cases} 200 & \text{at terminal step } T, \text{ if successful} \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

where $\delta \boldsymbol{p}_t$ is the displacement of the robot, $\boldsymbol{d}_t$ is the distance of the robot from the obstacle, the terminal reward r_T is given if the episode terminates successfully, r_o is the obstacle avoidance cost, and r_a is the action cost. For both KGRL and sKGRL, $\boldsymbol{\xi}^{\mathcal{I}} = t$ and outcome is $\boldsymbol{\xi}^{\mathcal{O}} = \boldsymbol{p}_t$. For sKGRL, we define $P = 10$, $H = 40$, $\lambda = 6$, $\beta = 6$, and $\rho = 10^{-6}$. The robot follows the resultant 2D position $\hat{\boldsymbol{p}}_t = \mathbb{E}(\boldsymbol{\xi}^{\mathcal{O}}(t))$, derived from Equation (5.1) where modulations of the mean trajectory originate from the RL policy $\pi(\hat{\boldsymbol{\xi}}|t)$.

We compared the exploration trajectories of KGRL and sKGRL as shown in the Figure 5.5a and Figure 5.5b. KGRL exploration in

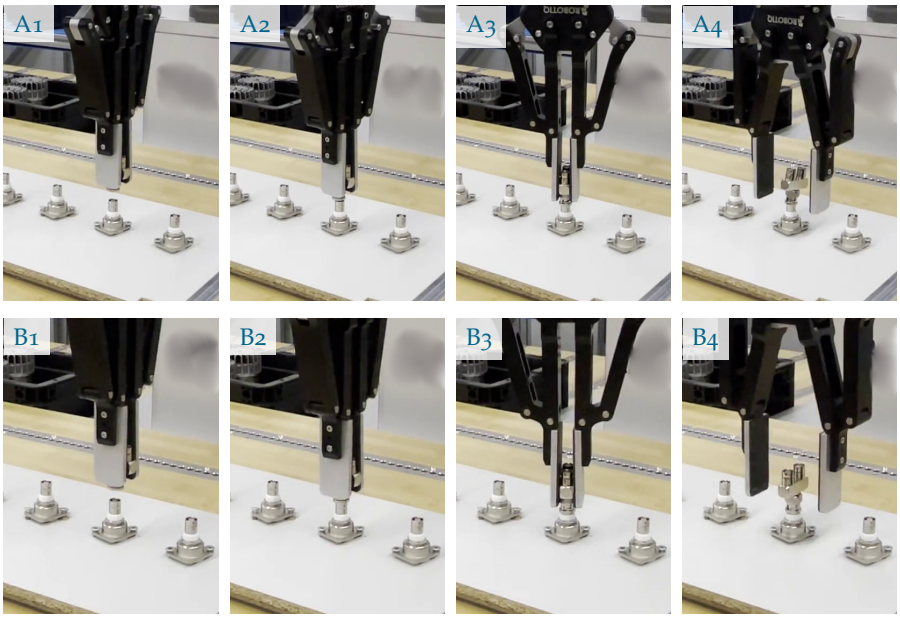


Figure 5.7: sKGRL learning BNC connector assembly task. A1 to A4 show an unsuccessful episode while learning the task. B1 to B4 show that sKGRL learned the task successfully.

Figure 5.5a shows that robot explores around the mean trajectory following the uncertainty aware exploration, but the exploration is not smooth and leads to erratic robot motions due the high acceleration components in the robot motion. On the other hand, sKGRL exploration in Figure 5.5b shows more structured exploration where the robot trajectory evolves in a structured and smooth manner, without generating high accelerations in the robot.

Figure 5.6 shows the learning performance of sKGRL and KGRL. sKGRL shows overall superior performance in terms of total rewards achieved with structured exploration. This increase in performance is achieved primarily due to the smooth exploration which results in lower action cost. However, sKGRL takes comparatively more time to learn to avoid obstacles due to slower and gradual exploration in the state-space.

5.4.2 *Real Robot Experiment*

To evaluate our approach on a real robot, we selected a task from the NIST assembly benchmark 1 [35] involving the plugging of a BNC connector, similarly to [34]. This challenging task requires multiple manipulation strategies for different phases: aligning, inserting, and locking the connector. The strategy learned from demonstration alone is insufficient to complete the task, as shown in Figure 5.1, and relying purely on RL would necessitate an impractically large number of trials. Both KGRL and our approach sKGRL, use state-dependent

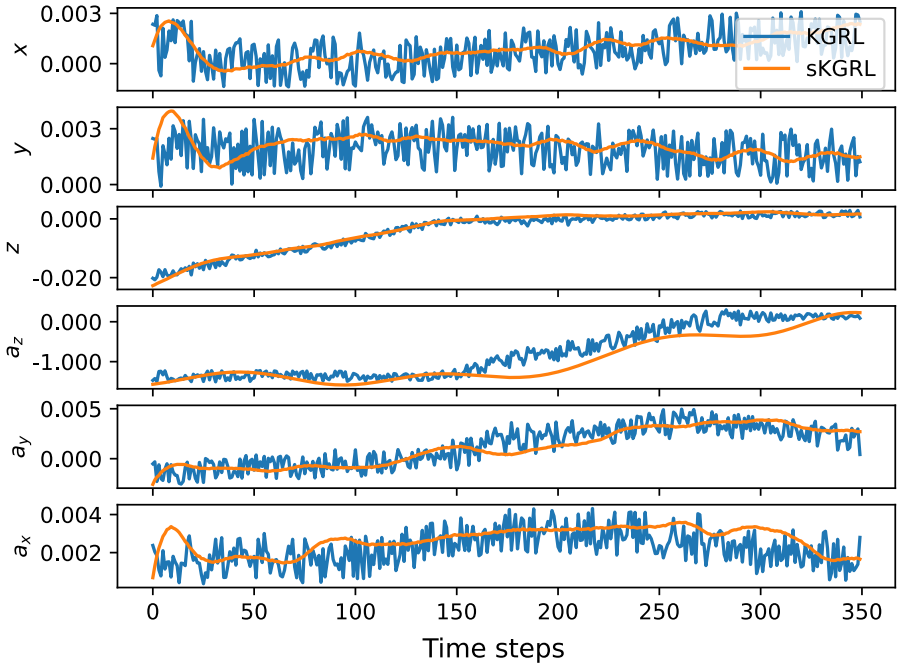


Figure 5.8: Comparison of KGRL and sKGRL exploration on the real robot. The KGRL trajectories show jumps in the robot positions amounting to the higher accelerations in the robot, due to the unstructured exploration. sKGRL conducts principled exploration leading to smoother trajectories.

guided exploration, allowing the robot to explore the state-action space selectively, where necessary. Linear inequality constraints reduce the state space and ensure the robot's safety. Additionally, smooth exploration strategy used in sKGRL ensures principled exploration. The experimental setup for learning the task of inserting a BNC connector is shown in Figure 5.7.

Figure 5.8 compares exploration trajectories generated by sKGRL and KGRL. sKGRL ensures that the modification introduced by RL exploration ensures the continuity of the robot trajectory. On the other hand, KGRL trajectories show jumps in the robot poses, hence amounting to higher accelerations in the robot.

We formulated KGRL and sKGRL problems and an RL policy $\pi(\hat{\boldsymbol{\xi}}|\boldsymbol{s}_t)$ was learned to complete the task in both cases with TQC algorithm, where the state for the RL policy is $\boldsymbol{s}_t = [t\ \boldsymbol{p}_t^\top\ \boldsymbol{f}_t^\top]^\top$ with $\boldsymbol{f}_t$ being the 6D wrench measured at the center of compliance, $\boldsymbol{\xi}^{\mathcal{I}} = t$, and $\boldsymbol{\xi}^{\mathcal{O}} = \boldsymbol{p}$. We defined the linear inequality constraints in XY-plane so that exploration does not deviate too far from the alignment pose,

$$\begin{aligned}
\boldsymbol{g}_{n,1}^\top &= [1\ 0\ 0\ 0\ 0\ 0], \quad & c_{n,1} &= -0.002, \\
\boldsymbol{g}_{n,2}^\top &= [-1\ 0\ 0\ 0\ 0\ 0], \quad & c_{n,2} &= -0.002, \\
\boldsymbol{g}_{n,3}^\top &= [0\ 1\ 0\ 0\ 0\ 0], \quad & c_{n,3} &= -0.002, \\
\boldsymbol{g}_{n,4}^\top &= [0\ -1\ 0\ 0\ 0\ 0], \quad & c_{n,4} &= -0.002, \ \forall n = 1 \dots N.
\end{aligned} \tag{5.8}$$

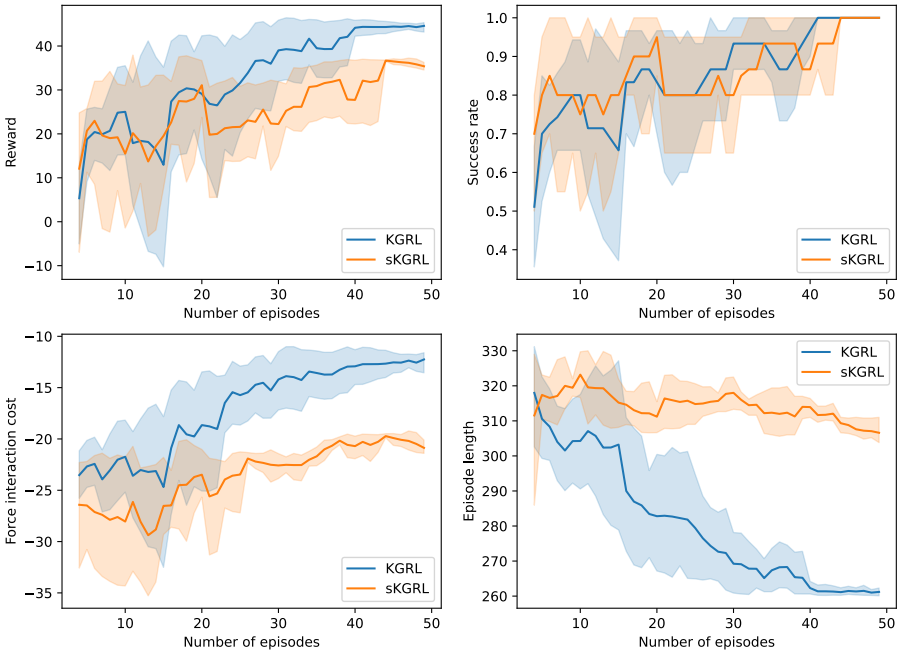


Figure 5.9: Comparison of KGRL and history based sKGRL in learning BNC connector insertion task on the real robot. The figure shows the average learning performance for 10 learning trails for each approach.

