Geometric Fabrics Guided Learning for Collision-Free Manipulator Global Motion Generation

by

Jingzhou Liu

Supervisors:
Florian Shkurti, University of Toronto
Karl Van Wyk, NVIDIA Research
Nathan Ratliff, NVIDIA Research

April 2024

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Geometric Fabrics Guided Learning for Collision-Free Manipulator Global Motion Generation

by

Jingzhou Liu

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF APPLIED SCIENCE IN ENGINEERING SCIENCE

in

Faculty of Applied Science and Engineering

(Engineering Science, Robotics)

Supervisors:
Florian Shkurti, University of Toronto
Karl Van Wyk, NVIDIA Research
Nathan Ratliff, NVIDIA Research

UNIVERSITY OF TORONTO

April 2024

Abstract

Collision-free global motion generation for manipulators in unknown and dynamic environments remains a challenging yet important problem for any manipulation tasks. Traditional methods, such as global planners and locally-reactive controllers, either lack the sufficient runtime speed necessary for reactive settings or fail to find valid solutions in complex environments. Recent advances in neural motion generation seek to amalgamate the strengths of these existing methods by training neural network policies on expert datasets, but they typically struggle to generalize to problems that are out of the training distribution. In this work, we present a novel neural motion generation method that aims to improve the performance of prior neural motion generation policies. We leverage geometric fabrics as a safe guiding medium to train a reinforcement learning policy conditioned on the scene. We show that our approach achieves superior performance over existing neural generation methods in terms of completeness, collision rate, and reactiveness, thereby demonstrating a new promising direction towards robust and reactive global manipulator motion generation.

Acknowledgements

I extend my heartfelt thanks to the myriad of people who have worked alongside me, provided their support, inspired me, and continually motivated me to reach new heights during my undergraduate journey.

I would like to express my profound gratitude to Prof. Florian Shkurti, Dr. Karl Van Wyk, and Dr. Nathan Ratliff for granting me the opportunity to work on this thesis project. Their invaluable guidance and insights have been instrumental in shaping the direction and success of my research.

I would also like to express my deepest appreciation to Dr. Ankur Handa for the extensive mentorship provided throughout my internship at NVIDIA. Dr. Handa has been a pivotal figure in my academic and professional development, offering guidance and support that has shaped my career path.

Additionally, I am grateful to Kelly Guo, Viktor Makoviychuk, and Gavriel State, for their support and mentorship at NVIDIA. Working alongside them has significantly enriched my professional skills and has provided a foundation for my ongoing growth.

To my friends, Arthur Allshire, Ben Agro, Dhruv Sirohi, Rassam Yazdi, Ritvik Singh, and Shrey Jain, I am profoundly grateful for being an endless source of inspiration. Our time together in Engineering Science was filled with thought-provoking discussions and steadfast support. Their camaraderie and perspective have been a source of motivation and have greatly contributed to my personal and academic growth.

Last but not least, I owe a debt of gratitude to my family for their unwavering love and support. Their belief in my abilities and constant encouragement have been the backbone of my achievements. Their sacrifices have not gone unnoticed and I am eternally grateful for everything they have done for me.

Table of Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Tables vii

List of Figures viii

1 Introduction 1

2 Background 3

2.1 Geometric Fabrics 3

2.1.1 Spectral Semi-Sprays 5

2.1.2 Riemannian Motion Policies 7

2.1.3 Fabrics 8

2.1.4 Lagrangian Fabrics and Conservative Fabrics 8

2.1.5 Energized Fabrics 10

2.1.6 Generalized Non-Linear Geometries 13

2.1.7 Finsler Geometry 14

2.1.8 Geometric Fabrics 16

2.1.9 Geometric Fabrics and The Spec Algebra 18

2.2 Reinforcement Learning 23

2.2.1 Value Functions 24

2.2.2 Policy Gradient Methods 24

2.2.3 Proximal Policy Optimization 29

2.3 Related Work 31

2.3.1 Locally Reactive Controller . . . 31
2.3.2 Global Planner . . . 31
2.3.3 Neural Motion Generation . . . 32
2.4 Research Gap . . . 33

3 Methods . . . 34
3.1 Problem Formulation . . . 34
3.2 Methods Motivation . . . 34
3.3 Geometric Fabrics Design . . . 36
3.3.1 End-Effector Attractor . . . 37
3.3.2 Joint-Space Attractor . . . 37
3.3.3 Distance-Space Repulsion . . . 38
3.3.4 Satisfying Joint Jerk and Acceleration Constraints . . . 39
3.3.5 Interactions with Real Dynamics . . . 40
3.4 Geometric Fabrics Guided Reinforcement Learning . . . 41
3.4.1 Simulation Environment . . . 42
3.4.2 Observation Space . . . 44
3.4.3 Action Space . . . 46
3.4.4 Network Architecture . . . 48
3.4.5 Reward Function and Curriculum . . . 49
3.4.6 PPO Training Setup . . . 51

4 Results and Discussion . . . 52
4.1 Quantitative Metrics . . . 52
4.2 Quantitative Results and Comparisons . . . 53
4.3 Qualitative Results . . . 54
4.4 Ablations . . . 56
4.4.1 Task Reward Formulation . . . 56
4.4.2 Performance Impacts of Intrinsic Reward . . . 56
4.4.3 Performance Impacts of the Training Curriculum . . . 57
4.4.4 Network Architecture . . . 58
4.4.5 Performance Impacts of Policy-Controlled Joint-Space . . . 58
4.4.6 Performance Impacts of Using Exploration Map in Observation . . . 60

4.4.7 Real Robot Evaluation 60

5 Conclusion 62

Bibliography 64

List of Tables

3.1 Categorical distribution for sampling one of the four possible modes at environment resets for dataset scenes. 43

3.2 Observations of the actor policy and the critic network for the Franka Panda manipulator. 44

3.3 Action space of the actor policy network for the Franka Panda manipulator. 47

3.4 Parameters of the actor and the critic network. 49

3.5 The reward function is computed by multiplying each reward term by their weight and summing them up at each policy step. 50

3.6 Proximal Policy Optimization parameters. 51

4.1 Performance comparison of methods on the MπNets dataset. 53

4.2 Parameters of the MLP-only actor and the critic network. 58

List of Figures

2.1 The potential $\psi(\mathbf{x})$ (in black) pushing the system behaviour (in blue) away from its nominal path (in green). 9

2.2 L_t^{CLIP} as a function of $r_t(\theta, \theta_k)$ for positive $\hat{A}_t$ (left) and negative $\hat{A}_t$ (right). 30

3.1 Illustration of an example solution trajectory for a collision-free manipulator global motion generation problem. 35

3.2 The potential (left) and the Finsler energy (right) used for the end-effector attractor. 38

3.3 The potential (left) and the Finsler energy (right) used for the distance-space repulsion. 39

3.4 High level PPO training pipeline of geometric fabrics guided reinforcement learning. 41

3.5 MotionBenchMaker Scenes: Table, Box, Bookshelf, and Cage. 42

3.6 MπNets Scenes: Cubby, Dresser, and Tabletop. 42

3.7 Procedurally Generated Scenes: Cuboid Forest, Shelves, and Flying Cuboids. 44

3.8 End-effector goal pose is represented using the eight keypoints at the vertices of the oriented bounding box surrounding the end-effector (in light blue). 45

3.9 Visualization of the static basis point sets and their associated vector features, displayed at half density. 46

3.10 Visualization of the dynamic basis points and their associated vector features. 47

3.11 Actor policy network architecture. 48

4.1 Success rate comparison between AIT*, Geometric Fabrics, MπNets, CuRobo, and our method. 54

4.2 Collision rate comparison between AIT*, Geometric Fabrics, MπNets, CuRobo, and our method. 55

4.3 Reaction time comparison between AIT*, Geometric Fabrics, MπNets, CuRobo, and our method. 55

4.4 Training set success rate (left) and keypoints distance error (right) of policies trained with a sparse task reward (green) vs. a dense task reward (orange). 57
4.5 Training set success rate (left) and keypoints distance error (right) of policies trained with intrinsic + sparse task reward (green) vs. sparse task reward only (orange). . . 57
4.6 Training set success rate (left) and keypoints distance error (right) of policies trained with curriculum (green) vs. without curriculum (orange). 58
4.7 Training set success rate (left) and keypoints distance error (right) of policies trained with the multi-headed 3D CNN architecture (green) vs. a vanilla MLP (orange). . . 59
4.8 Training set success rate (left), keypoints distance error (middle), and grid visitation count (right) of policies that control the joint-space (green) and policies that do not have control over the joint-space (orange). 59
4.9 Training set success rate (left), keypoints distance error (middle), and grid visitation count (right) of policies trained with the exploration map (green) and policies trained without the exploration map (orange). 60
4.10 Simulation environment (left) and the real-world deployment (right) of the neural motion generation policy trained for the Yaskawa Motoman NEX10 manipulator. . . 61

Chapter 1

Introduction

Collision-free global motion generation in unknown and dynamic environments is a fundamental skill for manipulators, underpinning all manipulation tasks. However, achieving such robust motions remains challenging due to two primary reasons. Firstly, existing approaches often rely on complete state information with known environment models, necessitating some form of visual representation of dynamic obstacles for real world deployment, such as occupancy grids [11], dynamic mesh, or signed distance fields [25]. Constructing these scene representations demands substantial computation to process raw sensor data at each time step to accommodate the dynamic world, which often proves unachievable in real time [21, 24]. Secondly, it is difficult for one motion generation method to be complete, optimal, smooth, and fast to execute. Locally reactive controllers, such as Geometric Fabrics and STORM [6, 46], prioritize smoothness and real-time speed, but often fail to find globally valid paths in scenes with complex obstacle arrangements. Alternatively, global planners, such as RRT* or AIT* [15, 18], are probabilistically complete and asymptotically optimal, but lack reactivity in dynamic settings due to slower runtime speeds.

To address these challenges, neural motion generation methods have been proposed, leveraging neural networks' ability to distill expert knowledge from optimal trajectories, provide rapid inference, and offer flexibility to various input modalities. Recently, MπNets [9] employed imitation learning (IL) to train a policy that operates directly on point clouds and outputs joint actions. While this technique is significantly faster than various global planners and exhibits reactivity with some degree of global awareness, it struggles to generalize to out-of-distribution settings due to its supervised learning approach where it is ultimately limited by the quality of the expert dataset.

This work aims to produce a collision-free neural motion generation policy that outperforms MπNets in terms of completeness, collision rate, and reactiveness while retaining its rapid inference speed without explicit scene reconstruction. Our approach is to use simulation-based reinforcement learning (RL) with Geometric Fabrics in the loop to train an actor policy. We use RL in place of IL in hopes of enhancing the policy's completeness since an RL agent can explore and learn from

continuous online data generation, whereas an IL agent is limited by the coverage of the expert dataset. In addition, RL inherently trains policies that are globally aware as it optimizes over entire trajectory rollouts, implicitly equipping policies with look-ahead capabilities [5]. More specifically, we use Proximal Policy Optimization (PPO) [39] in simulation as PPO's highly parallelizable nature scales well with GPU-based simulation [20, 22] — a technique proven successful in training other dynamic robotic tasks such as locomotion and in-hand manipulation [10, 35]. Simulation also provides a medium to systematically incorporate dynamic obstacles during training such that the policy explicitly learns to avoid fast moving obstacles, thus improving its reactiveness. Furthermore, integrating Geometric Fabrics in the loop of simulation RL training enforces smoothness in the resulting motion by constraining the policy output to satisfy joint space jerk and acceleration constraints [46], obviating the need to tune action penalization terms in the reward specification. Employing Geometric Fabrics also bridges the sim-to-real gap since the motions output by the policy during training are guaranteed to stay within the joint-level safety limits of the real robot.

This document details our work regarding geometric fabrics guided learning for collision-free manipulator global motion generation. We provide the theoretical background necessary for understanding our methods and discuss related work in Chapter 2. We delineate our methods in Chapter 3. We present experimental results and ablation studies in Chapter 4.

Chapter 2

Background

This section presents background information that is sufficient for understanding the theory behind Geometric Fabrics and Proximal Policy Optimization, constituting the two core methods used in our approach.

2.1 Geometric Fabrics

The underlying dynamics that a policy operates on has significant effects on policy learning, both in terms of sample efficiency during training and performance during inference [1, 2]. Consequently, it is beneficial to design these underlying dynamics such that they can provide a useful medium that facilitates policy learning.

In the most primitive setting where the policy is trained to output joint-level torque or position commands, the policy not only needs to achieve the task objective, but it must also implicitly learn to handle various non-task-specific behaviour, such as compensating for the natural dynamics of the system, satisfying joint-level constraints, and collision avoidance [3, 10].

To reduce the complexity of learning, control systems can be leveraged to create more powerful action spaces for the policy to use during training. For instance, inverse dynamics control effectively eliminates the natural dynamics of the system for the policy, hence providing the policy with a Euclidean geometry in joint-space such that it no longer needs to learn dynamic compensation [31]. Operational Space Control (OSC) further removes the need for the policy to learn a mapping between joint-space and task-space by establishing a Euclidean geometry directly in the task-space, enabling straight-line motion as the nominal behaviour of the end-effector [13]. Leveraging these classical controllers, it has been shown numerous times that policy learning becomes easier as the controller reshapes the underlying natural dynamics into a dynamical system that exhibits more useful behaviour [23, 47].