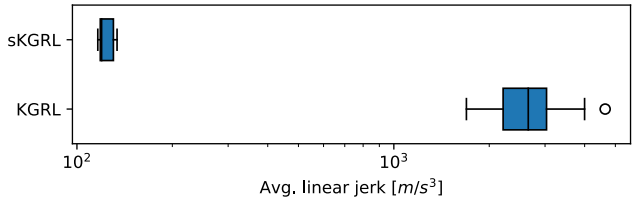
Using $\boldsymbol{g}_{n,f}$ and $c_{n,f}$ formulated in Equation (5.8), matrices $\bar{\boldsymbol{G}}$ and $\bar{\boldsymbol{C}}$ are constructed. For sKGRL, we define $P = 10$, $H = 40$, $\lambda = 6$, $\beta = 6$, and $\rho = 10^{-8}$. The reward function for the RL agent in both cases is given by,

$$r_t = -0.01\hat{\boldsymbol{\xi}}_t^\top \hat{\boldsymbol{\xi}}_t - 0.01\boldsymbol{f}_t^\top \boldsymbol{f}_t + r_T, \tag{5.9}$$

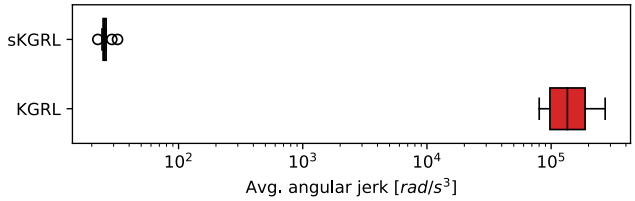
$$r_T = \begin{cases} 60 & \text{at terminal step } T \text{ if successful,} \\ -50 & \text{if robot detects collision} \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

Figure 5.9 compares the learning performance of KGRL and sKGRL in learning BNC insertion task. Both approaches learn the task under 45 episodes. KGRL shows success rate of 1 in 42 episodes whereas sKGRL shows success rate of 1 in 44 episodes, demonstrating comparable performance in terms of success rate. Post-convergence, sKGRL achieves lower maximum rewards than KGRL due to reduced optimization of interaction force costs and episode lengths.

Figure 5.10 compares the average jerk norm in trajectories generated by KGRL and sKGRL during learning. The figure shows the distribution of the average norm of the end-effector jerk, $\bar{\boldsymbol{j}}$ across first 10 episodes during a learning trial for each approach, where $\bar{\boldsymbol{j}} = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{j}_i\|$, computed separately for the linear and the angular components, where N is the length of the trajectory. The boxplots in



(a) Linear jerk



(b) Angular jerk

Figure 5.10: Comparison of the average jerk in KGRL and sKGRL exploration on the real robot. sKGRL trajectories exhibit significantly smaller jerk than the KGRL trajectories.

Figure 5.10 depict these jerks in blue for the linear components and red for the angular components. The sKGRL trajectories exhibit jerks that are orders of magnitude smaller than those of the KGRL trajectories, underscoring the effectiveness of sKGRL in enabling smoother exploration.

5.5 DISCUSSION

Being guided RL approaches, KGRL and sKGRL learn the tasks successfully in data-efficient manner, conducting state-dependent uncertainty-aware exploration. Constraint enforcement mechanism in both approaches ensured safe exploration while learning. In simulation experiments in Section 5.4.1, sKGRL exhibited structured but less aggressive exploration than KGRL, requiring more time to cover the same state space and thus slowing the learning of obstacle avoidance which was reflected in the obstacle avoidance cost (Figure 5.6). Structured exploration led to more stable learning and higher overall reward for sKGRL due to smaller action cost, whereas KGRL's unstructured exploration incurred very high action costs that were optimized over time. Due to the strong prior on smoothness from the exploration strategy in sKGRL, the action cost was not as high as KGRL from the beginning to optimize.

On the real robot (Section 5.4.2), sKGRL achieved comparable learning performance to KGRL in terms of success rate while generating smoother exploration trajectories for enhanced safety. Both methods learned the task in 45 episodes, reaching a success rate of 1 (Figure 5.9). Moreover, sKGRL produced jerks in exploration trajectories that were orders of magnitude lower than those in KGRL, providing an additional safety layer for the robot (Figure 5.10). However, post-

convergence, KGRL yielded higher rewards due to the greater reactiveness enabled by unconstrained exploration, which better optimized force interaction costs. In contrast, sKGRL's inherent smoothness bias limited instantaneous corrective actions against high contact forces, as actions exerted gradual effects over longer horizons. This limitation could be addressed by increasing the RL policy's control command update rate, which sKGRL supports safely without inducing the shaky behaviors typical of classical RL.

In both experiments, sKGRL enhanced overall robot behavior during learning, achieving orders-of-magnitude reductions in jerk while preserving task goal acquisition in data-efficient manner. Though this came at the cost of reduced reactivity, our goal was to demonstrate that uncertainty-aware methods like KGRL can be further extended for smooth exploration for enhanced safety. Suboptimal secondary cost optimization from low reactivity could be mitigated by task-phase-specific tuning of P, enabling more reactive actions (e.g., during contact) while maintaining smoothness elsewhere (e.g., in free space). Notably, selectively tuning exploration via a single parameter P could unify unstructured strategies from classical RL[34, 62] with the smoothness of evolution strategies, black-box optimization, and parameter-based exploration[133–135, 137].

5.6 CONCLUSION

In this chapter, we presented sKGRL, an extension of Kernelized Guided Reinforcement Learning (KGRL) that enables smooth, principled exploration in reinforcement learning while preserving KGRL's state-dependent, uncertainty-aware exploration and ability to enforce linear inequality constraints. By prepending low-variance robot state history to the reference trajectory distribution, sKGRL ensures continuity in the trajectories and reduces jerky motions, yielding jerk magnitudes orders of magnitude lower than KGRL. In simulation, sKGRL outperformed KGRL in per-episode rewards via lower action costs and stable learning, despite needing slightly more episodes to reach 100% success due to conservative exploration. On the real robot, sKGRL matched KGRL's 100% success convergence in 45 episodes with smoother exploration. Although post-convergence optimization of force interaction costs and episode lengths was slightly lower than KGRL, sKGRL enhanced safety with substantially reduced jerks in the motion during learning.

In future work, to mitigate the issue of optimization of force interaction cost, we will work on enhancing sKGRL's reactiveness by using higher command update rate and task-phase-dependent P to enable reactive exploration in necessary regions. Vision Language Models [145] can be deployed to infer such value of P based on the phase of the task.

6 CONCLUSION AND OUTLOOK

6.1 CONCLUSION

In this work, we developed novel algorithms and methods for guiding Reinforcement Learning (RL) using *task knowledge*. As outlined in Chapter 1, the application of RL in robotics is hindered by the need for a large number of training episodes and by safety concerns associated with unsafe exploration. This requirement for extensive training makes the learning process time-consuming and mechanically demanding. Moreover, exploratory actions and those generated by a newly initialized policy can be unsafe for direct execution on a real robot, potentially causing damage to the robot and its environment. We demonstrated that incorporating task knowledge and constraints into the RL process accelerates learning and promotes safe exploration, as validated through the experiments in Sections 3.4, 4.4 and 5.4. Faster and safer learning enables RL algorithms to be deployed directly on real robots, thereby reducing the reliance on simulations. Consequently, this approach helps mitigate the sim-to-real gap, a persistent challenge in transfer learning arising from differences between simulated and real environments, particularly in contact-rich manipulation tasks. By leveraging task knowledge and constraints, our methods substantially reduce the need for time-consuming and mechanically demanding real-world robot–environment interactions. Finally, through the direct imposition of constraints during exploration, our approach ensures safe RL behavior throughout the learning process.

In this work, we developed two novel frameworks—Reinforcement Learning with Shared Control Templates (RL-SCT) and Kernelized Guided Reinforcement Learning (KGRL)—to facilitate the incorporation of task knowledge and constraints into the RL process. Furthermore, we introduced an extension of KGRL, termed Smooth Kernelized Guided Reinforcement Learning (sKGRL), which enhances safety through a principled, smooth exploration strategy.

6.1.1 *Guiding RL with Hard-coded Task Knowledge and Constraints*

In Chapter 3, we presented RL-SCT for guiding RL through constraints represented as Shared Control Templates. We demonstrated that the properties typically expected from shared control: empowerment through freedom of movement, safety via constraint enforcement, and low-dimensional input commands that facilitate control, are equally

advantageous for RL on a robot. Our experiments demonstrated that the explicit representation of constraints leads to faster learning, without the need to design complex reward functions to encode these constraints. In particular, in Section 3.4, we demonstrated that RL-SCT enables effective RL on real robots. In a pouring liquid task (without contact between the robot and environment), RL-SCT allowed the robot to learn the task in only 16 minutes, without any unsafe interactions. Given the importance of safety in contact-rich tasks, we further applied our approach to a grid-clamp insertion task—similar to peg-in-hole—under position uncertainty, to learn a policy that successfully completes the task while minimizing contact forces. As with the pouring task, the robot quickly learned the insertion task in approximately 17 minutes, exhibiting substantially lower contact forces compared to a baseline that did not account for contact force minimization. Considering the difficulty of accurately modeling contact dynamics in simulation (and the resulting sim-to-real gap), these results are particularly significant, demonstrating that RL-SCT enables safe and efficient learning directly on real robots while minimizing interaction forces.

While RL-SCT is primarily tailored for tasks involving objects with known constraints, it is less suited for motor skills such as juggling, ball-in-cup, or intricate assembly tasks. However, subsequent methods in this work address intricate assembly tasks. Although designing SCTs can be cumbersome, our results demonstrate that they enable safer and faster learning compared to traditional reward shaping, which often demands extensive fine-tuning and may involve unsafe interactions. Moreover, the potential to reuse SCTs across related tasks presents a promising direction for reducing design overhead in future applications.

6.1.2 *Guiding RL with Demonstrations*

In Chapter 4, we presented a novel movement primitive representation called Linearly Constrained Null-space Kernelized Movement Primitives (LC-NS-KMP), which learns an initial base policy for robot manipulation from demonstrations. LC-NS-KMP enables modifications to the base policy through a soft null-space projector and null-space actions, in accordance with the variance observed in the demonstrations. Specifically, it allows greater modifications in regions of motion with higher variance compared to those regions where the variance is low. Finally, LC-NS-KMP provides a mechanism for enforcing linear inequality constraints on the robot's state, ensuring that the modifications respect the imposed constraints.

We leveraged this movement primitive representation to develop a novel constrained and guided RL method, Kernelized Guided Reinforcement Learning (KGRL). KGRL enables learning an RL policy

that leverages guidance from the base policy extracted from demonstrations, imposes safety constraints in the form of linear inequalities, and performs uncertainty-aware, state-dependent exploration using the null-space projector and null-space actions obtained from the RL policy. KGRL effectively unifies learning from demonstrations, constrained probabilistic prediction, and RL with uncertainty-aware state-dependent exploration. Furthermore, KGRL enhances knowledge extraction over RL-SCT by extracting task knowledge directly from user demonstrations, whereas in RL-SCT, an expert user handcrafts the task knowledge.

As discussed in Section 4.4.2, we evaluated our approach to highlight the effectiveness of KGRL in learning challenging manipulation tasks involving complex contacts directly on a real robot. By integrating state-dependent guided exploration and linear inequality constraints, KGRL facilitated efficient learning and enhanced the robot's operational safety. Our approach enabled the robot to master BNC connector insertion and locking in under 45 episodes, significantly reducing learning time and effort. Additionally, the observed reduction in interaction forces with the environment indicates a pathway toward long-term reliability and safety in robot manipulation. Finally, the mechanism for enforcing constraints and the ability to introduce variance-informed modifications in the base policy through null-space actions make LC-NS-KMP a highly desirable movement primitive framework for robotics.

6.1.3 *Smooth Exploration Strategy for RL*

In Chapter 5, we presented sKGRL, an extension of KGRL that enables smooth, principled exploration in RL while preserving KGRL's state-dependent, uncertainty-aware exploration and its ability to enforce linear inequality constraints. sKGRL ensures continuity in the trajectories and reduces jerky motions by prepending low-variance robot state history to the reference trajectory distribution, yielding jerk magnitudes orders of magnitude lower than KGRL. In simulation, as shown in Section 5.4.1, sKGRL outperformed KGRL in per-episode rewards through lower action costs and more stable learning, despite requiring slightly more episodes to reach 100% success due to its conservative exploration. On the real robot, as shown in Section 5.4.2, sKGRL matched KGRL's 100% success convergence in 45 episodes while producing smoother exploration. Although post-convergence optimization of force interaction costs and episode lengths was slightly lower than KGRL, sKGRL enhanced safety by substantially reducing jerks during learning.

6.1.4 *Simplified Reward Function Design*

All our frameworks provide an important additional benefit in the form of simplified reward function design. In classical RL, the reward function encodes the intended behavior that the agent should learn and incorporates costs for constraint violations. In our frameworks, constraints are enforced explicitly, eliminating the need for implicit representation in the reward function and reducing the efforts required to fine-tune a complex reward function. In KGRL, reward specification is further simplified to a sparse reward with only the action and force interaction costs. This is enabled by the guidance provided by the base policy and the incorporation of uncertainty-aware, state-dependent exploration.

6.2 OUTLOOK

6.2.1 *Use of Visual Language Models to Enhance Task Knowledge and Constraints*

With their proven capabilities in making RL faster and safer, our frameworks—RL-SCT, KGRL, and sKGRL—can be further enhanced through recent advances in Visual Language Models (VLMs) [145]. For example, RL-SCT can leverage VLMs to dynamically parameterize SCTs in environments where external agents change the state or phase of the task. Both RL-SCT and KGRL can also utilize VLMs to infer hard constraints specific to a class of objects, while constraints in KGRL and sKGRL may alternatively be derived directly from demonstrations by observing the range of motion across degrees of freedom. In sKGRL, VLMs can be deployed to adjust the smoothness parameter P, enabling reactive exploration in regions where higher responsiveness is required by identifying task phases that demand greater reactiveness. Alternatively, a higher control command update rate can be used.

6.2.2 *Extended State Representation in KGRL and sKGRL*

In KGRL and sKGRL, the RL policy generates null-space actions to modify the base policy by observing the robot's state, primarily including end-effector pose, end-effector wrench, and time, as discussed in Sections 4.4.2 and 5.4.2. Observations could also incorporate images or complex state representations obtained from visual feedback, augmented with pose and wrench information. KGRL and sKGRL inherently support such rich observations. Future work could investigate how incorporating these observations affects the ability of these methods to handle greater environmental variability and its impact on data efficiency.

6.2.3 *Policy Distillation with KGRL*

It would also be interesting to leverage the KGRL policy for policy distillation to learn a simpler deep RL policy, thereby reducing computational complexity while retaining the advantages of guided, constrained exploration. This can be approached in two ways. Firstly, the learned KGRL policy can serve as a teacher to distill a student policy that directly mimics KGRL's behavior. By training the student policy to reproduce the teacher's actions, we can eliminate the additional mathematical overhead of KGRL during deployment. This approach allows the student policy to inherit the structured guidance of KGRL without requiring explicit null-space computations during execution. Secondly, an off-policy RL algorithm can be deployed in parallel, with its experience buffer populated using the same state transitions as KGRL, but substituting the null-space actions with the equivalent task-space actions derived from the KGRL policy. This enables the student policy to operate directly in task space while observing the same RL state as KGRL, effectively distilling both the guidance and safety characteristics of the teacher policy. In this setup, the student policy benefits from KGRL's constrained exploration and safe learning behavior, while being a computationally lightweight generic policy that can be further refined to address potential limitations arising from incomplete or suboptimal demonstrations.

Such policy distillation allows deployment of a more efficient RL agent in real-time robotic applications where computational resources are limited.

7

OWN PUBLICATIONS

[1] Abhishek Padalkar, Gabriel Quere, Franz Steinmetz, Antonin Raffin, Matthias Nieuwenhuisen, João Silvério, and Freek Stulp. "Guiding Reinforcement Learning with Shared Control Templates." In: *40th IEEE International Conference on Robotics and Automation, ICRA 2023*. IEEE. 2023, pp. 11531–11537.

[2] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. "Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0." In: *2024 IEEE International Conference on Robotics and Automation, ICRA 2024*. IEEE. 2024, pp. 6892–6903.

[3] Abhishek Padalkar, Gabriel Quere, Antonin Raffin, João Silvério, and Freek Stulp. "Guiding real-world reinforcement learning for in-contact manipulation tasks with Shared Control Templates." In: *Autonomous Robots* 48.4 (2024), p. 12.

[4] Abhishek Padalkar, Freek Stulp, Gerhard Neumann, and João Silvério. "Towards safe and efficient learning in the wild: guiding RL with constrained uncertainty-aware movement primitives." In: *IEEE Robotics and Automation Letters* (2025).

[5] Abhishek Padalkar, Freek Stulp, Gerhard Neumann, and João Silvério. "Kernelized Guided Reinforcement Learning with Smooth Exploration." Manuscript under review at the IEEE International Conference on Robotics and Automation (ICRA), 2026. -.

BIBLIOGRAPHY

[1] Gabriel Quere, Annette Hagengruber, Maged Iskandar, Samuel Bustamante, Daniel Leidner, Freek Stulp, and Jörn Vogel. "Shared Control Templates for Assistive Robotics." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 1956–1962.

[2] Maged Iskandar, Christian Ott, Oliver Eiberger, Manuel Keppler, Alin Albu-Schäffer, and Alexander Dietrich. "Joint-level control of the DLR lightweight robot SARA." In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 8903–8910.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602* (2013).

[4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587 (2016), pp. 484–489.

[5] Miguel Vasco, Takuma Seno, Kenta Kawamoto, Kaushik Subramanian, Peter R Wurman, and Peter Stone. "A super-human vision-based reinforcement learning agent for autonomous racing in gran turismo." In: *arXiv preprint arXiv:2406.12563* (2024).

[6] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of go without human knowledge." In: *nature* 550.7676 (2017), pp. 354–359.

[7] Ilija Radosavovic, Tete Xiao, Bike Zhang, Trevor Darrell, Jitendra Malik, and Koushil Sreenath. "Real-world humanoid locomotion with reinforcement learning." In: *Science Robotics* 9.89 (2024), eadi9579.

[8] Kai Ploeger, Michael Lutter, and Jan Peters. "High acceleration reinforcement learning for real-world juggling with binary rewards." In: *arXiv preprint arXiv:2010.13483* (2020).

[9] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. "Learning dexterous in-hand manipulation." In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[10] Devin Schwab, Tobias Springenberg, Murilo F Martins, Thomas Lampe, Michael Neunert, Abbas Abdolmaleki, Tim Hertweck, Roland Hafner, Francesco Nori, and Martin Riedmiller. "Simultaneously learning vision and feature-based control policies for real-world ball-in-a-cup." In: *arXiv preprint arXiv:1902.04706* (2019).

[11] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." In: *The International journal of robotics research* 37.4-5 (2018), pp. 421–436.

[12] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning quadrupedal locomotion over challenging terrain." In: *Science robotics* 5.47 (2020), eabc5986.

[13] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. "Recent advances in robot learning from demonstration." In: *Annual review of control, robotics, and autonomous systems* 3 (2020), pp. 297–330.

[14] 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.

[15] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. "Probabilistic movement primitives." In: *Advances in neural information processing systems* 26 (2013).

[16] Yanlong Huang, Leonel Rozo, Joao Silvério, and Darwin G Caldwell. "Kernelized movement primitives." In: *IJRR* 38.7 (2019), pp. 833–852.

[17] Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. "A review on reinforcement learning for contact-rich robotic manipulation tasks." In: *Robotics and Computer-Integrated Manufacturing* 81 (2023), p. 102517.

[18] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. "Deep reinforcement learning for robotics: A survey of real-world successes." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 27. 2025, pp. 28694–28698.

[19] Thomas Rückstieß, Martin Felder, and Jürgen Schmidhuber. "State-dependent exploration for policy gradient methods." In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2008, pp. 234–249.

[20] Antonin Raffin, Jens Kober, and Freek Stulp. "Smooth exploration for robotic reinforcement learning." In: *Conference on robot learning*. PMLR. 2022, pp. 1634–1644.

[21] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. "Crossing the Reality Gap: a Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning." In: *IEEE Access* 9 (2021), pp. 153171–153187.

[22] Jaemin Yoon, Bukun Son, and Dongjun Lee. "Comparative study of physics engines for robot simulation with mechanical interaction." In: *Applied Sciences* 13.2 (2023), p. 680.

[23] Yashraj Narang, Kier Storey, Iretiayo Akinola, Miles Macklin, Philipp Reist, Lukasz Wawrzyniak, Yunrong Guo, Adam Moravanszky, Gavriel State, Michelle Lu, et al. "Factory: Fast contact for robotic assembly." In: *arXiv preprint arXiv:2205.03532* (2022).

[24] Michael Noseworthy, Bingjie Tang, Bowen Wen, Ankur Handa, Chad Kessens, Nicholas Roy, Dieter Fox, Fabio Ramos, Yashraj Narang, and Iretiayo Akinola. "Forge: Force-guided exploration for robust contact-rich manipulation under uncertainty." In: *IEEE Robotics and Automation Letters* (2025).