However, if one attempts to capture even more commonly useful behavioural elements, such as collision avoidance, joint-limit avoidance, etc., these classical controllers often fail to exhibit

the desired behaviour due to fundamental limitations in the expressivity of classical mechanical systems. These limitations are evident in the following example.

Consider an acceleration policy derived from combining multiple classical systems together, where each classical system is designed to describe one behaviour (end-effector reaching, repelling from obstacles, etc.). Each classical system evolves according to the following equation:

$$\mathbf{M}_{\mathrm{i}}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{f}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}}) = -\partial_{\mathbf{q}}\psi_{\mathrm{i}}(\mathbf{q}) - \mathbf{B}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \tag{2.1}$$

where $\mathbf{M}_i(\mathbf{q})$ is the system's generalized mass matrix, $\mathbf{f}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}})$ describes Coriolis forces, $\psi_{\mathrm{i}}(\mathbf{q})$ is a potential function that exerts a force on the nominal system, and $\mathbf{B}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ is a positive definite damping matrix. If we define each system's policy as $\pi_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}}) = \ddot{\mathbf{q}}$, we get the following:

$$\pi_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}}) = -\mathbf{M}_{\mathrm{i}}^{-\mathbf{1}} \left(-\mathbf{f}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}}) - \partial_{\mathbf{q}}\psi_{\mathrm{i}}(\mathbf{q}) - \mathbf{B}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}\right) \tag{2.2}$$

Combining these classical systems results in a combined policy with the following form:

$$\ddot{\mathbf{q}} = \left(\sum_{\mathrm{i}} \mathbf{M}_{\mathrm{i}}\right)^{\text{-1}} \sum_{\mathrm{i}} \mathbf{M}_{\mathrm{i}}\pi_{\mathrm{i}} \tag{2.3}$$

Note that the combined system's acceleration is a metric-weighted average where the metric is each system's mass matrix. From this example, we can immediately recognize several issues from using classical mechanical systems. Firstly, the metric $\mathbf{M}_i(\mathbf{q})$ is merely position dependent, which is not as expressive as one that is both position and velocity dependent. These metrics dictate the priority of each policy π_i. Obstacle avoidance policies are an example of where velocity dependent metric is useful since such policies should be deprioritized by the metric if the system is moving away from an obstacle; this behaviour cannot be specified if the metric is only position dependent. Secondly, the nominal behaviour of the system $\mathbf{M}_{\mathrm{i}}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{f}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{0}$ is intrinsically coupled with the metric in classical systems since both $\mathbf{M}_i(\mathbf{q})$ and $\mathbf{f}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}})$ are derived from computing the Euler-Lagrange equation using a Lagrangian. The nominal behaviour of the combined system depends on both the nominal behaviour of each sub-system as well as the priority metrics that weigh the sub-systems. Since the nominal behaviour and its associated metric cannot be specified independently in a classical system, one cannot effectively design the overall system's nominal behaviour. Thirdly, due to the lack of design expressivity of the system's nominal behaviour, almost all behavioural shaping in classical systems are achieved via potentials, which induce external forces that fight

against the nominal dynamics of the system in order to achieve the desired motion. However, combining multiple potentials can easily result in task conflicts in practice, causing the system to converge to undesired local minima.

In spite of the various aforementioned drawbacks of classical mechanical systems, it is worth noting that these systems are intrinsically stable due to their conservative nature. This is a direct result from classical mechanics, where systems derived from taking the Euler-Lagrange equation (EL) of a Lagrangian (in the form of $\mathbf{M}_{\mathrm{i}}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{f}_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{0}$) are conservative due to the absence of any non-conservative forces.

Geometric Fabrics, on the other hand, is a class of provably-stable acceleration-based controllers that aims to address each of the three aforementioned issues of classical mechanical systems. Firstly, it generalizes classical mechanics to Finsler geometries, which permits velocity-dependent metrics. Secondly, it allows one to design the system's nominal behaviour independently from the system's metric; this is achieved by bending the Finsler system to match a desired geometry (hence obtaining the desired nominal behaviour), while conserving the Finsler system's energy (which directly determines the system's metric). Lastly, since a Geometric Fabric's nominal behaviour can be freely specified and it is geometric in nature, one can pack many useful behaviour in the system's geometry (nominal behaviour) itself, which reduces the number of potentials required for behavioural shaping. Intuitively, the geometric nominal behaviour of Geometric Fabrics can be viewed as a road network of paths, and potentials are merely steering the system from one road to another in order to achieve a task objective (see Figure 2.1).

The following sections will explain the above claims in detail, building up to a working understanding of the ways in which Geometric Fabrics can serve as a useful guiding-medium for policy learning. Note that the following results and derivations are adapted from [33, 34, 46].

2.1.1 Spectral Semi-Sprays

Geometric fabrics belong to a class of second-order systems called *spectral semi-sprays*, or *specs* for short. These systems have form of $\mathbf{M}\ddot{\mathbf{x}} + \mathbf{f} = \mathbf{0}$, where x is in a task space $\mathcal{X}$. Using the notation of spectral semi-spray, the differential equation above can be denoted as $(\mathbf{M}, \mathbf{f})_{\mathcal{X}}$, which is known as the nominal form of a spec.

Each spec can be designed to exhibit a certain behaviour in a desired task space. For instance, one spec can be specified as an end-effector attractor while another can be used for obstacle avoidance for a particular link. Therefore, we need to derive the ways in which specs can be transformed

between different task spaces and combined into one spec that captures all of the desired behaviour. These transformation and combination operations constitute *spec algebra*.

2.1.1.1 Pullback

Pullback is an operation of spec algebra that specifies the way in which specs are transformed from one task space to another. For example, if a spec defines a certain behaviour in a manipulator's end effector space, this spec can be pulled-back to the robot's joint space such that the same behaviour is exhibited in terms of the robot's joint variables.

Given a differentiable map $\phi : \mathcal{Q} \to \mathcal{X}$, we have $\mathbf{x} = \phi(\mathbf{q})$, $\mathbf{J} = \partial_{\mathbf{q}}\phi$, and $\ddot{\mathbf{x}} = \mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}}$. To transform a spec $(\mathbf{M}, \mathbf{f})_{\mathcal{X}}$ from task space $\mathcal{X}$ to $\mathcal{Q}$, we need to find a differential equation in terms of $\ddot{\mathbf{q}}$ such that its behaviour matches the desired behaviour defined by $\mathbf{M}\ddot{\mathbf{x}}^d + \mathbf{f} = \mathbf{0}$. Concretely, we can construct the following optimization problem :

$$\begin{aligned} \min_{\ddot{\mathbf{q}}} \quad & \frac{1}{2} \left\| \ddot{\mathbf{x}}^d - \ddot{\mathbf{x}} \right\|_{\mathbf{M}}^2 \\ \text{s.t.} \quad & \ddot{\mathbf{x}}^d = -\mathbf{M}^{-1}\mathbf{f}, \\ & \dot{\mathbf{x}} = \mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\mathbf{q} \end{aligned}$$

The close-form solution to the optimization problem is:

$$\ddot{\mathbf{q}} = -(\mathbf{J}^T\mathbf{M}\mathbf{J})^{\dagger}\mathbf{J}^T(\mathbf{f} + \mathbf{M}\dot{\mathbf{J}}\dot{\mathbf{q}}) \tag{2.4}$$

Rearranging the above yields:

$$(\mathbf{J}^T\mathbf{M}\mathbf{J})\ddot{\mathbf{q}} + \mathbf{J}^T(\mathbf{f} + \mathbf{M}\dot{\mathbf{J}}\dot{\mathbf{q}}) = \mathbf{0} \tag{2.5}$$

We note that (2.5) is the spec $(\mathbf{M}, \mathbf{f})_{\mathcal{X}}$ pulled-back into $\mathcal{Q}$. Therefore, we can define the pullback operation as:

$$\text{pull}_{\phi}(\mathbf{M}, \mathbf{f})_{\mathcal{X}} = (\mathbf{J}^T\mathbf{M}\mathbf{J}, \mathbf{J}^T(\mathbf{f} + \mathbf{M}\dot{\mathbf{J}}\dot{\mathbf{q}}))_{\mathcal{Q}} \tag{2.6}$$

2.1.1.2 Combination

Combination is another operation of spec algebra. We can combine specs in the same task space by simply summing up their associated differential equations:

$$\left(\sum_i \mathbf{M}_i\right)\ddot{\mathbf{x}} + \left(\sum_i \mathbf{f}_i\right) = \mathbf{0} \tag{2.7}$$

Therefore, we can define the spec combination operation as:

$$\sum_i (\mathbf{M}_i, \mathbf{f}_i)_{\mathcal{X}} = (\sum_i \mathbf{M}_i, \sum_i \mathbf{f}_i)_{\mathcal{X}} \tag{2.8}$$

We can also rewrite the differential equation $\mathbf{M}_{\mathrm{i}}\ddot{\mathbf{x}} + \mathbf{f}_{\mathrm{i}} = \mathbf{0}$ as $\ddot{\mathbf{x}} = -\mathbf{M}_{\mathrm{i}}^{-1}\mathbf{f}_{\mathrm{i}}$, where $\ddot{\mathbf{x}}$ can be interpreted as a sub-policy $\pi_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}})$. Note that each $\pi_{\mathrm{i}}(\mathbf{q}, \dot{\mathbf{q}})$ specifies the behaviour of $(\mathbf{M}_i, \mathbf{f}_i)_{\mathcal{X}}$. We then define the canonical form of a spec as $(\mathbf{M}_{\mathrm{i}}, \pi_{\mathrm{i}})^{\mathcal{C}}_{\mathcal{X}}$. By rearranging Equation (2.7), we can write the spec combination operation in the canonical form as:

$$\sum_i (\mathbf{M}_i, \pi_{\mathrm{i}})^{\mathcal{C}}_{\mathcal{X}} = \left(\sum_i \mathbf{M}_i, (\sum_{\mathrm{i}} \mathbf{M}_{\mathrm{i}})^{\text{-}1} \sum_{\mathrm{i}} \mathbf{M}_{\mathrm{i}}\pi_{\mathrm{i}}\right)^{\mathcal{C}}_{\mathcal{X}} \tag{2.9}$$

From Equation (2.9), we can readily observe in the combined policy $\ddot{\mathbf{x}} = (\sum_{\mathrm{i}} \mathbf{M}_{\mathrm{i}})^{\text{-}1} \sum_{\mathrm{i}} \mathbf{M}_{\mathrm{i}}\pi_{\mathrm{i}}$ that each sub-policy π_{i} is weighted by its associated metric $\mathbf{M}_i$.

When combining specs that are expressed in different task spaces, we first pullback each spec to a common task space, then combine the pulled-back specs using the combination operation.

2.1.2 Riemannian Motion Policies

Riemannian Motion Policies (RMPs) [32] is a predecessor to Geometric Fabrics. In the RMPs framework, each RMP specifies a particular behaviour using a user-designed acceleration sub-policy π_{i}. In addition, each RMP also has a user-defined velocity-dependent priority metric $\mathbf{M}_{\mathrm{i}}$ that weighs the associated sub-policy. Therefore, an RMP is by definition the same as a spectral semi-spray and is commonly denoted in the canonical form of a spec as $(\mathbf{M}_{\mathrm{i}}, \pi_{\mathrm{i}})^{\mathcal{C}}_{\mathcal{X}}$. The RMPs are combined together using spec algebra.

It is worth noting that the RMPs framework gives one the freedom to design the sub-policy independently from the metric for each RMP, which means that the metric is generally not coupled with the sub-policy via a Lagrangian. This also allows the metric to be velocity-dependent. As a

result, an RMP cannot be considered a classical mechanical system and thus is not guaranteed to be intrinsically stable. In practice, each RMP is specified based on one's intuition and experience, and the stability of RMPs are empirically verified. Thus, designing stable RMPs can pose a challenge for novice users [34]. Some basic examples of RMPs are shown in Section V.B of [32], including an end-effector position tracking RMP, an obstacle-avoidance RMP, as well as a joint-space redundancy resolution RMP.

Overall, the RMPs framework enables expressive policy designs by allowing velocity-dependent metrics that are also decoupled from the sub-policies; however, RMPs lack theoretical stability guarantees. We note that Geometric Fabrics is a provably-stable class of RMPs, while still retaining RMPs' ability to design metrics and sub-policies independently.

2.1.3 Fabrics

A spec can be *forced* by a position-dependent potential $\psi_{\mathrm{i}}(\mathbf{q})$ as follows:

$$\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) = -\partial_{\mathbf{x}}\psi(\mathbf{x}) \tag{2.10}$$

where the negative gradient of the potential $-\partial_{\mathbf{x}}\psi(\mathbf{x})$ is the force exerted by the potential onto the spec. If forcing the spec with $\psi(\mathbf{x})$ causes $\mathbf{x}$ to converge to a local minimum of $\psi(\mathbf{x})$, then we define this spec as a *fabric*. From arranging the fabric equation for the system's acceleration $\ddot{\mathbf{x}} = -\mathbf{M}^{-1}\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}) - \mathbf{M}^{-1}\partial_{\mathbf{x}}\psi(\mathbf{x})$, we define $-\mathbf{M}^{-1}\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}})$ as the system's nominal acceleration and $\mathbf{M}^{-1}\partial_{\mathbf{x}}\psi(\mathbf{x})$ as the forced acceleration. The forced acceleration serves to push the system away from its nominal behaviour, illustrated in Figure 2.1.

We can further observe that both the nominal acceleration and the forced acceleration are dependent on the metric $\mathbf{M}$. Specifically, the eigenvectors and the associated eigenvalues of $\mathbf{M}$ dictate the directions and the extent that the forces can exert on the system to impact its behaviour.

2.1.4 Lagrangian Fabrics and Conservative Fabrics

Given a Lagrangian $\mathcal{L}(\mathbf{x}, \dot{\mathbf{x}})$, its associated equation of motion (EOM) derived from the Euler-Lagrange (EL) equation forms a spec: $\mathbf{M}_{\mathcal{L}}\ddot{\mathbf{x}} + \mathbf{f}_{\mathcal{L}} = \mathbf{0}$, where $\mathbf{M}_{\mathcal{L}} = \partial^{\mathbf{2}}_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}$ and $\mathbf{f}_{\mathcal{L}} = \partial_{\dot{\mathbf{x}}\mathbf{x}}\mathcal{L}\dot{\mathbf{x}} - \partial_{\mathbf{x}}\mathcal{L}$. This Lagrangian induced spec $(\mathbf{M}_{\mathcal{L}}, \mathbf{f}_{\mathcal{L}})_{\mathcal{X}}$ forms a *Lagrangian fabric*.

In addition, if $\mathbf{M}$ is full-rank, its associated Lagrangian is known as an energy Lagrangian, denoted as $\mathcal{L}_e$. The Lagrangian fabric formed from an energy Lagrangian $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$ is known as

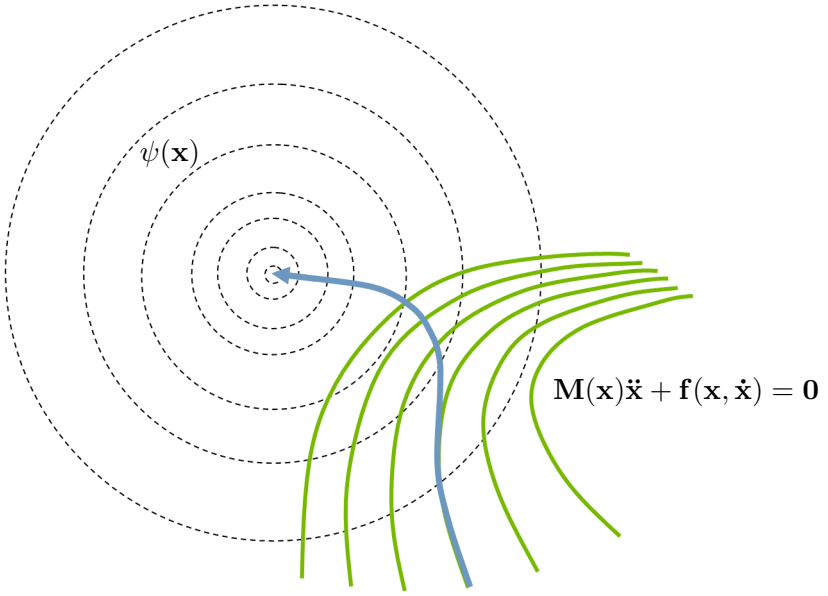

Figure 2.1: The potential $\psi(\mathbf{x})$ (in black) pushing the system behaviour (in blue) away from its nominal path (in green).

a *conservative fabric* since it conserves the system's energy. The system's energy is also known as the system's Hamiltonian $\mathcal{H}_e = \partial_{\dot{\mathbf{x}}}\mathcal{L}_e^T\dot{\mathbf{x}} - \mathcal{L}_e$.

We prove the energy conservation property of $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$ by first calculating the time derivative of $\mathcal{H}_e$, then showing that $\dot{\mathcal{H}}_e = 0$:

$$\dot{\mathcal{H}}_e = \frac{d}{dt}[\partial_{\dot{\mathbf{x}}}\mathcal{L}_e - \mathcal{L}_e] \tag{2.11}$$

$$= (\partial_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_e\ddot{\mathbf{x}} + \partial_{\dot{\mathbf{x}}\mathbf{x}}\mathcal{L}_e\dot{\mathbf{x}})^T\dot{\mathbf{x}} + \partial_{\dot{\mathbf{x}}}\mathcal{L}_e^T\ddot{\mathbf{x}} - (\partial_{\dot{\mathbf{x}}}\mathcal{L}_e^T\ddot{\mathbf{x}} + \partial_{\mathbf{x}}\mathcal{L}_e^T\dot{\mathbf{x}}) \tag{2.12}$$

$$= \dot{\mathbf{x}}^T(\partial_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_e\ddot{\mathbf{x}} + \partial_{\dot{\mathbf{x}}\mathbf{x}}\mathcal{L}_e\dot{\mathbf{x}} - \partial_{\dot{\mathbf{x}}}\mathcal{L}_e^T\dot{\mathbf{x}}) \tag{2.13}$$

$$= \dot{\mathbf{x}}^T(\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e) \tag{2.14}$$

Since $\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e = \mathbf{0}$, it is evident from Equation (2.14) that $\dot{\mathcal{H}}_e = 0$, hence demonstrating that the system's energy is conserved.

We can also modify the conservative fabric $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$ by introducing an additional force term $\mathbf{f}_f$. As such, the modified spec becomes $(\mathbf{M}_e, \mathbf{f}_e + \mathbf{f}_f)_{\mathcal{X}}$, with the associated differential equation $\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e + \mathbf{f}_f = 0$. We now prove that this modified system $(\mathbf{M}_e, \mathbf{f}_e + \mathbf{f}_f)_{\mathcal{X}}$ still conserves energy

if and only if $\dot{\mathbf{x}}^T\mathbf{f}_f = 0$. Following Equation (2.14) and substituting $\ddot{\mathbf{x}} = -\mathbf{M}_e^{-1}(\mathbf{f}_e + \mathbf{f}_f)$:

$$\dot{\mathcal{H}}_e = \dot{\mathbf{x}}^T(\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e) \tag{2.15}$$

$$= \dot{\mathbf{x}}^T\left(\mathbf{M}_e\left(-\mathbf{M}_e^{-1}(\mathbf{f}_e + \mathbf{f}_f)\right) + \mathbf{f}_e\right) \tag{2.16}$$

$$= \dot{\mathbf{x}}^T\left(-\mathbf{f}_e - \mathbf{f}_f + \mathbf{f}_e\right) \tag{2.17}$$

$$= \dot{\mathbf{x}}^T\mathbf{f}_f = 0. \tag{2.18}$$

Therefore, the modified system $(\mathbf{M}_e, \mathbf{f}_e + \mathbf{f}_f)_{\mathcal{X}}$ remains a conservative fabric. We denote this modified system as a *zero-work modification* to $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$.

We now consider the conservative fabric $(\mathbf{M}_e, \mathbf{f}_e+\mathbf{f}_f)_{\mathcal{X}}$ being forced by a lower-bounded potential $\psi(\mathbf{x})$ and damped via $-\mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}$. The system's differential equation thus becomes $\mathbf{M}_e\ddot{\mathbf{x}}+\mathbf{f}_e+\mathbf{f}_f = -\partial_{\mathbf{x}}\psi - \mathbf{B}\dot{\mathbf{x}}$, with the system's total energy equaling $\mathcal{H}_e^\psi = \mathcal{H}_e + \psi(\mathbf{x})$. We can prove that this system always converges and thus is guaranteed to be stable by examining $\dot{\mathcal{H}}_e^\psi$:

$$\dot{\mathcal{H}}_e^\psi = \dot{\mathcal{H}}_e + \dot{\psi} \tag{2.19}$$

$$= \dot{\mathbf{x}}^T(\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e) + \partial_{\mathbf{x}}\psi^T\dot{\mathbf{x}} \tag{2.20}$$

$$= \dot{\mathbf{x}}^T(\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e + \partial_{\mathbf{x}}\psi) \tag{2.21}$$

$$= \dot{\mathbf{x}}^T\left(\mathbf{M}_e\left(-\mathbf{M}_e^{-1}(\mathbf{f}_e + \mathbf{f}_f + \partial_{\mathbf{x}}\psi + \mathbf{B}\dot{\mathbf{x}})\right) + \mathbf{f}_e + \partial_{\mathbf{x}}\psi\right) \tag{2.22}$$

$$= \dot{\mathbf{x}}^T\left(-\mathbf{f}_e - \partial_{\mathbf{x}}\psi - \mathbf{B}\dot{\mathbf{x}} + \mathbf{f}_e + \partial_{\mathbf{x}}\psi\right) - \dot{\mathbf{x}}^T\mathbf{f}_f \tag{2.23}$$

$$= -\dot{\mathbf{x}}^T\mathbf{B}\dot{\mathbf{x}} \tag{2.24}$$

If we specify $\mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})$ such that it is strictly positive definite, then $\dot{\mathcal{H}}_e^\psi \leq 0$. Moreover, since $\mathcal{H}_e^\psi$ is lower-bounded and $\dot{\mathcal{H}}_e^\psi \leq 0$, we can know that $\dot{\mathcal{H}}_e^\psi \to 0$, implying $\dot{\mathbf{x}} \to \mathbf{0}$. Therefore, conservative fabrics are guaranteed to be stable by adding a damper that drains the system's energy overtime.

2.1.5 Energized Fabrics

Consider an arbitrary spec of the form $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ with the differential equation $\ddot{\mathbf{x}} + \mathbf{h}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}$. In addition, consider a conservative fabric $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$ induced from an energy Lagrangian $\mathcal{L}_e$ with an associated energy level $\mathcal{H}_e$. We claim that the spec $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ can be transformed into a conservative fabric that has the same energy level as $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$. This process is called the *energization of* $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ *with* $\mathcal{L}_e$, which produces a conservative fabric denoted as an *energized fabric*. We emphasize that

this resulting energized fabric is energy conserving with an energy level $\mathcal{H}_e$. We can represent this energized fabric with the following differential equation:

$$\ddot{\mathbf{x}} + \mathbf{h}(\mathbf{x}, \dot{\mathbf{x}}) + \alpha_{\mathcal{H}_e}\dot{\mathbf{x}} = 0 \tag{2.25}$$

where we define $\alpha_{\mathcal{H}_e}$ as

$$\alpha_{\mathcal{H}_e} = -\left(\dot{\mathbf{x}}^T\mathbf{M}_e\dot{\mathbf{x}}\right)^{-1}\dot{\mathbf{x}}^T[\mathbf{M}_e\mathbf{h} - \mathbf{f}_e] \tag{2.26}$$

By rearranging Equation (2.25), we can represent the energized fabric system as:

$$\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h} - \mathbf{f}_e] = 0 \tag{2.27}$$

where $\mathbf{P}_e = \mathbf{M}_e^{\frac{1}{2}}\left[\mathbf{I} - \hat{\mathbf{v}}\hat{\mathbf{v}}^T\right]\mathbf{M}_e^{-\frac{1}{2}}$ with $\mathbf{v} = \mathbf{M}_e^{\frac{1}{2}}\dot{\mathbf{x}}$ and $\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$. Note that it is straight forward to show that for an arbitrary $\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}})$,

$$\dot{\mathbf{x}}^T\mathbf{P}_e\mathbf{f} = 0 \tag{2.28}$$

Hence, we can interpret $\mathbf{P}_e$ as a scaled orthogonal projector of $\dot{\mathbf{x}}$ since $\mathbf{P}_e$ projects $\mathbf{f}$ onto the orthogonal space of $\dot{\mathbf{x}}$.

We note that Equation (2.27) in spec notation is $(\mathbf{M}_e, \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h} - \mathbf{f}_e])_{\mathcal{X}}$. Let $\mathbf{f} = [\mathbf{M}_e\mathbf{h} - \mathbf{f}_e]$ and $\mathbf{f}_f = \mathbf{P}_e\mathbf{f} = \mathbf{P}_e[\mathbf{M}_e\mathbf{h} - \mathbf{f}_e]$. The energized fabric spec is thus $(\mathbf{M}_e, \mathbf{f}_e + \mathbf{f}_f)_{\mathcal{X}}$. From Equation (2.28), it can be easily observed that $\mathbf{f}_f = \mathbf{P}_e[\mathbf{M}_e\mathbf{h} - \mathbf{f}_e]$ is a *zero-work modification* to $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$ because $\dot{\mathbf{x}}^T\mathbf{f}_f = \dot{\mathbf{x}}^T\mathbf{P}_e\mathbf{f} = 0$. Since the energized fabric $(\mathbf{M}_e, \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h} - \mathbf{f}_e])_{\mathcal{X}}$ is merely a zero-work modification to a conservative fabric $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$, we have shown that the energized fabric remains conservative with an energy level $\mathcal{H}_e$—the same energy as $(\mathbf{M}_e, \mathbf{f}_e)_{\mathcal{X}}$.