[25] James J. Gibson. *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin Harcourt (HMH), 1979, p. 127.

[26] Daniel Leidner, Christoph Borst, and Gerd Hirzinger. "Things are made for what they are: Solving manipulation tasks by using functional object classes." In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE. 2012, pp. 429–435.

[27] Khimya Khetarpal, Zafarali Ahmed, Gheorghe Comanici, David Abel, and Doina Precup. "What can I do here? a theory of affordances in reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5243–5253.

[28] Karl Pertsch, Youngwoon Lee, and Joseph Lim. "Accelerating reinforcement learning with learned skill priors." In: *Conference on robot learning*. PMLR. 2021, pp. 188–204.

[29] Fernando Fernández and Manuela Veloso. "Probabilistic policy reuse in a reinforcement learning agent." In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. 2006, pp. 720–727.

[30] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. "Constrained policy optimization." In: *International conference on machine learning*. PMLR. 2017, pp. 22–31.

[31] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3387–3395.

[32] Lukas Brunke, Melissa Greeff, Adam W Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P Schoellig. "Safe learning in robotics: From learning-based control to safe reinforcement learning." In: *Annual Review of Control, Robotics, and Autonomous Systems* 5.1 (2022), pp. 411–444.

[33] Abhishek Padalkar, Gabriel Quere, Antonin Raffin, João Silvério, and Freek Stulp. "Guiding real-world reinforcement learning for in-contact manipulation tasks with Shared Control Templates." In: *Autonomous Robots* 48.4 (2024), p. 12.

[34] Abhishek Padalkar, Freek Stulp, Gerhard Neumann, and João Silvério. "Towards safe and efficient learning in the wild: guiding RL with constrained uncertainty-aware movement primitives." In: *IEEE Robotics and Automation Letters* (2025).

[35] Kenneth Kimble, Karl Van Wyk, Joe Falco, Elena Messina, Yu Sun, Mizuho Shibata, Wataru Uemura, and Yasuyoshi Yokokohji. "Benchmarking protocols for evaluating small parts robotic assembly systems." In: *RA-L* 5.2 (2020), pp. 883–889.

[36] Abhishek Padalkar, Freek Stulp, Gerhard Neumann, and João Silvério. "Kernelized Guided Reinforcement Learning with Smooth Exploration." Manuscript under review at the IEEE International Conference on Robotics and Automation (ICRA), 2026. -.

[37] Solomon Kullback and Richard A Leibler. "On information and sufficiency." In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[38] Roland Schwan, Yuning Jiang, Daniel Kuhn, and Colin N. Jones. "PIQP: A Proximal Interior-Point Quadratic Programming Solver." In: *2023 62nd IEEE Conference on Decision and Control (CDC)*. 2023, pp. 1088–1093.

[39] Sylvain Calinon. *Robot programming by demonstration*. EPFL Press, 2009.

[40] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. "A survey of robot learning from demonstration." In: *RAS* 57.5 (2009), pp. 469–483.

[41] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15.

Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 627–635.

[42] Karl Pertsch, Youngwoon Lee, Yue Wu, and Joseph J Lim. "Demonstration-Guided Reinforcement Learning with Learned Skills." In: Proceedings of Machine Learning Research 164 (Aug. 2022). Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann, pp. 729–739.

[43] Jonathan Ho and Stefano Ermon. "Generative adversarial imitation learning." In: *Advances in neural information processing systems* 29 (2016).

[44] Pieter Abbeel and Andrew Y Ng. "Apprenticeship learning via inverse reinforcement learning." In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.

[45] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. "Maximum entropy inverse reinforcement learning." In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

[46] Sylvain Calinon, Florent Guenter, and Aude Billard. "On learning, representing, and generalizing a task in a humanoid robot." In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 37.2 (2007), pp. 286–298.

[47] Sylvain Calinon. "Mixture models for the analysis, edition, and synthesis of continuous time series." In: *Mixture Models and Applications*. Springer, 2019, pp. 39–57.

[48] Sylvain Calinon. "A tutorial on task-parameterized movement learning and retrieval." In: *Intelligent service robotics* 9.1 (2016), pp. 1–29.

[49] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. "Using probabilistic movement primitives in robotics." In: *Autonomous Robots* 42.3 (2018), pp. 529–551.

[50] João Silvério and Yanlong Huang. "A non-parametric skill representation with soft null space projectors for fast generalization." In: *ICRA*. 2023, pp. 2988–2994.

[51] João Silvério, Yanlong Huang, Fares J Abu-Dakka, Leonel Rozo, and Darwin G Caldwell. "Uncertainty-aware imitation learning using kernelized movement primitives." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 90–97.

[52] Yanlong Huang and Darwin G Caldwell. "A linearly constrained nonparametric framework for imitation learning." In: *ICRA*. 2020, pp. 4400–4406.

[53] Thibaut Kulak, Hakan Girgin, Jean-Marc Odobez, and Sylvain Calinon. "Active learning of Bayesian probabilistic movement primitives." In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2163–2170.

[54] Matthias Seeger. "Gaussian processes for machine learning." In: *International journal of neural systems* 14.02 (2004), pp. 69–106.

[55] Martijn JA Zeestraten, Ioannis Havoutis, and Sylvain Calinon. "Programming by demonstration for shared control with an application in teleoperation." In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1848–1855.

[56] Dylan P Losey, Craig G McDonald, Edoardo Battaglia, and Marcia K O'Malley. "A review of intent detection, arbitration, and communication aspects of shared control for physical human–robot interaction." In: *Applied Mechanics Reviews* 70.1 (2018), p. 010804.

[57] Anca D Dragan and Siddhartha S Srinivasa. "A policy-blending formalism for shared control." In: *The International Journal of Robotics Research* 32.7 (2013), pp. 790–805.

[58] Jörn Vogel, Annette Hagengruber, Maged Iskandar, Gabriel Quere, Ulrike Leipscher, Samuel Bustamante Gomez, Alexander Dietrich, Hannes Höppner, Daniel Leidner, and Alin Olimpiu Albu-Schäffer. "EDAN-An EMG-controlled Daily Assistant To Help People With Physical Disabilities." In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020*. 2020.

[59] Stuart A Bowyer, Brian L Davies, and Ferdinando Rodriguez y Baena. "Active constraints/virtual fixtures: A survey." In: *IEEE Transactions on Robotics* 30.1 (2013), pp. 138–157.

[60] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[61] Christopher JCH Watkins and Peter Dayan. "Q-learning." In: *Machine learning* 8.3 (1992), pp. 279–292.

[62] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[63] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[64] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017).

[65] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[66] Hado Hasselt. "Double Q-learning." In: *Advances in neural information processing systems* 23 (2010).

[67] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5556–5566.

[68] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *arXiv preprint arXiv:1509.02971* (2015).

[69] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. "Soft actor-critic algorithms and applications." In: *arXiv preprint arXiv:1812.05905* (2018).

[70] Jonas Buchli, Freek Stulp, Evangelos Theodorou, and Stefan Schaal. "Learning Variable Impedance Control." In: *International Journal of Robotics Research* 30.7 (2011), pp. 820–833.

[71] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. "Learning force control policies for compliant manipulation." In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 4639–4644.

[72] Shengyi Huang and Santiago Ontañón. "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms." In: *CoRR* abs/2006.14171 (2020).

[73] Anssi Kanervisto, Christian Scheller, and Ville Hautamäki. "Action Space Shaping in Deep Reinforcement Learning." In: *CoRR* abs/2004.00980 (2020).

[74] Andrew Taylor, Andrew Singletary, Yisong Yue, and Aaron Ames. "Learning for safety-critical control with control barrier functions." In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 708–717.

[75] Puze Liu, Davide Tateo, Haitham Bou Ammar, and Jan Peters. "Robot reinforcement learning on the constraint manifold." In: *Conference on Robot Learning*. PMLR. 2022, pp. 1357–1366.

[76] Francisco Cruz, Sven Magg, Cornelius Weber, and Stefan Wermter. "Improving reinforcement learning with interactive feedback and affordances." In: *4th International Conference on Development and Learning and on Epigenetic Robotics*. IEEE. 2014, pp. 165–170.

[77] Francisco Cruz, Sven Magg, Cornelius Weber, and Stefan Wermter. "Training agents with interactive reinforcement learning and contextual affordances." In: *IEEE Transactions on Cognitive and Developmental Systems* 8.4 (2016), pp. 271–284.

[78] Francisco Cruz, German I Parisi, and Stefan Wermter. "Multi-modal feedback for affordance-driven interactive reinforcement learning." In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[79] Kevin Sebastian Luck, Gerhard Neumann, Erik Berger, Jan Peters, and Heni Ben Amor. "Latent space policy search for robotics." In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 1434–1440.

[80] J Zico Kolter, Andrew Y Ng, et al. "Learning omnidirectional path following using dimensionality reduction." In: *Robotics: Science and Systems*. 2007, pp. 27–30.

[81] Abhishek Padalkar, Matthias Nieuwenhuisen, Sven Schneider, and Dirk Schulz. "Learning to Close the Gap: Combining Task Frame Formalism and Reinforcement Learning for Compliant Vegetable Cutting." In: *ICINCO*. 2020, pp. 221–231.

[82] Abhishek Padalkar, Matthias Nieuwenhuisen, Dirk Schulz, and Freek Stulp. "Closing the gap: Combining task specification and reinforcement learning for compliant vegetable cutting." In: *International conference on informatics in control, automation and robotics*. Springer. 2020, pp. 187–206.

[83] Matthew T Mason. "Compliance and force control for computer controlled manipulators." In: *IEEE Transactions on Systems, Man, and Cybernetics* 11.6 (1981), pp. 418–432.

[84] Herman Bruyninckx and Joris De Schutter. "Specification of force-controlled actions in the "task frame formalism" - a synthesis." In: *IEEE transactions on robotics and automation* 12.4 (1996), pp. 581–589.

[85] René Felix Reinhart and Jochen Jakob Steil. "Efficient policy search in low-dimensional embedding spaces by generalizing motion primitives with a parameterized skill memory." In: *Autonomous Robots* 38 (2015), pp. 331–348.

[86] Zhanpeng He and Matei Ciocarlie. "Discovering synergies for robot manipulation with multi-task reinforcement learning." In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 2714–2721.

[87] Anca D Dragan and Siddhartha S Srinivasa. "A policy-blending formalism for shared control." en. In: *The International Journal of Robotics Research* 32.7 (June 2013), pp. 790–805. ISSN: 0278-3649, 1741-3176.

[88] Sebastian Bitzer, Matthew Howard, and Sethu Vijayakumar. "Using dimensionality reduction to exploit constraints in reinforcement learning." In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 3219–3225.