Now that we have proved that the energized fabric is conservative with an energy level $\mathcal{H}_e$, we can derive Equation (2.26) starting from Equation (2.14) and then substituting Equation (2.25):

$$\dot{\mathcal{H}}_e = \dot{\mathbf{x}}^T(\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e) = 0 \tag{2.29}$$

$$\dot{\mathbf{x}}^T\left[\mathbf{M}_e(-\mathbf{h} - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}}) + \mathbf{f}_e\right] = 0 \tag{2.30}$$

Now solving for $\alpha_{\mathcal{H}_e}$, we arrive at Equation (2.26):

$$\alpha_{\mathcal{H}_e} = -\frac{\dot{\mathbf{x}}^T \mathbf{M}_e \mathbf{h} - \dot{\mathbf{x}}^T \mathbf{f}_e}{\dot{\mathbf{x}}^T \mathbf{M}_e \dot{\mathbf{x}}} \tag{2.31}$$

$$= -\left(\dot{\mathbf{x}}^T \mathbf{M}_e \dot{\mathbf{x}}\right)^{-1} \dot{\mathbf{x}}^T [\mathbf{M}_e \mathbf{h} - \mathbf{f}_e] \tag{2.32}$$

We finally define the energization operator using the spec notation in the natural form:

$$\text{energize}_{\mathcal{L}_e}\{(\mathbf{I}, \mathbf{h})_{\mathcal{X}}\} = (\mathbf{M}_e, \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e \mathbf{h} - \mathbf{f}_e])_{\mathbf{x}} \tag{2.33}$$

We can also write the energization operator in the spec's canonical form by rearranging Equation (2.25) as $\ddot{\mathbf{x}} = -\mathbf{h}(\mathbf{x}, \dot{\mathbf{x}}) - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}}$, thus forming:

$$\text{energize}_{\mathcal{L}_e}\{(\mathbf{I}, -\mathbf{h})^{\mathcal{C}}_{\mathcal{X}}\} = (\mathbf{M}_e, -\mathbf{h}(\mathbf{x}, \dot{\mathbf{x}}) - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}})^{\mathcal{C}}_{\mathbf{x}} \tag{2.34}$$

To summarize, the energization of $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ with $\mathcal{L}_e$ is the process of making $\ddot{\mathbf{x}} + \mathbf{h}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}$ have the same energy as the system $\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e = 0$ derived from $\mathcal{L}_e$. The resulting system is called an *energized fabric*, which is an energy conserving fabric with an energy level $\mathcal{H}_e$.

We now provide a high-level overview regarding the ways in which energization is used for remedying some of the aforementioned issues associated with classical mechanical systems. Firstly, the spec $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ serves the sole purpose of specifying a certain sub-policy's nominal behaviour. However, $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ is not guaranteed to be stable. We can make $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ stable by transforming it into a conservative fabric via energization. Secondly, from Equation (2.33), it can be seen that the energized fabric has an associated metric $\mathbf{M}_e$ that is derived from $\mathcal{L}_e$. The idea is that since the metric $\mathbf{M}_e$ is not originally related to the system $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ that specifies the nominal behaviour, one can design the metric independently from the nominal behaviour.

However, from Equation (2.34), one can observe that the energized fabric's nominal behaviour is modified to $\ddot{\mathbf{x}} = -\mathbf{h}(\mathbf{x}, \dot{\mathbf{x}}) - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}}$, thus differing from the intended nominal behaviour specified by $\ddot{\mathbf{x}} = -\mathbf{h}(\mathbf{x}, \dot{\mathbf{x}})$. In general, the added acceleration term $-\alpha_{\mathcal{H}_e}\dot{\mathbf{x}}$ causes the energized fabric's nominal behaviour to change from that of $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$. Specifically, the modified nominal behaviour traces out a different path from $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ despite the same initial conditions; this severely hinders one's ability to design the fabric's nominal behaviour at will. However, we will show that we can impose a few simple design constraints on both $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$ and $\mathcal{L}_e$ such that the modified nominal behaviour traces out the same path as $(\mathbf{I}, \mathbf{h})_{\mathcal{X}}$. When two systems' behaviour are path-consistent, they share the

same underlying *geometry*, and thus we denote them as *geometric*. Therefore, in the next sections, we will explore the necessary properties that will make an energized fabric *geometric*. To emphasize, if an energized fabric is geometric, one can design both the system's nominal behaviour and its priority metric completely independently while guaranteeing the system's stability, hence making this framework much more powerful and expressive than a classical mechanical system.

2.1.6 Generalized Non-Linear Geometries

A generalized non-linear geometry is a second-order differential equation describing a smooth collection of paths. This means that the solutions of the differential equation with the same initial position and the same initial velocity direction $(\mathbf{x}_0, \hat{\mathbf{v}}_0)$ trace out the same path. Second-order differential equations with this property is denoted as *path-consistent*. We note that an arbitrary second order differential equation in general is not path-consistent. This means that different initial speeds result in trajectories that trace out different paths. An intuitive example of a non-path-consistent system is an orbiting satellite, where depending on the initial speed, the satellite would either spiral inward, stay in orbit, or break orbit.

For these generalized non-linear geometries, trajectories starting from the same $(\mathbf{x}_0, \hat{\mathbf{v}}_0)$ are merely one time re-parameterization away from one another; these trajectories are known to be *equivalent*. Concretely, if $\mathbf{x}_t(t)$ is a trajectory with time index t, given a smooth monotonically increasing function $t(s)$, then $\mathbf{x}_s(s) = \mathbf{x}_t(t(s))$ creates an equivalent trajectory that is a time re-parameterization of $\mathbf{x}_t(t)$.

2.1.6.1 Geometry Generator Equation

A geometry generator is an ordinary second-order differential equation of the form

$$\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0} \tag{2.35}$$

where $\mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}})$ is a smooth, positively homogeneous of degree 2 (HD2) in velocities. $\mathbf{h}_2(\mathbf{x}, \lambda\dot{\mathbf{x}}) = \lambda^2\mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}})$ for $\lambda > 0$.

Given an initial condition $(\mathbf{x}_0, \lambda\hat{\mathbf{v}}_0)$ for any $\lambda > 0$, solutions to a geometry generator produce equivalent trajectories. In other words, given the same initial position and the same initial velocity direction, the solutions of a geometry generator trace out the same path. We emphasize that for the same $\mathbf{x}_0$ and $\hat{\mathbf{v}}_0$ but different values of λ, solutions given the initial condition$(\mathbf{x}_0, \lambda\hat{\mathbf{v}}_0)$ are unique

since geometry generators are ordinary differential equations. However, those solutions only differ from each other by a time re-parameterization.

2.1.6.2 Geometric Equation

Given a geometry generator equation $\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}$, its corresponding *geometric equation* is the following:

$$\mathbf{P}_{\dot{\mathbf{x}}}^{\perp} \left[\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}})\right] = \mathbf{0} \tag{2.36}$$

where $\mathbf{P}_{\dot{\mathbf{x}}}^{\perp} = \mathbf{I} - \hat{\mathbf{x}}\hat{\mathbf{x}}^T$ is a projector that projects orthogonally to $\dot{\mathbf{x}}$. Since $\mathbf{P}_{\dot{\mathbf{x}}}^{\perp}$ has a non-trivial nullspace spanned by $\dot{\mathbf{x}}$, any solution to the geometric equation satisfies:

$$\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) + \alpha(t)\dot{\mathbf{x}} = \mathbf{0} \tag{2.37}$$

where $\alpha(t)$ is some smooth function that can be interpreted as an acceleration along $\dot{\mathbf{x}}$. Equation (2.37) is known as the *explicit form geometric equation*. In other words, if $\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}$ generates geometric trajectories, then $\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) + \alpha(t)\dot{\mathbf{x}} = \mathbf{0}$ generates trajectories that follow the same path.

We emphasize that there is no unique solution to the geometric equation for an initial condition since $\mathbf{P}_{\dot{\mathbf{x}}}^{\perp}$ has a non-trivial nullspace, hence providing a degree of redundancy. Concretely, given $(\mathbf{x}_0, \lambda\hat{\mathbf{v}_0})$, the solutions to the geometric equation are the set of all trajectories following the same path.

2.1.7 Finsler Geometry

Finsler geometries are non-linear geometries such that their geometric equations are defined by the EOM induced from taking the Euler-Lagrange equation of Finsler structures.

2.1.7.1 Finsler Structure

A Finsler structure, denoted as $\mathcal{L}_g(\mathbf{x}, \dot{\mathbf{x}})$, is a Lagrangian with the following properties:

1. $\mathcal{L}_g(\mathbf{x}, \dot{\mathbf{x}}) \geq 0$ for all $\dot{\mathbf{x}}$ with equality if and only if $\dot{\mathbf{x}} = \mathbf{0}$.
2. $\mathcal{L}_g(\mathbf{x}, \dot{\mathbf{x}})$ is positively homogenous (HD1) in velocity. Specifically, $\mathcal{L}_g(\mathbf{x}, \lambda\dot{\mathbf{x}}) = \lambda\mathcal{L}_g(\mathbf{x}, \dot{\mathbf{x}})$ for $\lambda \geq 0$.

3. $\mathcal{L}_g(\mathbf{x}, \dot{\mathbf{x}})$ has an associated Finseler energy $\mathcal{L}_e(\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2}\mathcal{L}_g^2$ such that $\partial^2_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_e$ is invertible for all $(\mathbf{x}, \dot{\mathbf{x}})$.

Taking the Euler-Lagrange equation with respect to a Finsler structure $\mathcal{L}_g$ results in the following EOM:

$$\mathbf{M}_\mathrm{g}\ddot{\mathbf{x}} + \mathbf{f}_\mathrm{g} = \mathbf{0} \tag{2.38}$$

where $\mathbf{M}_\mathrm{g} = \partial^{\mathbf{2}}_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_\mathrm{g}$ and $\mathbf{f}_g = \partial_{\dot{\mathbf{x}}\mathbf{x}}\mathcal{L}_\mathrm{g}\dot{\mathbf{x}} - \partial_\mathbf{x}\mathcal{L}_\mathrm{g}$. This EOM is a geometric equation, meaning that the solutions to the induced EOM given an initial condition is a set of trajectories that trace out the same path, hence producing time re-parameterization invariant trajectories. $\mathcal{L}_g$'s EOM being a geometric equation can be intuitively shown by proving that $\mathcal{L}_g$'s action functional is time re-parameterization invariant itself [1]. Given a time re-parameterization $t = t(s)$ with $\mathbf{x}_s(s) = \mathbf{x}_t(t(s))$ and $\dot{\mathbf{x}}_s = \frac{dt}{ds}\dot{\mathbf{x}}_t$, $\mathcal{L}_g$'s action integral $A[\mathbf{x}_s]$ can be shown to equal $A[\mathbf{x}_t]$, hence being invariant to time re-parameterization:

$$A[\mathbf{x}_s] = \int \mathcal{L}_g(\mathbf{x}_s, \dot{\mathbf{x}}_s)\, ds = \mathcal{L}_g\left(\mathbf{x}_t, \frac{dt}{ds}\dot{\mathbf{x}}_t\right) ds \tag{2.39}$$

$$= \int \mathcal{L}_g(\mathbf{x}_t, \dot{\mathbf{x}}_t)\frac{dt}{ds}\, ds \tag{2.40}$$

$$= \int \mathcal{L}_g(\mathbf{x}_t, \dot{\mathbf{x}}_t)dt = A[\mathbf{x}_t]. \tag{2.41}$$

2.1.7.2 Finsler Energy

We now investigate Finsler energy $\mathcal{L}_e(\mathbf{x}, \dot{\mathbf{x}})$, which is a Lagrangian that follows $\mathcal{L}_e = \frac{1}{2}\mathcal{L}_g^2$. We can immediately observe that $\mathcal{L}_e$ is HD2 in $\dot{\mathbf{x}}$ since $\mathcal{L}_g$ is HD1 in $\dot{\mathbf{x}}$:

$$\mathcal{L}_e(\mathbf{x}, \lambda\dot{\mathbf{x}}) = \frac{1}{2}\left(\mathcal{L}_g(\mathbf{x}, \lambda\dot{\mathbf{x}})\right)^2 \tag{2.42}$$

$$= \frac{1}{2}\left(\lambda\mathcal{L}_g(\mathbf{x}, \dot{\mathbf{x}})\right)^2 \tag{2.43}$$

$$= \lambda^2\mathcal{L}_e(\mathbf{x}, \dot{\mathbf{x}}) \tag{2.44}$$

From Euler's theorem on homogeneous functions [17], it is known that if $\mathcal{L}(\mathbf{x}, \dot{\mathbf{x}})$ is HDk in $\dot{\mathbf{x}}$, then its corresponding Hamiltonian $\mathcal{H}$ is:

$$\mathcal{H} = \partial_{\dot{x}}\mathcal{L}^T\dot{\mathbf{x}} - \mathcal{L} = (k-1)\mathcal{L} \tag{2.45}$$

[1] Recall from Calculus of Variations, solutions to the EOM derived from taking the Euler-Lagrange equation of a Lagrangian are the stationary solutions that minimize the Lagrangian's action functional.

In the case of $\mathcal{L}_e(\mathbf{x}, \dot{\mathbf{x}})$, since it is HD2, its Hamiltonian is:

$$\mathcal{H}_e = (2-1)\mathcal{L}_e = \mathcal{L}_e \tag{2.46}$$

Therefore, $\mathcal{L}_e$'s Hamiltonian is itself. This is why $\mathcal{L}_e$ is called the Finsler *energy* since $\mathcal{L}_e$ also happens to equal to the system's energy.

We now claim that the EOM derived from taking the Euler-Lagrange equation of $\mathcal{L}_e$ is a geometry generator equation, where its corresponding geometric equation is the EOM of the associated $\mathcal{L}_g$. The EOM induced from $\mathcal{L}_e$ is the following:

$$\mathbf{M}_\mathrm{e}\ddot{\mathbf{x}} + \mathbf{f}_\mathrm{e} = \mathbf{0} \tag{2.47}$$

where $\mathbf{M}_\mathrm{e} = \partial^{\mathbf{2}}_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_\mathrm{e}$ and $\mathbf{f}_e = \partial_{\dot{\mathbf{x}}\mathbf{x}}\mathcal{L}_\mathrm{e}\dot{\mathbf{x}} - \partial_{\mathbf{x}}\mathcal{L}_\mathrm{e}$. It can be shown that $\mathbf{M}_\mathrm{e}$ is HD0 in $\dot{\mathbf{x}}$ and $\mathbf{f}_e$ is HD2 in $\dot{\mathbf{x}}$. We note that since Equation (2.47) is derived from a Lagrangian $\mathcal{L}_e$, it is also a conservative fabric, meaning that the system conserves energy; in this case, the conserved energy is $\mathcal{H}_e$, which is the Finsler energy $\mathcal{L}_e$ itself.

To summarize, the EOM of a Finsler structure $\mathcal{L}_g$ is a geometric equation and the EOM of the associated Finsler energy $\mathcal{L}_e$ forms a conservative fabric that conserves $\mathcal{L}_e$ while also being a geometry generator.

2.1.8 Geometric Fabrics

An energized fabric, $\text{energize}_{\mathcal{L}_e}\{(\mathbf{I}, \mathbf{h}_2)_\mathcal{X}\}$, is a *geometric fabric* if $(\mathbf{I}, \mathbf{h}_2)_\mathcal{X}$ is a geometry generator and $\mathcal{L}_e$ is a Finsler energy. Note that if $(\mathbf{I}, \mathbf{h}_2)_\mathcal{X}$ is a geometry generator, $\mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}})$ must be HD2 in $\dot{\mathbf{x}}$, hence we denote it as $\mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}})$ explicitly. We now explicitly present the canonical form (2.48) and the nominal form (2.49) of the geometric fabric equation:

$$\text{energize}_{\mathcal{L}_e}\{(\mathbf{I}, -\mathbf{h}_2)^\mathcal{C}_\mathcal{X}\} = (\mathbf{M}_e, -\mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}})^\mathcal{C}_\mathcal{X} \tag{2.48}$$

$$\text{energize}_{\mathcal{L}_e}\{(\mathbf{I}, \mathbf{h}_2)_\mathcal{X}\} = (\mathbf{M}_e, \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h}_2 - \mathbf{f}_e])_\mathcal{X} \tag{2.49}$$

where $\mathbf{M}_\mathrm{e} = \partial^{\mathbf{2}}_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_\mathrm{e}$ and $\mathbf{f}_e = \partial_{\dot{\mathbf{x}}\mathbf{x}}\mathcal{L}_\mathrm{e}\dot{\mathbf{x}} - \partial_{\mathbf{x}}\mathcal{L}_\mathrm{e}$.

We note that the canonical form of the geometric fabric equation (2.48) represents the differential equation $\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}} = \mathbf{0}$. Since $\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}$ is a geometry generator and the term

$-\alpha_{\mathcal{H}_e}\dot{\mathbf{x}}$ is merely an instantaneous acceleration along the direction of motion $\dot{\mathbf{x}}$, solutions to the geometric fabric $\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}} = \mathbf{0}$ trace out the same paths as solutions to $\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{0}$, given the same initial conditions. In other words, the nominal behaviour of the geometric fabric, energize$_{\mathcal{L}_e}\{(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}\}$, matches the intended nominal behaviour specified by $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$, while maintaining a stable energy level of $\mathcal{H}_e = \mathcal{L}_e$. To emphasize, by specifying the nominal behaviour via a geometry generator $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$, we can ensure that energizing $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$ does not modified the intended nominal behaviour.

Furthermore, since $\mathcal{L}_e$ is a Finsler energy, $\mathbf{M}_\mathrm{e}$ is HD0 in $\dot{\mathbf{x}}$ and $\mathbf{f}_e$ is HD2 in $\dot{\mathbf{x}}$. Using these properties, it is simple to show that $-\alpha_{\mathcal{H}_e}\dot{\mathbf{x}}$ is HD2 in $\dot{\mathbf{x}}$. This means that $\mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}}$ is also HD2 in $\dot{\mathbf{x}}$. Hence, the geometric fabric $\ddot{\mathbf{x}} + \mathbf{h}_2(\mathbf{x}, \dot{\mathbf{x}}) - \alpha_{\mathcal{H}_e}\dot{\mathbf{x}} = \mathbf{0}$ is a geometry generator itself. In fact, the geometric fabric, energize$_{\mathcal{L}_e}\{(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}\}$, is a geometry generator that has the same geometry as $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$, thus exhibiting the same nominal behaviour.

We now investigate the nominal form of the geometric fabric equation (2.49), representing the differential equation $\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h}_2 - \mathbf{f}_e] = \mathbf{0}$. We note that this differential equation is essentially the EOM of the Finsler energy $\mathcal{L}_e$ with an additional zero-work modification term $\mathbf{P}_e[\mathbf{M}_e\mathbf{h}_2 - \mathbf{f}_e]$ from the energization process. Recall that the EOM of a Finsler energy $\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e = \mathbf{0}$ is a geometric generator itself, thus denoted as the Finsler geometry. However, the canonical form of the geometric fabric equation (2.48) clearly indicates that geometric fabric actually exhibits the geometry of $\ddot{\mathbf{x}} + \mathbf{h}_2 = \mathbf{0}$, which differs from the geometry of $\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e = \mathbf{0}$. Therefore, the additional zero-work modification term can be viewed as bending the Finsler geometry to match the intended geometry $\ddot{\mathbf{x}} + \mathbf{h}_2 = \mathbf{0}$ without changing the system's energy.

In addition, from $\mathbf{M}_e\ddot{\mathbf{x}} + \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h}_2 - \mathbf{f}_e] = \mathbf{0}$, it can be seen that the geometric fabric's metric is the same as the metric $\mathbf{M}_e$ from the unbent Finsler geometry. Therefore, bending the Finsler geometry does not affect the fabric's metric, but it allows its nominal behaviour to completely adhere to another geometry $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$, effectively enabling one to decouple the design of the fabric's metric from its nominal behaviour.

We summarize the main properties of a geometric fabric, denoted as energize$_{\mathcal{L}_e}\{(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}\}$, as follows:

1. $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$ solely specifies the nominal behaviour of the geometric fabric.

2. Energizing $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$ using any arbitrary $\mathcal{L}$ ensures that $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$ is stable with an energy level of $\mathcal{H}_e$ without modifying its nominal behaviour.

3. Energizing $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$ using a Finsler energy $\mathcal{L}_e$, ensures that the resulting fabric is stable with an energy level of $\mathcal{H}_e = \mathcal{L}_e$. In addition, the resulting fabric is a geometry generator that exhibits the same geometry as $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$.

4. The selection of the Finsler energy $\mathcal{L}_e$ determines the fabric's metric via $\mathbf{M}_{\mathrm{e}} = \partial^{\mathbf{2}}_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_{\mathrm{e}}$. This metric is used to determine the priority of the geometry and/or the forcing potential when combining with other fabric policies via the spec combination operation (see Section 2.1.1.2).

5. Since the metric is solely dependent on $\mathcal{L}_e$ and the nominal behaviour is solely specified by $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$, the design of the metric is completely decoupled from the fabric's nominal behaviour, resulting in more expressivity in the fabric.

2.1.9 Geometric Fabrics and The Spec Algebra

Geometric fabrics are fundamentally a class of specs and hence the pullback and combination operations described in the Section 2.1.1 hold for geometric fabrics. Therefore, the spec algebra can be leveraged to design geometric fabrics in parts, where each part specifies a certain behaviour in a particular task space; each of individual the parts can then be pulled-back to a common task space and combined to yield a final geometric fabric. For example, in the case of a manipulator, one may define a geometric fabric in the joint space for joint-space posing and a geometric fabric for each of the robot links for obstacle-avoidance. It is worth noting that any pullbacks and combinations of geometric fabrics still result in a geometric fabric. In fact, under the spec algebra, closure holds for any of the aforementioned classes of fabrics [34]. In this section, we further explore the spec algebra as we outline the ways in which Lagrangians and the energization operation behave under the spec algebra.

2.1.9.1 Lagrangians Under Pullback

Given a differentiable map $\phi : \mathcal{Q} \to \mathcal{X}$, for a Lagrangian $\mathcal{L}(\mathbf{x}, \dot{\mathbf{x}})$, we can pullback this Lagrangian from $\mathcal{X}$ to $\mathcal{Q}$ as follows:

$$\mathrm{pull}_{\phi}[\mathcal{L}] = \mathcal{L}(\phi(\mathbf{q}), \frac{d}{dt}\phi(\mathbf{q})) \tag{2.50}$$

With this definition, the following property holds:

$$\mathrm{pull}_{\phi}\left[\mathrm{EOM}[\mathcal{L}(\mathbf{x}, \dot{\mathbf{x}})]\right] = \mathrm{EOM}\left[\mathrm{pull}_{\phi}[\mathcal{L}(\mathbf{x}, \dot{\mathbf{x}})]\right] \tag{2.51}$$

where $\text{EOM}[\mathcal{L}(\mathbf{x}, \dot{\mathbf{x}})] = (\mathbf{M}, \mathbf{f})_{\mathcal{X}} = \mathbf{M}\ddot{\mathbf{x}} + \mathbf{f} = \mathbf{0}$ denotes the equation of motion derived from taking the Euler-Lagrange equation of the Lagrangian $\mathcal{L}$.

2.1.9.2 Energization Under Pullback

Given a differentiable map $\phi : \mathcal{Q} \to \mathcal{X}$, a geometry generator defined in $\mathcal{X}$: $(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}$, and a Finsler energy defined in $\mathcal{X}$: $\mathcal{L}_e$, the following property holds:

$$\text{pull}_{\phi}\left[\text{energize}_{\mathcal{L}_e}(\mathbf{I}, \mathbf{h}_2)_{\mathcal{X}}\right] = \text{energize}_{\text{pull}_{\phi}\mathcal{L}_e}\left[\text{pull}_{\phi}[(\mathbf{M}_e, \mathbf{M}_e\mathbf{h}_2)_{\mathcal{X}}]\right] \tag{2.52}$$

where $\mathbf{M}_{\text{e}} = \partial^{\mathbf{2}}_{\dot{\mathbf{x}}\dot{\mathbf{x}}}\mathcal{L}_{\text{e}}$. In essence, energizing a geometry with $\mathcal{L}_e$ in $\mathcal{X}$, then pulling back to $\mathcal{Q}$ is equivalent to first pulling back to $\mathcal{Q}$ with the metric $\mathbf{M}_e$, then energizing with $\text{pull}_{\phi}\mathcal{L}_e$. Therefore, energization commutes with pullback.

2.1.9.3 Energization Under Combination

For $i \in [1, m]$, we have m geometry generators $(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}}$, and m associated Finsler energies $\mathcal{L}_{e,i}$. The following property holds:

$$\sum_i \left(\text{energize}_{\mathcal{L}_{e,i}}\left[(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}}\right]\right) = \text{energize}_{\sum_i \mathcal{L}_{e,i}}\left[\sum_i (\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}}\right] \tag{2.53}$$

Equation (2.53) states that energizing $(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}}$ with their individual $\mathcal{L}_{e,i}$ then combining them is equivalent to combining $(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}}$ first and then energizing the combined geometry generator with the sum of their $\mathcal{L}_{e,i}$.

2.1.9.4 Geometric Fabrics on a Transform Tree

We now consider a transform tree where there exists m differentiable maps $\phi_i : \mathcal{Q} \to \mathcal{X}_i$. Each task space $\mathcal{X}_i$ has a geometry specified by: $(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}_i}$, and an associated Finsler energy $\mathcal{L}_{e,i}$. We desire to yield a combined geometric fabric in $\mathcal{Q}$, which consists of the following steps:

1. Energizing each $(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}_i}$ with $\mathcal{L}_{e,i}$, forming m geometric fabrics.
2. Pulling back each of the geometric fabrics from $\mathcal{X}_i$ to $\mathcal{Q}$.
3. Combining the pulled-back geometric fabrics to yield a final geometric fabric in $\mathcal{Q}$.

However, energizing each geometry individually can be computationally inefficient. Instead, we can leverage the following property to only perform energization once:

$$\sum_i \left(\text{pull}_{\phi_i}\left[\text{energize}_{\mathcal{L}_{e,i}}\left[(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}_i}\right]\right]\right) = \text{energize}_{\sum_i \text{pull}_{\phi_i}\mathcal{L}_{e,i}}\left[\sum_i \text{pull}_{\phi_i}(\mathbf{M}_{e,i}, \mathbf{M}_{e,i}\mathbf{h}_{2,i})_{\mathcal{X}_i}\right] \tag{2.54}$$

Therefore, in practice, deriving the combined geometric fabric consists of the following steps:

1. Pulling back each of the geometries from $\mathcal{X}_i$ to $\mathcal{Q}$ with the metric $\mathbf{M}_{e,i}$ derived from $\mathcal{L}_{e,i}$.
2. Combining the geometries to yield a final geometry in $\mathcal{Q}$.
3. Energize the combined geometry with the sum of the pulled-back $\mathcal{L}_{e,i}$ to yield the final geometric fabric in $\mathcal{Q}$.