[89] Samuele Tosatto, Georgia Chalvatzaki, and Jan Peters. "Contextual latent-movements off-policy optimization for robotic manipulation skills." In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 10815–10821.

[90] William Curran, Tim Brys, David Aha, Matthew Taylor, and William D Smart. "Dimensionality reduced reinforcement learning for assistive robots." In: *2016 AAAI Fall Symposium Series*. 2016.

[91] Simone Parisi, Simon Ramstedt, and Jan Peters. "Goal-driven dimensionality reduction for reinforcement learning." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 4634–4639.

[92] Antonin Raffin, Ashley Hill, Kalifou René Traoré, Timothée Lesort, Natalia Díaz-Rodríguez, and David Filliat. "Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics." In: *SPiRL Workshop ICLR* (2019).

[93] 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 (Aug. 1999), pp. 181–211. ISSN: 0004-3702.

[94] Martin Stolle and Doina Precup. "Learning options in reinforcement learning." In: *Abstraction, Reformulation, and Approximation: 5th International Symposium, SARA 2002 Kananaskis, Alberta, Canada August 2–4, 2002 Proceedings 5*. Springer. 2002, pp. 212–223.

[95] Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. "Learning sequential motor tasks." In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 2626–2632.

[96] Oliver Kroemer, Christian Daniel, Gerhard Neumann, Herke Van Hoof, and Jan Peters. "Towards learning hierarchical skills for multi-phase manipulation tasks." In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 1503–1510.

[97] Sanjay Krishnan, Animesh Garg, Richard Liaw, Brijen Thananjeyan, Lauren Miller, Florian T Pokorny, and Ken Goldberg. "SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards." In: *The international journal of robotics research* 38.2-3 (2019), pp. 126–145.

[98] Dorothea Koert, Maximilian Kircher, Vildan Salikutluk, Carlo D'Eramo, and Jan Peters. "Multi-channel interactive reinforcement learning for sequential tasks." In: *Frontiers in Robotics and AI* 7 (2020), p. 97.

[99] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping." In: *ICML*. Vol. 99. 1999, pp. 278–287.

[100] Tony Z Zhao, Jianlan Luo, Oleg Sushkov, Rugile Pevceviciute, Nicolas Heess, Jon Scholz, Stefan Schaal, and Sergey Levine. "Offline meta-reinforcement learning for industrial insertion." In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 6386–6393.

[101] Aleksandra Anna Apolinarska, Matteo Pacher, Hui Li, Nicholas Cote, Rafael Pastrana, Fabio Gramazio, and Matthias Kohler. "Robotic assembly of timber joints using reinforcement learning." In: *Automation in Construction* 125 (2021), p. 103569.

[102] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothörl, Todd Hester, and Jon Scholz. "A practical approach to insertion with variable socket position using deep reinforcement learning." In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 754–760.

[103] Cristian Camilo Beltran-Hernandez, Damien Petit, Ixchel Georgina Ramirez-Alpizar, Takayuki Nishi, Shinichi Kikuchi, Takamitsu Matsubara, and Kensuke Harada. "Learning force control for contact-rich manipulation tasks with rigid position-controlled robots." In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5709–5716.

[104] Michelle A Lee, Carlos Florensa, Jonathan Tremblay, Nathan Ratliff, Animesh Garg, Fabio Ramos, and Dieter Fox. "Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning." In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 7505–7512.

[105] Shir Kozlovsky, Elad Newman, and Miriam Zacksenhouse. "Reinforcement learning of impedance policies for peg-in-hole tasks: Role of asymmetric matrices." In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10898–10905.

[106] Todor Davchev, Kevin Sebastian Luck, Michael Burke, Franziska Meier, Stefan Schaal, and Subramanian Ramamoorthy. "Residual learning from demonstration: Adapting dmps for contact-rich manipulation." In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4488–4495.

[107] Yong-Geon Kim, Minwoo Na, and Jae-Bok Song. "Reinforcement learning-based sim-to-real impedance parameter tuning for robotic assembly." In: *2021 21st International Conference on Control, Automation and Systems (ICCAS)*. IEEE. 2021, pp. 833–836.

[108] Jianlan Luo, Eugen Solowjow, Chengtao Wen, Juan Aparicio Ojea, Alice M Agogino, Aviv Tamar, and Pieter Abbeel. "Reinforcement learning on variable impedance controller for high-precision robotic assembly." In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3080–3087.

[109] Niladri S. Chatterji, Aldo Pacchiano, Peter L. Bartlett, and Michael I. Jordan. *On the Theory of Reinforcement Learning with Once-per-Episode Feedback*. 2022.

[110] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2021.

[111] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. "Stable-Baselines3: Reliable Reinforcement Learning Implementations." In: *Journal of Machine Learning Research* 22.268 (2021). Ed. by Andreas Mueller, pp. 1–8.

[112] Lukas Jaeger et al. "How the CYBATHLON Competition Has Advanced Assistive Technologies." In: *Annual Review of Control, Robotics, and Autonomous Systems* 6.1 (2023), pp. 447–476.

[113] Jörn Vogel, Annette Hagengruber, Gabriel Quere, Samuel Bustamante, Jianxian Feng, Sebastian Jung, Elle Miller, Maged Iskander, and Hanna Riesch. *Mattias and EDAN winning at CYBATHLON Challenges March 2023*. 2023.

[114] Gabriel Quere, Samuel Bustamante, Annette Hagengruber, Jörn Vogel, Franz Steinmetz, and Freek Stulp. "Learning and interactive design of shared control templates." In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1887–1894.

[115] Gabriel Quere, Freek Stulp, David Filliat, and Joao Silvério. "A probabilistic approach for learning and adapting shared control skills with the human in the loop." In: *International Conference on Robotics and Automation (ICRA)*. 2024.

[116] Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell. "Kernelized Movement Primitives." In: *International Journal of Robotics Research* 38.7 (2019), pp. 833–852.

[117] João Silvério and Yanlong Huang. "A Non-parametric Skill Representation with Soft Null Space Projectors for Fast Generalization." In: *ICRA*. 2023, pp. 2988–2994.

[118] Julian Eßer, Nicolas Bach, Christian Jestel, Oliver Urbann, and Sören Kerner. "Guided Reinforcement Learning: A Review and Evaluation for Efficient and Effective Real-World Robotics [Survey]." In: *RAM* 30.2 (2023), pp. 67–85.

[119] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards." In: *arXiv preprint arXiv:1707.08817* (2017).

[120] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Overcoming exploration in reinforcement learning with demonstrations." In: *ICRA*. 2018, pp. 6292–6299.

[121] Justin Fu, Katie Luo, and Sergey Levine. "Learning robust rewards with adversarial inverse reinforcement learning." In: *arXiv preprint arXiv:1710.11248* (2017).

[122] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations." In: *arXiv preprint arXiv:1709.10087* (2017).

[123] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. "Deep q-learning from demonstrations." In: *AAAI*. Vol. 32. 2018.

[124] Minttu Alakuijala, Gabriel Dulac-Arnold, Julien Mairal, Jean Ponce, and Cordelia Schmid. "Residual reinforcement learning from demonstrations." In: *arXiv preprint arXiv:2106.08050* (2021).

[125] Jean-Pierre Sleiman, Mayank Mittal, and Marco Hutter. "Guided Reinforcement Learning for Robust Multi-Contact Loco-Manipulation." In: *8th Annual Conference on Robot Learning (CoRL 2024)*. 2024.

[126] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. "Optlayer-practical constrained optimization for deep reinforcement learning in the real world." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 6236–6243.

[127] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. "Safe exploration in continuous action spaces." In: *arXiv preprint arXiv:1801.08757* (2018).

[128] Yanan Sui, Alkis Gotovos, Joel Burdick, and Andreas Krause. "Safe exploration for optimization with Gaussian processes." In: *International conference on machine learning (ICML)*. 2015, pp. 997–1005.

[129] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. "Safe exploration in finite markov decision processes with gaussian processes." In: *Advances in neural information processing systems* 29 (2016).

[130] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. "Diffusion policy: Visuomotor policy learning via action diffusion." In: *The International Journal of Robotics Research (IJRR)* (2024).

[131] Zhengbang Zhu, Hanye Zhao, Haoran He, Yichao Zhong, Shenyu Zhang, Haoquan Guo, Tingting Chen, and Weinan Zhang. "Diffusion models for reinforcement learning: A survey." In: *arXiv preprint arXiv:2311.01223* (2023).

[132] Yinan Zheng, Jianxiong Li, Dongjie Yu, Yujie Yang, Shengbo Eben Li, Xianyuan Zhan, and Jingjing Liu. "Safe Offline Reinforcement Learning with Feasibility-Guided Diffusion Model." In: *The Twelfth International Conference on Learning Representations*.

[133] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. "Natural evolution strategies." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 949–980.

[134] Maximilian Hüttenrauch and Gerhard Neumann. "Robust black-box optimization for stochastic search and episodic reinforcement learning." In: *Journal of Machine Learning Research* 25.153 (2024), pp. 1–44.

[135] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. "Policy gradients with parameter-based exploration for control." In: *International Conference on Artificial Neural Networks*. Springer. 2008, pp. 387–396.

[136] Thomas Rückstiess, Frank Sehnke, Tom Schaul, Daan Wierstra, Yi Sun, and Jürgen Schmidhuber. "Exploring parameter space in reinforcement learning." In: *Paladyn* 1.1 (2010), pp. 14–24.

[137] Fabian Otto, Hongyi Zhou, Onur Celik, Ge Li, Rudolf Lioutikov, and Gerhard Neumann. "MP3: Movement Primitive-Based (Re-) Planning Policy." In: *CoRR* (2023).

[138] Fabian Otto, Onur Celik, Hongyi Zhou, Hanna Ziesche, Vien Anh Ngo, and Gerhard Neumann. "Deep black-box reinforcement learning with movement primitives." In: *Conference on Robot Learning*. PMLR. 2023, pp. 1244–1265.

[139] Ge Li, Hongyi Zhou, Dominik Roth, Serge Thilges, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. "Open the Black Box: Step-based Policy Updates for Temporally-Correlated Episodic Reinforcement Learning." In: *The Twelfth International Conference on Learning Representations*.

[140] Alberto Silvio Chiappa, Alessandro Marin Vargas, Ann Huang, and Alexander Mathis. "Latent exploration for reinforcement learning." In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 56508–56530.

[141] Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, and Georg Martius. "Pink noise is all you need: Colored noise exploration in deep reinforcement learning." In: *The Eleventh International Conference on Learning Representations*. 2023.

[142] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. "Learning to walk via deep reinforcement learning." In: *arXiv preprint arXiv:1812.11103* (2018).

[143] Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. "Learning to walk in the real world with minimal human effort." In: *arXiv preprint arXiv:2002.08550* (2020).

[144] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. "Continuous-discrete reinforcement learning for hybrid control in robotics." In: *Conference on Robot Learning*. PMLR. 2020, pp. 735–751.

[145] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. "Llama: Open and efficient foundation language models." In: *arXiv preprint arXiv:2302.13971* (2023).

A

APPENDIX

A.1 DERIVATION OF LC-NS-KMP

This appendix provides more details on the derivation of Linearly Constrained Null-Space Kernelized Movement Primitives (LC-NS-KMP).

We start with the Lagrangian of the optimization problem in Equation (4.6) of the Chapter 4,

$$\begin{aligned}\mathcal{L}(\boldsymbol{\mu}_w, \boldsymbol{\alpha}) = &\sum_{n=1}^{N} \frac{1}{2}(\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n)^\top \hat{\boldsymbol{\Sigma}}_n^{-1}(\boldsymbol{\Theta}^\top(\boldsymbol{\xi}_n^{\mathcal{I}})\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_n) \\ &+ \frac{1}{2}\lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w + \frac{1}{2}\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_{\boldsymbol{w}})^\top(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_{\boldsymbol{w}}) \\ &- \sum_{n=1}^{N}\sum_{f=1}^{F} \alpha_{n,f}(\boldsymbol{g}_{n,f}^\top \boldsymbol{\Theta}(\boldsymbol{\xi}_{\boldsymbol{n}}^{\mathcal{I}})^\top \boldsymbol{\mu}_{\boldsymbol{w}} - c_{n,f}).\end{aligned} \tag{A.1}$$

Please refer to Section 2.2.7 and Section 4.3.4 for background of KGRL and notations used in this appendix.

Equation (A.1) can be re-written using matrix notation as

$$\begin{aligned}\mathcal{L}(\boldsymbol{\mu}_w, \boldsymbol{\alpha}) = &\frac{1}{2}(\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\mu}) \\ &+ \frac{1}{2}\lambda \boldsymbol{\mu}_w^\top \boldsymbol{\mu}_w + \frac{1}{2}\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_{\boldsymbol{w}})^\top(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_{\boldsymbol{w}}) \\ &- \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\alpha}^\top \bar{\boldsymbol{C}},\end{aligned} \tag{A.2}$$

where,

$$\begin{aligned}&\boldsymbol{\Phi} = [\boldsymbol{\Theta}(\boldsymbol{\xi}_1^{\mathcal{I}})\ \boldsymbol{\Theta}(\boldsymbol{\xi}_2^{\mathcal{I}})\ \boldsymbol{\Theta}(\boldsymbol{\xi}_3^{\mathcal{I}})\ \ldots\ \boldsymbol{\Theta}(\boldsymbol{\xi}_n^{\mathcal{I}})], \\ &\boldsymbol{\Sigma} = \text{blockdiag}(\hat{\boldsymbol{\Sigma}}_1,\ \hat{\boldsymbol{\Sigma}}_2, \hat{\boldsymbol{\Sigma}}_3,\ \ldots,\ \hat{\boldsymbol{\Sigma}}_n), \\ &\boldsymbol{\mu} = [\hat{\boldsymbol{\mu}}_1^\top,\ \hat{\boldsymbol{\mu}}_2^\top,\ \hat{\boldsymbol{\mu}}_3^\top,\ \ldots\ \hat{\boldsymbol{\mu}}_n^\top]^\top, \\ &\boldsymbol{G}_n = [\boldsymbol{g}_{n,1}\ \boldsymbol{g}_{n,2}\ \boldsymbol{g}_{n,3}\ \ldots\ \boldsymbol{g}_{n,F}], \quad \forall n \in \{1, 2, 3, \ldots, N\}, \quad \text{and} \\ &\bar{\boldsymbol{G}} = \text{blockdiag}(\boldsymbol{G}_1,\ \boldsymbol{G}_2,\ \boldsymbol{G}_3,\ \ldots\ \boldsymbol{G}_n), \\ &\boldsymbol{\alpha} = [\alpha_{1,1},\ \alpha_{1,2},\ \ldots,\ \alpha_{1,F},\ \ldots,\ \alpha_{N,1},\ \ldots,\ \alpha_{N,F}], \\ &\bar{\boldsymbol{C}} = [\boldsymbol{C}_1^\top\ \boldsymbol{C}_2^\top\ \ldots\ \boldsymbol{C}_N^\top]^\top, \\ &\boldsymbol{C}_n = [c_{n,1}\ c_{n,2}\ \ldots\ c_{n,F}]^\top, \forall n \in \{1, 2, \ldots, N\}.\end{aligned}$$

Differentiating Equation (A.2) with respect to $\boldsymbol{\mu}_w$, we get[1]

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{\mu}_w, \boldsymbol{\alpha})}{\partial \boldsymbol{\mu}_w} &= \frac{1}{2}(2\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\mu})) + \frac{1}{2}(2\lambda \boldsymbol{\mu}_w) \\ &\quad + \frac{1}{2}(2\beta(\boldsymbol{\mu}_w - \hat{\boldsymbol{\mu}}_w)) - \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top \\ &= \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + (\lambda + \beta)\boldsymbol{\mu}_w \\ &\quad - \beta \hat{\boldsymbol{\mu}}_w - \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top \\ &= \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w - \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \hbar \boldsymbol{\mu}_w - \beta \hat{\boldsymbol{\mu}}_w - \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\ &= (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top + \hbar \boldsymbol{I})\boldsymbol{\mu}_w - (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \beta \hat{\boldsymbol{\mu}}_w + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}),\end{aligned} \tag{A.3}$$

where, $\hbar = \lambda + \beta$.

Rewriting $\boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha} = \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top$ is possible as $\boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top$ is a scalar.

Setting the partial derivative $\frac{\partial \mathcal{L}(\boldsymbol{\mu}_w, \boldsymbol{\alpha})}{\partial \boldsymbol{\mu}_w} = 0$, we get

$$0 = (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top + \hbar \boldsymbol{I})\boldsymbol{\mu}_w - (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \beta \hat{\boldsymbol{\mu}}_w + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}),$$

therefore,

$$(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top + \hbar \boldsymbol{I})\boldsymbol{\mu}_w^* = (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \beta \hat{\boldsymbol{\mu}}_w + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}),$$

and hence,

$$\boldsymbol{\mu}_w^* = (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top + \hbar \boldsymbol{I})^{-1}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \beta \hat{\boldsymbol{\mu}}_w + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) \tag{A.4}$$

Using Woodbury identity[2] to simplify the first term of Equation (A.4), we get

$$\begin{aligned}(\hbar \boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1} &= \frac{1}{\hbar}\boldsymbol{I} - \frac{1}{\hbar}\boldsymbol{I}\boldsymbol{\Phi}(\boldsymbol{\Sigma} + \boldsymbol{\Phi}^\top \frac{\mathbf{1}}{\hbar}\boldsymbol{I}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \frac{1}{\hbar}\boldsymbol{I} \\ &= \frac{1}{\hbar}\boldsymbol{I} - \boldsymbol{\Phi}(\hbar\boldsymbol{\Sigma} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \frac{1}{\hbar}\boldsymbol{I} \\ &= \frac{1}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}(\hbar\boldsymbol{\Sigma} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top).\end{aligned} \tag{A.5}$$

Using another variant of Woodbury identity[3] to simplify the first term of Equation (A.4), we get

$$\begin{aligned}(\hbar \boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1}\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} &= \frac{1}{\hbar}\boldsymbol{I}\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top \frac{1}{\hbar}\boldsymbol{I}\boldsymbol{\Phi} + \boldsymbol{\Sigma})^{-1} \\ &= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \hbar\boldsymbol{\Sigma})^{-1}.\end{aligned} \tag{A.6}$$

1 Using identity $\frac{\partial}{\partial \mathbf{s}}(\mathbf{x} - \mathbf{A}\mathbf{s})^\mathsf{T}\mathbf{W}(\mathbf{x} - \mathbf{A}\mathbf{s}) = -2\mathbf{A}^\mathsf{T}\mathbf{W}(\mathbf{x} - \mathbf{A}\mathbf{s})$ for symmetric $\mathbf{W}$.

2 Woodbury identity: $(\mathbf{A} + \mathbf{C}\mathbf{B}\mathbf{C}^\mathsf{T})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{C}\left(\mathbf{B}^{-1} + \mathbf{C}^\mathsf{T}\mathbf{A}^{-1}\mathbf{C}\right)^{-1}\mathbf{C}^\mathsf{T}\mathbf{A}^{-1}$

3 Woodbury identity: if $\boldsymbol{P} \succ 0$ and $\boldsymbol{R} \succ 0$, $(\boldsymbol{P}^{-1} + \boldsymbol{B}^\top\boldsymbol{R}^{-1}\boldsymbol{B})^{-1}\boldsymbol{B}^\top\boldsymbol{R}^{-1} = \boldsymbol{P}\boldsymbol{B}^\top(\boldsymbol{B}\boldsymbol{P}\boldsymbol{B}^\top + \boldsymbol{R})^{-1}$.

Further simplifying Equation (A.4) using expressions obtained in Equations (A.5) and (A.6), we get

$$\begin{aligned}
\boldsymbol{\mu}_w^* &= (\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top + \hbar\boldsymbol{I})^{-1}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \beta\hat{\boldsymbol{\mu}}_w + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) \\
&= (\hbar\boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} + \boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) \qquad \text{(A.7)} \\
&\quad + (\hbar\boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1}\beta\hat{\boldsymbol{\mu}}_w \\
&= (\hbar\boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1}\Big(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} + \boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})\Big) \qquad \text{(A.8)} \\
&\quad + (\hbar\boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1}\beta\hat{\boldsymbol{\mu}}_w \\
&= (\hbar\boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1})(\boldsymbol{\mu} + \boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) \qquad \text{(A.9)} \\
&\quad + (\hbar\boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1}\beta\hat{\boldsymbol{\mu}}_w \\
&= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \hbar\boldsymbol{\Sigma})^{-1}(\boldsymbol{\mu} + \boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) \\
&\quad + \beta(\hbar\boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top)^{-1}\hat{\boldsymbol{\mu}}_w \\
&= \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \hbar\boldsymbol{\Sigma})^{-1}(\boldsymbol{\mu} + \boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) \\
&\quad + \frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \hbar\boldsymbol{\Sigma})^{-1}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w \\
&= \boldsymbol{\Phi}\boldsymbol{A}(\boldsymbol{\mu} + \boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) + \frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w \\
&= \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w, \qquad \text{(A.10)}
\end{aligned}$$

where $\boldsymbol{A} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \hbar\boldsymbol{\Sigma})^{-1}$.