We now investigate Step 1 and 2 further by expanding out the pullback operations, then expressing the pulled-back specs in their canonical form, and finally performing combination in the canonical form:

$$\begin{aligned}
\sum_i \text{pull}_{\phi_i}(\mathbf{M}_{e,i}, \mathbf{M}_{e,i}\mathbf{h}_{2,i})_{\mathcal{X}_i} &= \sum_i \left(\mathbf{J}_i^T\mathbf{M}_{e,i}\mathbf{J}_i, \mathbf{J}_i^T(\mathbf{M}_{e,i}\mathbf{h}_{2,i} + \mathbf{M}_{e,i}\dot{\mathbf{J}}_i\dot{\mathbf{q}})\right)_{\mathcal{Q}} && (2.55)\\
&= \sum_i \left(\widetilde{\mathbf{M}}_i, -\widetilde{\mathbf{M}}_i^\dagger\mathbf{J}_i^T\mathbf{M}_{e,i}(\mathbf{h}_{2,i} + \dot{\mathbf{J}}_i\dot{\mathbf{q}})\right)_{\mathcal{Q}}^{\mathcal{C}} && (2.56)\\
&= \sum_i \left(\widetilde{\mathbf{M}}_i, -\widetilde{\mathbf{h}}_{2,i}\right)_{\mathcal{Q}}^{\mathcal{C}} && (2.57)\\
&= \left(\sum_{i=1}^m \widetilde{\mathbf{M}}_i, -\left(\sum_{i=1}^m \widetilde{\mathbf{M}}_i\right)^{-1}\sum_{i=1}^m \widetilde{\mathbf{M}}_i\widetilde{\mathbf{h}}_{2,i}\right)_{\mathcal{Q}}^{\mathcal{C}} && (2.58)
\end{aligned}$$

where $\widetilde{\mathbf{M}}_i = \mathbf{J}_i^T\mathbf{M}_{e,i}\mathbf{J}_i$, representing the pulled-back metrics, and $\widetilde{\mathbf{h}}_{2,i} = \widetilde{\mathbf{M}}_i^\dagger\mathbf{J}_i^T\mathbf{M}_{e,i}(\mathbf{h}_{2,i} + \dot{\mathbf{J}}_i\dot{\mathbf{q}})$, representing the pulled-back geometries.

We now define $\widetilde{\mathbf{h}}_2 = \left(\sum_{i=1}^m \widetilde{\mathbf{M}}_i\right)^{-1}\sum_{i=1}^m \widetilde{\mathbf{M}}_i\widetilde{\mathbf{h}}_{2,i}$ such that $\ddot{\mathbf{q}} + \widetilde{\mathbf{h}}_2 = \mathbf{0}$, or $\left(\mathbf{I}, \widetilde{\mathbf{h}}_2\right)_{\mathcal{Q}}$, represents the combined geometry in $\mathcal{Q}$. It is worth noting that Equation (2.58) clearly demonstrates that each pulled-back geometry $\widetilde{\mathbf{h}}_{2,i}$ is weighted by the pulled-back metric $\widetilde{\mathbf{M}}_i$; therefore, we can design the metric to modulate the priority of each geometry.

For Step 3, we first compute the system's combined Finsler energy $\mathcal{L}_e$ as follows:

$$\mathcal{L}_e = \sum_i \text{pull}_{\phi_i}\mathcal{L}_{e,i} \tag{2.59}$$

Then we energize the combined geometry $\left(\mathbf{I}, \widetilde{\mathbf{h}}_2\right)_{\mathcal{Q}}$ with $\mathcal{L}_e$ to yield the final geometric fabric in $\mathcal{Q}$: $\text{energize}_{\mathcal{L}_e}\left(\mathbf{I}, \widetilde{\mathbf{h}}_2\right)_{\mathcal{Q}}$.

2.1.9.5 Forcing Geometric Fabrics

Given a geometric fabric $\text{energize}_{\mathcal{L}_e}\{(\mathbf{I}, \mathbf{h}_2)_{\mathcal{Q}}\} = (\mathbf{M}_e, \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h}_2 - \mathbf{f}_e])_{\mathcal{Q}}$, such as the final combined geometric fabric described in Section 2.1.9.4, it is often useful to add a potential function $\psi(\mathbf{q})$ to the system to inject energy into the system in order to achieve some task-specific behaviour. Since the geometric fabric alone specifies the system's nominal behaviour, the potential hence serves to push the system away from its nominal behaviour. For instance, in the case of a manipulator, a well designed geometric fabric could exhibit the nominal behaviour of joint-space posing and obstacle avoidance; in order to achieve end-effector posing, one may impose an end-effector attractor potential on the geometric fabric. It is also common to add a damper $-\mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ that can drain the system's energy in order for the system to converge. A geometric fabric in the presence of a potential and a damper is referred as a forced geometric fabric, which has the following form:

$$\mathbf{M}_e\ddot{\mathbf{q}} + \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h}_2 - \mathbf{f}_e] = -\partial_{\mathbf{q}}\psi - \mathbf{B}\dot{\mathbf{q}} \tag{2.60}$$

One can also design multiple potentials in different task spaces and combine them in one common task space. Given a transform tree with n differentiable maps $\phi_i : \mathcal{Q} \to \mathcal{X}_i$. Each task space has a potential function $\psi_i(\mathbf{x}_i)$ and an associated metric $\mathbf{M}_{\psi,i}$ that can be used to weigh its potential against other potentials for priority modulation. For each potential and metric, we can construct the following spec $(\mathbf{M}_{\psi,i}, \mathbf{M}_{\psi,i}\partial_{\mathbf{x}_i}\psi_i(\mathbf{x}_i))_{\mathcal{X}_i}$, representing the following differential equation:

$$\mathbf{M}_{\psi,i}\ddot{\mathbf{x}}_i = -\mathbf{M}_{\psi,i}\partial_{\mathbf{x}_i}\psi_i(\mathbf{x}_i) \tag{2.61}$$

We can then use the spec algebra to pullback each potential to $\mathcal{Q}$ and combine them as such:

$$\sum_i \text{pull}_{\phi_i}\left(\mathbf{M}_{\psi,i}, \mathbf{M}_{\psi,i}\partial_{\mathbf{x}_i}\psi_i\right) = \sum_i \left(\mathbf{J}_i^T\mathbf{M}_{\psi,i}\mathbf{J}_i, \mathbf{J}_i^T(\mathbf{M}_{\psi,i}\partial_{\mathbf{x}_i}\psi_i + \mathbf{M}_{\psi,i}\dot{\mathbf{J}}_i\dot{\mathbf{q}})\right)_{\mathcal{Q}} \tag{2.62}$$

$$= \sum_i \left(\widetilde{\mathbf{M}}_{\psi,i}, -\widetilde{\mathbf{M}}_{\psi,i}^{\dagger}\mathbf{J}_i^T\mathbf{M}_{\psi,i}(\partial_{\mathbf{x}_i}\psi_i + \dot{\mathbf{J}}_i\dot{\mathbf{q}})\right)_{\mathcal{Q}}^{\mathcal{C}} \tag{2.63}$$

$$= \sum_i \left(\widetilde{\mathbf{M}}_{\psi,i}, -\partial\widetilde{\psi_i}\right)_{\mathcal{Q}}^{\mathcal{C}} \tag{2.64}$$

$$= \left(\sum_{i=1}^m \widetilde{\mathbf{M}}_{\psi,i}, -\left(\sum_{i=1}^m \widetilde{\mathbf{M}}_{\psi,i}\right)^{-1}\sum_{i=1}^m \widetilde{\mathbf{M}}_{\psi,i}\partial\widetilde{\psi_i}\right)_{\mathcal{Q}}^{\mathcal{C}} \tag{2.65}$$

where $\widetilde{\mathbf{M}}_{\psi,i} = \mathbf{J}_i^T\mathbf{M}_{\psi,i}\mathbf{J}_i$ and $\partial\widetilde{\psi_i} = \widetilde{\mathbf{M}}_{\psi,i}^{\dagger}\mathbf{J}_i^T\mathbf{M}_{\psi,i}(\partial_{\mathbf{x}_i}\psi_i + \dot{\mathbf{J}}_i\dot{\mathbf{q}})$. We can therefore set $-\partial_{\mathbf{q}}\psi$ in Equation (2.60) as $-\left(\sum_{i=1}^m \widetilde{\mathbf{M}}_{\psi,i}\right)^{-1}\sum_{i=1}^m \widetilde{\mathbf{M}}_{\psi,i}\partial\widetilde{\psi_i}$.

2.2 Reinforcement Learning

The objective of reinforcement learning (RL) is to maximize an agent's expected discounted return via learning from its interactions with its environment, commonly modelled as a Markov Decision Process (MDP) [26]. An MDP is denoted as $\langle S, A, R, P, \rho_0 \rangle$, where

- S represents the set of all valid states.
- A represents the set of all valid actions.
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, with $r_t = R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$.
- $P : S \times A \to \mathcal{P}(S)$ is the state transition probability, with $P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ representing the probability of being in $\mathbf{s}'$ starting from $\mathbf{s}$ and taking action $\mathbf{a}$.
- ρ_0 represents the distribution of the starting state $\mathbf{s}_0$.

The agent's interactions with its environment are organized into episodes. At the beginning of each episode, the agent starts in $\mathbf{s}_0 \in S$, sampled according to $\mathbf{s}_0 \sim \rho_0(\mathbf{s}_0)$. At each proceeding time step t, the agent observes the state $\mathbf{s}_t \in S$ from its environment and takes an action $\mathbf{a}_t \in A$ sampled from its policy $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$. Given this action $\mathbf{a}_t$, the environment transitions to a new state $\mathbf{s}_{t+1} \in S$ at the next time step, sampled according to the state transition probability $\mathbf{s}_{t+1} \sim P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. In addition, the agent also receives a reward $r_t = R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ for its behaviour. The agent repeatedly observes its environment and applies actions for T time steps, which forms a sequence of states and actions, denoted as a trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T)$. Given a trajectory τ, the associated discounted return is calculated as follows:

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t \tag{2.66}$$

where $\gamma \in (0, 1)$ is the reward discount factor. The objective of RL is to learn an optimal policy π^* that maximizes the expected discounted return $J(\pi)$, defined as follows:

$$\pi^* = \arg\max_{\pi} J(\pi) \tag{2.67}$$

$$J(\pi) = \int_{\tau} P(\tau|\pi) R(\tau) = \mathbb{E}_{\tau \sim \pi}[R(\tau)] \tag{2.68}$$

$$P(\tau|\pi) = \rho_0(\mathbf{s}_0) \prod_{t=0}^{T-1} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t|\mathbf{s}_t) \tag{2.69}$$

where $P(\tau|\pi)$ is the probability of a T-step trajectory τ induced by the policy π.

2.2.1 Value Functions

In reinforcement learning, it is useful to have a means of estimating the current performance of the agent. Value functions serve this purpose as it provides the expected return given the current state or state-action pair [42].

The on-policy value function $V^\pi(\mathbf{s})$ gives the expected return of an agent at state $\mathbf{s}$ and act according to π for T time steps, defined as follows:

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\tau \sim P(\tau|\pi, \mathbf{s}_0 = \mathbf{s})} \left[R(\tau) \right] \tag{2.70}$$

$$P(\tau|\pi, \mathbf{s}_0 = \mathbf{s}) = \prod_{t=0}^{T-1} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t) \tag{2.71}$$

Similarly, the on-policy action-value function $Q^\pi(\mathbf{s}, \mathbf{a})$ gives the expected return of an agent applying action $\mathbf{a}$ from state $\mathbf{s}$ and following π for T time steps, defined as follows:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\tau \sim P(\tau|\pi, \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a})} \left[R(\tau) \right] \tag{2.72}$$

$$P(\tau|\pi, \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}) = P(\mathbf{s}_1|\mathbf{s}_0, \mathbf{a}_0) \prod_{t=1}^{T-1} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t) \tag{2.73}$$

2.2.2 Policy Gradient Methods

Policy gradient methods are a class of reinforcement learning algorithms that are well-suited for dynamic robot control tasks due to their compatibility with continuous action spaces and their capacity to leverage neural networks as powerful function approximators [26, 43]. Policy gradient methods optimize the policy's parameters via directly performing gradient ascent [2] on the expected discounted return objective $J(\pi_\theta)$:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_k} \tag{2.74}$$

Therefore, policy gradient methods rely on obtaining good estimates of the policy gradient $\nabla_\theta J(\pi_\theta)$, which we will discuss in this section.

[2] Any gradient-based optimization methods, such as Adam [14], can be used.