Hence,

$$\boldsymbol{\mu}_w^* = \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w. \qquad \text{(A.11)}$$

For further simplification of the Lagrangian in Equation (A.2), we substitute optimal $\boldsymbol{\mu}_w^*$ from Equation (A.11) in Equation (A.2) as

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\alpha}) = &\frac{1}{2}(\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w^* - \boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w^* - \boldsymbol{\mu}) + \frac{1}{2}\lambda\boldsymbol{\mu}_w^{*\top}\boldsymbol{\mu}_w^* \\
&+ \frac{1}{2}\beta(\boldsymbol{\mu}_w^* - \hat{\boldsymbol{\mu}}_w)^\top(\boldsymbol{\mu}_w^* - \hat{\boldsymbol{\mu}}_w) - \boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w^* - \boldsymbol{\alpha}^\top\bar{\boldsymbol{C}}.
\end{aligned} \qquad \text{(A.12)}$$

Below, we simplify each term in Equation (A.12) separately for convenience.

We begin by expanding $\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w^* - \boldsymbol{\mu}$ as

$$\begin{aligned}
\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w^* - \boldsymbol{\mu} &= \boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}(\boldsymbol{\mu} + \boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) + \frac{\beta}{\hbar}\boldsymbol{\Phi}^\top(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w - \boldsymbol{\mu}, \\
&= (\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A} - \boldsymbol{I})\boldsymbol{\mu} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&\quad + \frac{\beta}{\hbar}\boldsymbol{\Phi}^\top(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w. \qquad \text{(A.13)}
\end{aligned}$$

Taking transpose of Equation (A.13), we get

$$\begin{aligned}
(\boldsymbol{\Phi}^\top\boldsymbol{\mu}_w^* - \boldsymbol{\mu})^\top &= \boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \boldsymbol{I}) + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi} \\
&\quad + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}
\end{aligned} \qquad \text{(A.14)}$$

$$(\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w^* - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} = \boldsymbol{\mu}^\top (\boldsymbol{A}\boldsymbol{\Phi}^\top \boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{\Sigma}^{-1} + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} \tag{A.15}$$

$$\begin{aligned}
&(\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w^* - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{\Phi}^\top \boldsymbol{\mu}_w^* - \boldsymbol{\mu}) \\
&\quad = \boldsymbol{\mu}^\top (\boldsymbol{A}\boldsymbol{\Phi}^\top \boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{\Sigma}^{-1} (\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A} - \boldsymbol{I})\boldsymbol{\mu} \\
&\qquad + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} (\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A} - \boldsymbol{I})\boldsymbol{\mu} \\
&\qquad + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} (\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A} - \boldsymbol{I})\boldsymbol{\mu} \\
&\qquad + \boldsymbol{\mu}^\top (\boldsymbol{A}\boldsymbol{\Phi}^\top \boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&\qquad + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&\qquad + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&\qquad + \frac{\beta}{\hbar}\boldsymbol{\mu}^\top (\boldsymbol{A}\boldsymbol{\Phi}^\top \boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w \\
&\qquad + \frac{\beta}{\hbar}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w \\
&\qquad + (\frac{\beta}{\hbar})^2 \hat{\boldsymbol{\mu}}_w^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w
\end{aligned} \tag{A.16}$$

$$\boldsymbol{\mu}_w^{*\top} = \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\Phi}^\top + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) \tag{A.17}$$

$$\begin{aligned}
\boldsymbol{\mu}_w^{*\top} \boldsymbol{\mu}_w &= \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} \\
&\quad + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} \\
&\quad + \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&\quad + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&\quad + \frac{\beta}{\hbar}\boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\Phi}^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w \\
&\quad + \frac{\beta}{\hbar}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w \\
&\quad + (\frac{\beta}{\hbar})^2 \hat{\boldsymbol{\mu}}_w^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w
\end{aligned} \tag{A.18}$$

$$\begin{aligned}
\boldsymbol{\mu}_w^* - \hat{\boldsymbol{\mu}}_w &= \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w - \hat{\boldsymbol{\mu}}_w \\
&= \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + (\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\hat{\boldsymbol{\mu}}_w
\end{aligned} \tag{A.19}$$

$$\begin{aligned}
(\boldsymbol{\mu}_w^* - \hat{\boldsymbol{\mu}}_w)^\top &= (\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + (\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\hat{\boldsymbol{\mu}}_w)^\top \\
&= \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\Phi}^\top + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \\
&\quad + \hat{\boldsymbol{\mu}}_w^\top (\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})
\end{aligned} \tag{A.20}$$

$$
\begin{aligned}
&(\boldsymbol{\mu}_w^* - \hat{\boldsymbol{\mu}}_w)^\top (\boldsymbol{\mu}_w^* - \hat{\boldsymbol{\mu}}_w) \\
&\quad = \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} \\
&\qquad + \hat{\boldsymbol{\mu}}_w^\top (\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} \\
&\qquad + \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + (\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&\qquad + \hat{\boldsymbol{\mu}}_w^\top (\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&\qquad + \boldsymbol{\mu}^\top \boldsymbol{A}\boldsymbol{\Phi}^\top (\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\hat{\boldsymbol{\mu}}_w \\
&\qquad + \frac{\beta}{\hbar}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top \boldsymbol{\Phi}^\top (\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\hat{\boldsymbol{\mu}}_w \\
&\qquad + \hat{\boldsymbol{\mu}}_w^\top (\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})(\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\hat{\boldsymbol{\mu}}_w
\end{aligned}
\tag{A.21}
$$

$$
\begin{aligned}
-\boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top \boldsymbol{\mu}_w^* = &-\boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} - \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&- \frac{\beta}{\hbar}\boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Phi}^\top (\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w
\end{aligned}
\tag{A.22}
$$

Expanding Equation (A.12) using Equations (A.13) to (A.22), we get

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\alpha}) = \frac{1}{2}&(\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}-\boldsymbol{I})\boldsymbol{\mu}\\
&+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}-\boldsymbol{I})\boldsymbol{\mu}\\
&+\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}-\boldsymbol{I})\boldsymbol{\mu}\\
&+\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+\frac{\beta}{\hbar}\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w\\
&+\frac{\beta}{\hbar}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w\\
&+(\frac{\beta}{\hbar})^2\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w)\\
&+\frac{\lambda}{2}(\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}\\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}+\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\hat{\boldsymbol{\mu}}_w\\
&+\frac{\beta}{\hbar}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\hat{\boldsymbol{\mu}}_w\\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\hat{\boldsymbol{\mu}}_w)\\
&+\frac{\beta}{2}(\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}\\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}\\
&+\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\hat{\boldsymbol{\mu}}_w\\
&+\frac{\beta}{\hbar}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\hat{\boldsymbol{\mu}}_w\\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\hat{\boldsymbol{\mu}}_w)\\
&-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&-\frac{\beta}{\hbar}\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w-\boldsymbol{\alpha}^\top\bar{\boldsymbol{C}}.
\end{aligned}
\tag{A.23}
$$

Further Simplifying the above expression, to obtain

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\alpha}) = \frac{1}{2}&(\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}-\boldsymbol{I})\boldsymbol{\mu} \\
&+2\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}-\boldsymbol{I})\boldsymbol{\mu} \\
&+2\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&+2\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&+(\frac{\beta}{\hbar})^2\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w) \\
&+\frac{\lambda}{2}(\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}+2\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} \\
&+2\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&+2\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\hat{\boldsymbol{\mu}}_w) \\
&+\frac{\beta}{2}(\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}+2\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} \\
&+2\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&+2\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\hat{\boldsymbol{\mu}}_w) \\
&-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
&-\frac{\beta}{\hbar}\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w-\boldsymbol{\alpha}^\top\bar{\boldsymbol{C}},
\end{aligned}
\tag{A.24}
$$

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\alpha}) = \frac{1}{2}&(2\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+2\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})\\
&+\frac{\lambda}{2}(2\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+2\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})\\
&+\frac{\beta}{2}(2\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&+2\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})\\
&-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&-\frac{\beta}{\hbar}\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w\\
&-\boldsymbol{\alpha}^\top\bar{\boldsymbol{C}}+const,
\end{aligned}
\tag{A.25}
$$

where

$$
\begin{aligned}
const = \frac{1}{2}&(\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}-\boldsymbol{I})\boldsymbol{\mu}\\
&+2\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}-\boldsymbol{I})\boldsymbol{\mu}\\
&+(\frac{\beta}{\hbar})^2\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w\\
&+\frac{\lambda}{2}(\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}\\
&+2\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}\\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\hat{\boldsymbol{\mu}}_w)\\
&+\frac{\beta}{2}(\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}\\
&+2\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}\\
&+\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\hat{\boldsymbol{\mu}}_w).
\end{aligned}
\tag{A.26}
$$

Further simplification gives

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\alpha}) = & \boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \frac{1}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \lambda(\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\lambda}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \lambda\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \beta\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} + \frac{\beta}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \beta\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}) \\
& - \boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} - \boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& - \frac{\beta}{\hbar}\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w - \boldsymbol{\alpha}^\top\bar{\boldsymbol{C}} + const.
\end{aligned} \tag{A.27}$$

We group the terms in order to take the common factors out, to obtain

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\alpha}) = & \boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \lambda\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& - \boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu} + \beta\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& \\
& + \frac{1}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \frac{\beta}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \frac{\lambda}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} - \boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& \\
& + \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \lambda\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& + \beta\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top) - \boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& - \frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \\
& \\
& - \bar{\boldsymbol{C}}^\top\boldsymbol{\alpha} + const.
\end{aligned} \tag{A.28}$$

We now focus on simplifying the first group of terms in Equation A.28, i.e.,

$$
\begin{aligned}
&\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+\lambda\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}+\beta\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+\lambda\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad-\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+\beta\boldsymbol{\mu}^\top\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\boldsymbol{\mu}^\top((\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{I})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}+\lambda\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\\
&\quad-\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{A}^{-1}\boldsymbol{A}+\beta\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A})\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\boldsymbol{\mu}^\top(\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}-\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}+\lambda\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\\
&\quad-\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{A}^{-1}\boldsymbol{A}+\beta\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A})\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\boldsymbol{\mu}^\top\boldsymbol{A}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{A}^{-1}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\lambda\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\\
&\quad-\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{A}^{-1}+\beta\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\boldsymbol{\mu}^\top\boldsymbol{A}(-\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\hbar\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=2\boldsymbol{\mu}^\top\boldsymbol{A}\mathcal{K}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha},
\end{aligned}
\tag{A.29}
$$

where $\mathcal{K}=-\frac{1}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\frac{\hbar}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$.
We now focus on simplifying the second group of terms in Equation A.28, i.e.,

$$
\begin{aligned}
&\frac{1}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+\frac{\beta}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad+\frac{\lambda}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\frac{1}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+\frac{\beta}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad+\frac{\lambda}{2}(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}-\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Sigma}\boldsymbol{A}\boldsymbol{A}^{-1}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top(\frac{1}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\frac{\beta}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\\
&\quad+\frac{\lambda}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{A}^{-1}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top(\frac{1}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\frac{\hbar}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\\
&\quad-(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\hbar\boldsymbol{\Sigma})\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top(\frac{1}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\frac{\hbar}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\\
&\quad-\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\hbar\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=(\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha})^\top(-\frac{1}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\frac{\hbar}{2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Sigma}\boldsymbol{A}\mathcal{K}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}.
\end{aligned}
\tag{A.30}
$$