We now derive an analytical expression for $\nabla_\theta J(\pi_\theta)$. We first apply logarithm to Equation (2.69) to obtain the log-probability of a trajectory:

$$\log P(\tau|\theta) = \log \rho_0(\mathbf{s}_0) + \sum_{t=0}^{T-1} \left(\log P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) + \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right) \tag{2.75}$$

Since only the policy π_θ is dependent on θ, the gradients of $\log \rho_0(\mathbf{s}_0)$ and $\log P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ with respect to θ is 0. Thus, we obtain the following expression for $\nabla_\theta \log P(\tau|\theta)$:

$$\nabla_\theta \log P(\tau|\theta) = \cancel{\nabla_\theta \log \rho_0(\mathbf{s}_0)} + \sum_{t=0}^{T-1} \Big(\cancel{\nabla_\theta \log P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} + \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\Big) \tag{2.76}$$

$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \tag{2.77}$$

We also note that $\nabla_\theta \log P(\tau|\theta)$ can be written as follows:

$$\nabla_\theta \log P(\tau|\theta) = \frac{1}{P(\tau|\theta)} \nabla_\theta P(\tau|\theta) \tag{2.78}$$

Rearranging Equation (2.78) for $\nabla_\theta P(\tau|\theta)$ yields and substituting $\nabla_\theta \log P(\tau|\theta)$ in accordance with Equation (2.77) yields:

$$\nabla_\theta P(\tau|\theta) = P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) \tag{2.79}$$

$$= P(\tau|\theta) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \tag{2.80}$$

We finally derive the analytical expression for the policy gradient $\nabla_\theta J(\pi_\theta)$. Note that since the return $R(\tau)$ is independent of θ, $\nabla_\theta R(\tau) = 0$.

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \tag{2.81}$$

$$= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \tag{2.82}$$

$$= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) + \cancel{P(\tau|\theta)\nabla_\theta R(\tau)} \tag{2.83}$$

$$= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) \tag{2.84}$$

$$= \int_\tau P(\tau|\theta) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)R(\tau) \tag{2.85}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}\left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right) R(\tau)\right] \tag{2.86}$$

Equation (2.86) indicates that the policy gradient $\nabla_\theta J(\pi_\theta)$ can be expressed as an expectation, which means that we can obtain an estimate of the policy gradient $\hat{g}$ via Monte Carlo sampling. Specifically, after collecting a dataset of N trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ via rolling out the policy π_θ, we can calculate a policy gradient estimate as follows:

$$\hat{g} = \frac{1}{N} \sum_{\tau\in\mathcal{D}} \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right) R(\tau)\right] \tag{2.87}$$

In practice, calculating the policy gradient estimate according to Equation (2.87) results in estimates with high variance, which means that we need to collect a high number of trajectories to ensure an accurate estimate of the policy gradient. To reduce the number of trajectories needed, we can reduce the variance of the policy gradient estimate as such:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right) R(\tau)\right] \tag{2.88}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} (\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)R(\tau))\right] \tag{2.89}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \left(\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{R}_t\right)\right] \tag{2.90}$$

where $\hat{R}_t = \sum_{t'=t}^{T-1} \gamma^{t'} R(\mathbf{s}_{t'}, \mathbf{a}_{t'}, \mathbf{s}_{t'+1})$ is the *rewards-to-go*, which is the sum of rewards left to be obtained in the trajectory from a given time step. Intuitively, Equation (2.90) states that $\nabla_\theta J(\pi_\theta)$ is a weighted-sum of $\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, where the weight is the rewards-to-go. $\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ represents the direction of change for θ that would increase the probability of the policy outputting $\mathbf{a}_t$ given $\mathbf{s}_t$.

Therefore, if the rewards-to-go is large for a given $\mathbf{s}_t$, the policy gradient would be more weighted towards a direction that would increase the probability of the policy outputting the associated $\mathbf{a}_t$ and vice versa.

One may further reduce variance by replacing the empirical rewards-to-go $\hat{R}_t$ with the on-policy action-value function $Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)$ in Equation (2.90), resulting in the following:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{2.91}$$

Moreover, we can introduce a baseline $b(\mathbf{s}_t)$ without changing the the gradient estimate [26]:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \left(Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) - b(\mathbf{s}_t) \right) \right] \tag{2.92}$$

A common baseline we use is the on-policy value function $V^{\pi_\theta}(\mathbf{s}_t)$ due to its empirical effect of reducing the variance of the policy gradient estimate [26]:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \left(Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) - V^{\pi_\theta}(\mathbf{s}_t) \right) \right] \tag{2.93}$$

In practice, $V^{\pi_\theta}(\mathbf{s}_t)$ cannot be computed exactly. However, we can approximate $V^{\pi_\theta}(\mathbf{s}_t)$ with a neural network $V_\phi(\mathbf{s}_t)$. For each each policy update via gradient ascent, we also train $V_\phi(\mathbf{s}_t)$ as follows:

$$\phi_{k+1} = \arg\min_\phi \mathbb{E}_{\mathbf{s}_t, \hat{R}_t \sim \pi_k} \left[\left(V_\phi(\mathbf{s}_t) - \hat{R}_t \right)^2 \right] \tag{2.94}$$

Using Monte Carlo sampling, we can express Equation (2.94) as the following:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{N_k T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T-1} \left(V_\phi(\mathbf{s}_t) - \hat{R}_t \right)^2 \tag{2.95}$$

We also note that $Q^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) - V^{\pi_\theta}(\mathbf{s}_t)$ in Equation (2.93) is denoted as the advantage function $A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)$, which can be interpreted as a measurement of how much better the current action $\mathbf{a}_t$ is in comparison to the average action from the current policy at $\mathbf{s}_t$. In practice, we estimate $A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)$ with a method called the Generalized Advantage Estimation (GAE), which involves using the value function estimate $V_\phi(\mathbf{s}_t)$ and the collected rewards to yield an advantage estimate $\hat{A}_t^{GAE}$ in a manner that allows the user to control the bias-variance trade-off of the estimate [38].

$\hat{A}_t^{GAE}$ is computed as follows:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l}^V \tag{2.96}$$

$$\delta_t^V = r_t + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t) \tag{2.97}$$

where $\gamma \in (0, 1)$ is the reward discount factor. $\lambda \in (0, 1)$ is a tunable parameter that controls the bias-variance trade off of the advantage estimate, where $\lambda \to 0$ has low variance but high bias and $\lambda \to 1$ has low bias but high variance.

The policy gradient and its Monte Carlo estimate can thus be expressed as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}_t^{GAE}\right] \tag{2.98}$$

$$\hat{g} = \frac{1}{N}\sum_{\tau\in\mathcal{D}}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}_t^{GAE}\right] \tag{2.99}$$

In practice, instead of following Equation (2.99) for optimizing the policy and Equation (2.95) for optimizing the value function estimate, we can optimize both simultaneously. We denote $L_t(\theta) = \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}_t^{GAE}$ such that its gradient $\nabla_\theta L_t = \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}_t^{GAE}$. Additionally, we denote $L_t^{VF}(\phi) = \left(V_\phi(\mathbf{s}_t) - \hat{R}_t\right)^2$. We then define an overall objective function $L(\theta, \phi)$ as follows:

$$L(\theta,\phi) = \frac{1}{N}\sum_{\tau\in\mathcal{D}}\left[\sum_{t=0}^{T-1} L_t(\theta) - c_1 L_t^{VF}(\phi)\right] \tag{2.100}$$

$$= \frac{1}{N}\sum_{\tau\in\mathcal{D}}\left[\sum_{t=0}^{T-1} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}_t^{GAE} - c_1\left(V_\phi(\mathbf{s}_t) - \hat{R}_t\right)^2\right] \tag{2.101}$$

Using an auto-differentiation framework and a gradient-based optimizer, we can maximize L with respect to θ and ϕ, which would optimize both the policy π_θ and the value function estimate V_ϕ.

We present the RL training pipeline for the vanilla policy gradient method in Algorithm 1.

Algorithm 1: Vanilla Policy Gradient Algorithm

1 $\pi_\theta \leftarrow$ Initialize policy with θ_0.

2 $V_\phi \leftarrow$ Initialize value function estimate with ϕ_0.

3 **for** $k = 0, 1, 2, \ldots$ **do**

4 Collect a set of trajectories $\mathcal{D}_k = \{\tau_i\}$ with $\pi_\theta\big|_{\theta_k}$ in the environment.

5 Compute rewards-to-go $\hat{R}_t$.

6 Compute advantage estimates $\hat{A}_t^{GAE}$ using GAE.

7 Compute the overall objective function $L(\theta, \phi)$ and maximize L using a gradient-based optimizer:

$$L(\theta, \phi) = \frac{1}{N} \sum_{\tau \in \mathcal{D}} \left[\sum_{t=0}^{T-1} L_t(\theta) - c_1 L_t^{VF}(\phi) \right]$$

$$\theta_{k+1}, \phi_{k+1} = \arg\max_{\theta,\phi} L(\theta, \phi)$$

8 **end**

We note that in Line 7 of Algorithm 1, we ideally only want to perform one gradient update on π_θ because the trajectory dataset $\mathcal{D}_k$ used to compute the policy gradient was collected from the current policy $\pi_\theta\big|_{\theta_k}$. As soon as π_θ receives a gradient update, the existing dataset $\mathcal{D}_k$ is technically no longer representative of the policy's trajectory distribution. If we were to use $\mathcal{D}_k$ to perform multiple gradient updates on the policy without collecting new data, the policy gradient estimates would become more and more inaccurate. In spite of this issue, we would want to perform more than one gradient updates in practice; this is to accelerate the training process since collecting a new trajectory per policy update is prohibitively time-consuming.

2.2.3 Proximal Policy Optimization

The vanilla policy gradient algorithm described in Algorithm 1 often result in unstable training, where the performance of the policy after receiving gradient updates would degrade catastrophically [37]. This problem occurs because when multiple gradient updates are performed without collecting new trajectory datasets, the policy gradient becomes inaccurate, as discussed above. Even a small sized inaccurate gradient update could collapse the performance of the policy, which makes it dangerous to use large step sizes in policy gradient updates, hence resulting in poor sample efficiency. To alleviate this issue, Proximal Policy Optimization (PPO) is a method that constrains the size of gradient updates to prevent catastrophic forgetting, thus improving the training stability [39].

PPO first replaces $L_t(\theta) = \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\hat{A}_t$ from the vanilla policy gradient algorithm with $L_t^{CPI}(\theta, \theta_k)$, defined as follows:

$$L_t^{CPI}(\theta, \theta_k) = \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_k}(\mathbf{a}_t|\mathbf{s}_t)}\hat{A}_t = r_t(\theta)\hat{A}_t \tag{2.102}$$

where $r_t(\theta, \theta_k) = \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_k}(\mathbf{a}_t|\mathbf{s}_t)}$ represents the probability ratio between the most current policy π_θ and the old policy from the previous iteration π_{θ_k}. Note that at the first gradient update after a new trajectory dataset $\mathcal{D}_k$ is collected, $\theta = \theta_k$ and hence $r_t(\theta, \theta_k) = 1$. If $\theta = \theta_k$, then $\nabla_\theta L_t(\theta) = \nabla_\theta L_t^{CPI}(\theta)$, which motivates why $L_t^{CPI}(\theta, \theta_k)$ can be used as a drop-in replacement for $L_t(\theta)$ for computing policy gradient estimates.

Maximizing $L_t^{CPI}(\theta, \theta_k)$ with respect to θ results in the following behaviour: if $\hat{A}_t$ is positive, $r_t(\theta, \theta_k)$ increases from 1; if $\hat{A}_t$ is negative, $r_t(\theta, \theta_k)$ decreases from 1 and approaches 0. Therefore, maximizing $L_t^{CPI}(\theta, \theta_k)$ directly without constraints would result in excessively large policy updates, which could collapse the policy's performance. Consequently, PPO modifies $L_t^{CPI}(\theta, \theta_k)$ by clipping $r_t(\theta, \theta_k)$ such that $r_t(\theta, \theta_k)$ cannot move too much away from 1. Specifically, PPO introduces $L_t^{CLIP}(\theta, \theta_k)$ defined as follows:

$$L_t^{CLIP}(\theta, \theta_k) = \begin{cases} \min(r_t(\theta, \theta_k),\ (1+\epsilon))\hat{A}_t & \text{if } \hat{A}_t \geq 0 \\ \max(r_t(\theta, \theta_k),\ (1-\epsilon))\hat{A}_t & \text{if } \hat{A}_t < 0 \end{cases} \tag{2.103}$$

where ϵ is a small hyperparameter, commonly set to 0.2. Figure 2.2 depicts the behaviour of L_t^{CLIP} as a function of $r_t(\theta, \theta_k)$ for the two cases of $\hat{A}_t$.