We now focus on simplifying the third group of terms in Equation A.28, i.e.,

$$
\begin{aligned}
&\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad+\lambda\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad+\beta\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad-\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}+\lambda(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\\
&\quad+\beta(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}-\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1})\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}+\lambda(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\\
&\quad+\beta(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{A}-\beta\boldsymbol{\Phi}\boldsymbol{A}-\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1})\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}+\hbar(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\\
&\quad-\beta\boldsymbol{\Phi}\boldsymbol{A}-\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1})\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}+\beta(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{A}\\
&\quad-\beta\boldsymbol{\Phi}\boldsymbol{A}-\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1})\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\beta(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\\
&\quad-\beta\boldsymbol{\Phi}-\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{A}^{-1})\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})+\beta(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\\
&\quad-\beta\boldsymbol{\Phi}-\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\hbar\boldsymbol{\Sigma}))\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})+\beta(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\\
&\quad-\beta\boldsymbol{\Phi}-\frac{\beta}{\hbar}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1})(\boldsymbol{\Phi}^\top\boldsymbol{\Phi}+\hbar\boldsymbol{\Sigma}))\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})+\beta(\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\\
&\quad-\beta\boldsymbol{\Phi}-\frac{\beta}{\hbar}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\\
&\quad-\frac{\beta}{\hbar}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\hbar\boldsymbol{\Sigma}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\hbar\boldsymbol{\Sigma}))\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})+\beta(\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\\
&\quad-\beta\boldsymbol{\Phi}-\frac{\beta}{\hbar}(\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\\
&\quad-\beta(\boldsymbol{\Phi}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}))\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=\hat{\boldsymbol{\mu}}_w^\top(-\beta\boldsymbol{\Phi})\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}
\end{aligned}
$$

Hence,

$$\begin{aligned}
&\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\boldsymbol{\Sigma}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad+\lambda\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top))\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad+\beta\hat{\boldsymbol{\mu}}_w^\top(\frac{\beta}{\hbar}(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)-\boldsymbol{I})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad-\frac{\beta}{\hbar}\hat{\boldsymbol{\mu}}_w^\top(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\boldsymbol{\Phi}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&=-\hat{\boldsymbol{\mu}}_w^\top\beta\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}
\end{aligned} \tag{A.31}$$

For a desired output $\hat{\boldsymbol{\xi}}=\hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\mu}}_w$, we can estimate the optimal weight vector $\hat{\boldsymbol{\mu}}_w$ given the target trajectory $\hat{\boldsymbol{\xi}}$, using the right pseudo-inverse of $\hat{\boldsymbol{\Phi}}^\top$, similarly to [50], hence,

$$\hat{\boldsymbol{\mu}}_w=\hat{\boldsymbol{\Phi}}(\hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\Phi}})^{-1}\hat{\boldsymbol{\xi}}.$$

We substitute $\hat{\boldsymbol{\mu}}_w=\hat{\boldsymbol{\Phi}}(\hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\Phi}})^{-1}\hat{\boldsymbol{\xi}}$, hence, $\hat{\boldsymbol{\mu}}_w^\top=\hat{\boldsymbol{\xi}}^\top(\hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\Phi}})^{-1}\hat{\boldsymbol{\Phi}}^\top$ into Equation (A.31) to get

$$-\hat{\boldsymbol{\mu}}_w^\top\beta\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}=-\beta\hat{\boldsymbol{\xi}}^\top(\hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\Phi}})^{-1}\hat{\boldsymbol{\Phi}}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha} \tag{A.32}$$

We now have all the components of the derivation simplified. Let's put them together to obtain the final equation for LC-NS-KMP prediction and constraint enforcement, subsequently Kernelizing the solution.

Substituting optimal value of $\boldsymbol{\mu}^*$ from Equation (A.11) in Equation (2.6) to obtain LC-NS-KMP prediction,

$$\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}})=\boldsymbol{\Phi}^\top(\boldsymbol{\xi}_*^{\mathcal{I}})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}+\boldsymbol{\Phi}^\top(\boldsymbol{\xi}_*^{\mathcal{I}})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+\frac{\beta}{\hbar}\boldsymbol{\Phi}^\top(\boldsymbol{\xi}_*^{\mathcal{I}})(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\mu}}_w, \tag{A.33}$$

where $\boldsymbol{\xi}_*^{\mathcal{O}}=\boldsymbol{\xi}^{\mathcal{O}}(\boldsymbol{\xi}_*^{\mathcal{I}})$.

Substituting value of $\hat{\boldsymbol{\mu}}_w$ into Equation (A.33),

$$\begin{aligned}
\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}})&=\boldsymbol{\Phi}^\top(\boldsymbol{\xi}_*^{\mathcal{I}})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\mu}+\boldsymbol{\Phi}^\top(\boldsymbol{\xi}_*^{\mathcal{I}})\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}\\
&\quad+\frac{\beta}{\hbar}\boldsymbol{\Phi}^\top(\boldsymbol{\xi}_*^{\mathcal{I}})(\boldsymbol{I}-\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Phi}^\top)\hat{\boldsymbol{\Phi}}(\hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\Phi}})^{-1}\hat{\boldsymbol{\xi}}.
\end{aligned} \tag{A.34}$$

After substituting the simplification of each group from Equations (A.29), (A.30) and (A.32) back into Equation (A.28), we have

$$\mathcal{L}(\boldsymbol{\alpha})=\boldsymbol{\alpha}^\top\bar{\boldsymbol{G}}^\top\boldsymbol{\Sigma}\boldsymbol{A}\mathcal{K}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}}\boldsymbol{\alpha}+(2\boldsymbol{\mu}^\top\boldsymbol{A}\mathcal{K}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}} \tag{A.35}$$

$$-\beta\hat{\boldsymbol{\xi}}^\top(\hat{\boldsymbol{\Phi}}^\top\hat{\boldsymbol{\Phi}})^{-1}\hat{\boldsymbol{\Phi}}^\top\boldsymbol{\Phi}\boldsymbol{A}\boldsymbol{\Sigma}\bar{\boldsymbol{G}})\boldsymbol{\alpha}+const \tag{A.36}$$

Similarly to KMP, we apply well known kernel trick, i.e.

$$\boldsymbol{k}(\boldsymbol{\xi}_i^{\mathcal{I}},\boldsymbol{\xi}_j^{\mathcal{I}})=\boldsymbol{\Theta}(\boldsymbol{\xi}_i^{\mathcal{I}})^\top\boldsymbol{\Theta}(\boldsymbol{\xi}_j^{\mathcal{I}})=k(\boldsymbol{\xi}_i^{\mathcal{I}},\boldsymbol{\xi}_j^{\mathcal{I}})\boldsymbol{I}^{\mathcal{O}},$$

where $k(.)$ is a kernel function.

With the kernel treatment, we can write

$$\begin{aligned}\tilde{L}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \bar{\boldsymbol{G}}^\top \boldsymbol{\Sigma} \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \boldsymbol{\alpha} + (2\boldsymbol{\mu}^\top \boldsymbol{A} \mathcal{K} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \\ - \beta \hat{\boldsymbol{\xi}}^\top \underline{\boldsymbol{K}}^{-1} \hat{\boldsymbol{K}} \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} + \bar{\boldsymbol{C}}^\top) \boldsymbol{\alpha} + \text{const},\end{aligned} \tag{A.37}$$

and

$$\mathbb{E}(\boldsymbol{\xi}_*^{\mathcal{O}}) = \boldsymbol{k}^* \boldsymbol{A} \boldsymbol{\mu} + \boldsymbol{k}^* \boldsymbol{A} \boldsymbol{\Sigma} \bar{\boldsymbol{G}} \boldsymbol{\alpha} + \frac{\beta}{\hbar} (\hat{\boldsymbol{k}}^* - \boldsymbol{k}^* \boldsymbol{A} \hat{\boldsymbol{K}}) \underline{\boldsymbol{K}}^{-1} \hat{\boldsymbol{\xi}}, \tag{A.38}$$

with $\boldsymbol{A} = (\boldsymbol{K} + \lambda \boldsymbol{\Sigma})^{-1}$, and $\mathcal{K} = -\frac{1}{2} \boldsymbol{K} \boldsymbol{\Sigma}^{-1} \boldsymbol{K} - \frac{\hbar}{2} \boldsymbol{K}$, where,

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & k(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \dots & k(\boldsymbol{\xi}_1^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \\ k(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & k(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \dots & k(\boldsymbol{\xi}_2^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}) & k(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_2^{\mathcal{I}}) & \dots & k(\boldsymbol{\xi}_N^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}}) \end{bmatrix},$$

$$\begin{aligned}\boldsymbol{k}^* &= [k(\boldsymbol{\xi}_*^{\mathcal{I}}, \boldsymbol{\xi}_1^{\mathcal{I}}), \ \dots, \ k(\boldsymbol{\xi}_*^{\mathcal{I}}, \boldsymbol{\xi}_N^{\mathcal{I}})], \\ \underline{\boldsymbol{K}} &= \hat{\boldsymbol{\Phi}}^\top \hat{\boldsymbol{\Phi}}, \\ \hat{\boldsymbol{K}} &= \boldsymbol{\Phi}^\top \hat{\boldsymbol{\Phi}}, \\ \hat{\boldsymbol{k}} &= \boldsymbol{\Phi}(\boldsymbol{\xi}_*^{\mathcal{I}})^\top \hat{\boldsymbol{\Phi}}, \\ \hat{\boldsymbol{k}}^* &= \boldsymbol{\Phi}(\boldsymbol{\xi}_*^{\mathcal{I}})^\top \hat{\boldsymbol{\Phi}}.\end{aligned}$$

Hence, we conclude the derivation of the Linearly Constrained Null-Space Kernelized Movement Primitives (LC-NS-KMP).