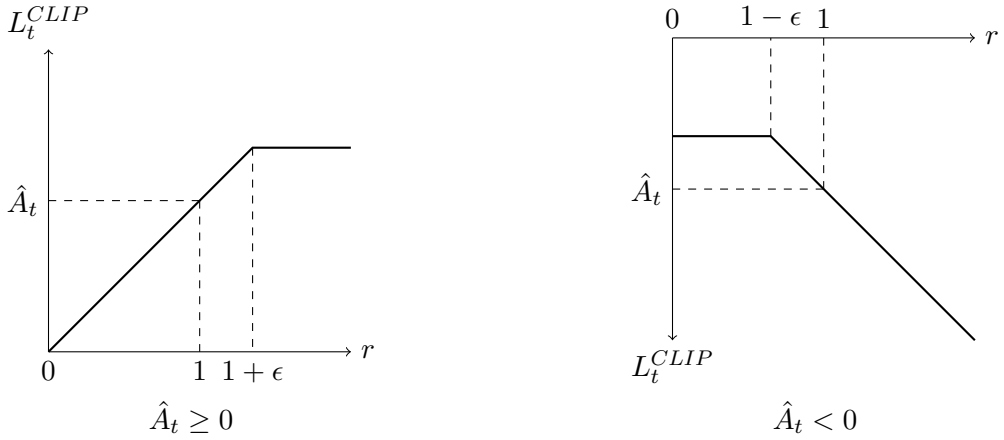

Figure 2.2: L_t^{CLIP} as a function of $r_t(\theta, \theta_k)$ for positive $\hat{A}_t$ (left) and negative $\hat{A}_t$ (right).

Note that if $r_t(\theta, \theta_k)$ is clipped to $(1+\epsilon)$ or $(1-\epsilon)$, it means that the output distribution of the new policy π_θ deviates more than the allowable amount from that of the old policy π_{θ_k}. In these cases, the gradient of L_t^{CLIP} with respect to θ is 0, effectively nullifying that particular state-action pair's contribution to the policy gradient. Thus, clipping prevents θ from moving in directions that result in large differences between the output distribution of π_θ and the output distribution of π_{θ_k}.

The overall objective function PPO maximizes is defined as follows:

$$L^{CLIP+VF+S}(\theta, \phi) = \frac{1}{N} \sum_{\tau \in \mathcal{D}} \left[\sum_{t=0}^{T-1} L_t^{CLIP}(\theta, \theta_k) - c_1 L_t^{VF}(\phi) + c_2 S[\pi_\theta](\mathbf{s}_t) \right] \tag{2.104}$$

where $S[\pi_\theta](\mathbf{s}_t)$ denotes the entropy of the policy's output distribution given state $\mathbf{s}_t$. This entropy bonus serves to prevent the policy's output distribution from collapsing, which encourages exploration for the agent throughout the training process.

2.3 Related Work

There are mainly three broad categories of approaches for manipulator collision-free motion generation: locally reactive controllers, global planners, and more recently neural motion generation methods. Each of these methods possesses its own set of advantages and drawbacks.

2.3.1 Locally Reactive Controller

Locally reactive controllers have long been applied to the problem of collision-free motion generation [6, 12, 32, 46]. These methods generate collision-free trajectories by generally moving in the direction of its goal while reacting to its surrounding obstacles. Geometric Fabrics is a recent example of this approach. Locally reactive methods are capable of running in real-time, enabling them to react to dynamic obstacles. They also can account for robot dynamics and hence can generally generate smooth trajectories. However, they are prone to local minima and frequently fail to find a solution in complex environments.

2.3.2 Global Planner

Global planners generate collision-free solutions by considering the entire state-space of the problem, hence they are generally asymptotically complete and optimal. Search-based planners, such as A* [18], are generally fast, complete, and optimal as it performs a graph search to find a solution;

consequently, it necessitates the construction of a graph, which requires the discretization of the continuous state space for manipulator motion generation. Sampling-based planners, such as RRT* and AIT* [15, 40], do not require discretization as they construct a tree via sampling in continuous space and find a solution once the tree discovers the goal in the state-space; however, building up a tree that has sufficient coverage over the state-space can be slow and the solution time is erratic due to the nature of random sampling. Optimization-based planners [36, 41, 48] generates collision-free paths by optimizing a combination of various objectives, such as distance to goal, distance to obstacles, and smoothness of trajectories; however, due to the nature of optimization, these methods tend to be computationally expensive, slow, and are prone to local minima depending on the optimization initialization. CuRobo [41] is a recent state-of-the-art optimization-based planner that has achieved significant speed ups via parallelizing the optimization scheme and accelerating the computation on the GPU. In spite of this decrease in solve time, CuRobo cannot be applied effectively in dynamic environments since its optimization process cannot guarantee a fixed solve rate, which is essential for reactive motion generation.

2.3.3 Neural Motion Generation

Neural motion generation is a recent paradigm that seeks to leverage various benefits of neural networks for motion generation. All of the non-learning based methods discussed above require a known environment model to function, which severely hinders their ability to be deployed in unknown environments. On the other hand, neural networks offer flexibility to various input modalities, such as point clouds of the scene, which can be readily obtained in any environment directly from the raw sensor output of depth cameras. In addition, neural networks possess rapid inference speed and can be executed at a constant rate, which makes it feasible to be deployed in dynamic environments where the robot needs to react to its surroundings at every time step. Seeking to reap these aforementioned benefits, recent work [9, 30] train neural policies that take in point cloud observations of the scene and output robot joint actions. Specifically, MπNets [9] is a state of the art neural motion generation method that is trained via imitation learning on three million optimal trajectories. While MπNets has a fast run time and displays reactive behaviour in dynamic settings, it performs poorly on goals that are out of the training distribution and exhibits erratic motion in unseen environments during training.

2.4 Research Gap

While neural motion generation methods possess inherent benefits from using neural networks, as discussed in Section 2.3.3, the quality of the generated motion from neural policies can be significantly improved. For instance, despite the large-scale dataset that was used to train the state-of-the-art MπNets, consisting of three million optimal trajectories from 575,000 unique scenes, the trained policy fails to generalize to most out-of-distribution settings. In addition, MπNets performs poorly in many difficult motion generation problems that were included in its training dataset as it was only able to solve 75.78% of the problems in the training dataset. In addition, the collision rate of trajectories generated by MπNets is 11% on the test environments despite the fact that the test environments are are sampled from the same distribution as the training set. Therefore, MπNets currently cannot be reliably deployed in most settings. Therefore, this work seeks to train a collision-free motion generation policy that outperforms MπNets in terms of completeness, smoothness, and reactiveness to obstacles while retaining its rapid inference speed.

Chapter 3

Methods

This section delineates the methods used in geometric fabrics guided reinforcement learning, a novel neural motion generation method that achieves state-of-the-art performance results for learning-based manipulator collision-free motion generation.

3.1 Problem Formulation

We define the problem of manipulator collision-free motion generation in this work as follows: for a fixed-base serial-link robot manipulator in an environment with a set of static or dynamic obstacles, given an initial joint configuration $\mathbf{q}_0$ for the manipulator and an end-effector goal position and orientation $\mathbf{x}_d$, generate robot joint torques that will steer the robot's end-effector to $\mathbf{x}_d$ without colliding with itself nor its environment. Figure 3.1 depicts an example solution trajectory for a collision-free manipulator global motion generation problem.

3.2 Methods Motivation

To achieve an improved performance over the previous state-of-the-art neural motion generation method MπNets, we formulate the problem as a reinforcement learning (RL) task instead of employing imitation learning (IL) which was previously used in MπNets. Policies learned from imitation learning are fundamentally limited by the quality and the coverage of the training dataset; consequently, IL policies often fail to generalize to out-of-distribution settings, which was one of the main exhibited with MπNets despite training on a large-scale dataset consisting of three million expert trajectories in 575,000 different environments [9]. Therefore, to improve the completeness of the neural policy, we seek to use reinforcement learning in simulation due to the fact that an RL agent in simulation can explore, generate data, and train on its own data continuously, whereas an IL agent is limited by the provided dataset. Moreover, RL inherently trains policies that encode long-term planning capabilities since RL optimizes over entire trajectory rollouts, which implicitly

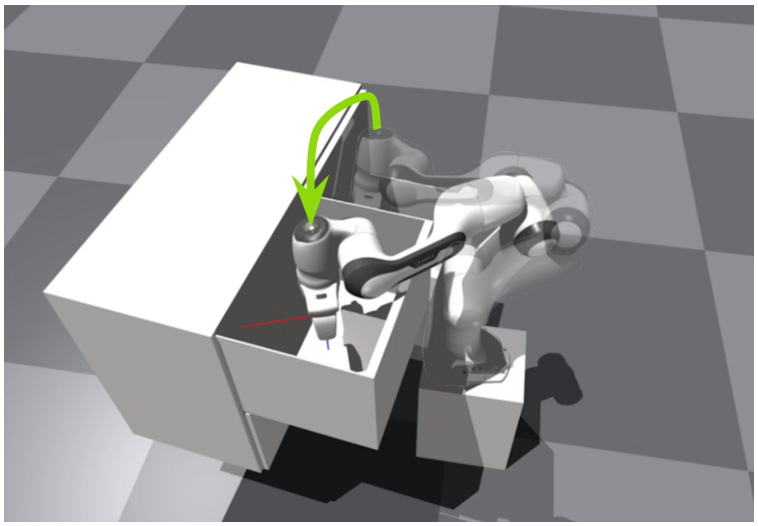

Figure 3.1: Illustration of an example solution trajectory for a collision-free manipulator global motion generation problem.

endows policies with the ability to anticipate future outcomes [5].

In most of the literature regarding RL for motor control, the RL agent is setup to directly output robot joint actions [3, 8, 35]. However, this action space formulation has two main problems in the context of producing feasible robot motion and the learning efficiency. Firstly, policies learned from RL algorithms such as PPO have a tendency to produce bang-bang control behaviour, resulting in jerky motion that is infeasible to achieve on real robots. This behaviour is usually partially alleviated by adding joint velocity penalty terms in the reward function or artificially smoothing the RL actions with an exponential moving average (EMA) low-pass filter [10]. Using these methods may appear to result in smoother robot motion, but there is no guarantee that the robot motion will stay within the allowable jerk and acceleration limits imposed by the robot hardware. In fact, joint jerk and acceleration information are not exposed and cannot be easily retrieved in robotic simulators like Isaac Gym. Therefore, in order to train a policy that produces physically feasible robot motion, a conservative amount of action damping is usually required via both velocity penalization and a high EMA factor; this approach produces excessively sluggish robot motion, hence resulting in sub-optimal behaviour for motion generation since the robot cannot safely utilize its full capabilities. Secondly, RL agent needs to learn a variety of non-task specific behaviour, such as dynamic compensation, local object avoidance, satisfying joint level constraints, in addition to the main task objective. Needing to learn these non-task specific behaviour only adds to the difficulty of the RL task, which could result in slower training time or complete failure in achieving

the main task objective.

Therefore, in consideration of these two problems, we employ geometric fabrics to serve as a safe guiding medium for the RL policy to operate on [31]. Specifically, the RL policy sets targets for the end-effector attractor and the joint-space attractor of geometric fabrics, which then generates joint position targets for the robot manipulator. This approach effectively solves the two aforementioned problems. Firstly, trajectories generated by geometric fabrics are guaranteed to be smooth and physically feasible since the geometric fabrics framework can explicitly ensure that its motion satisfies joint jerk and acceleration constraints [31]. This effectively nullifies the need for any action-penalization term in the reward function to implicitly slow down the robot's motion, allowing the RL agent to take full advantage of the robot's motion capabilities while adhering to the robot's physical limitations. Secondly, geometric fabrics is a powerful controller that exhibits all of the aforementioned non-task specific behaviour, which can reduce the complexity of the learning task. We choose to leverage geometric fabrics as opposed to other controllers because of its empirical performance superiority over other controllers and its provably stable characteristics [31, 46].

3.3 Geometric Fabrics Design

In this section, we delineate the design of geometric fabrics to create a powerful locally reactive controller for a robot manipulator. The combined geometric fabrics used by the RL policy comprises the following fabric components: end-effector attractor, joint-space attractor, and distance-space repulsion. Each fabric component consists a geometry generator $(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}_i}$ and a Finsler energy $\mathcal{L}_{e,i}$. The geometry generator $(\mathbf{I}, \mathbf{h}_{2,i})_{\mathcal{X}_i}$ defines the nominal behaviour of a geometric fabric and can be interpreted as an acceleration sub-policy $\pi_i = \ddot{\mathbf{x}}_i = -\mathbf{h}_{2,i}$. The Finsler energy $\mathcal{L}_{e,i}$ dictates the priority metric of the fabric in accordance with $\mathbf{M}_{e,i} = \partial^2_{\dot{\mathbf{x}}\dot{\mathbf{x}}} \mathcal{L}_{e,i}$. These fabric components are combined into the final geometric fabric using Equation (2.54). In accordance with Equation (2.60), the combined geometric fabric can be expressed as follows:

$$\mathbf{M}_e \ddot{\mathbf{q}}_e + \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e \mathbf{h}_2 - \mathbf{f}_e] = -\partial_{\mathbf{q}_e} \psi - \mathbf{B} \dot{\mathbf{q}}_e \tag{3.1}$$

$$\mathbf{M}_e \ddot{\mathbf{q}}_e + \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e \mathbf{h}_2 - \mathbf{f}_e] + \partial_{\mathbf{q}_e} \psi + \mathbf{B} \dot{\mathbf{q}}_e = \mathbf{0} \tag{3.2}$$

$$\mathbf{M}_e(\mathbf{q}_e, \dot{\mathbf{q}}_e) \ddot{\mathbf{q}}_e + \mathbf{f}_f(\mathbf{q}_e, \dot{\mathbf{q}}_e) = \mathbf{0} \tag{3.3}$$

where $\mathbf{f}_f(\mathbf{q}_e, \dot{\mathbf{q}}_e) = \mathbf{f}_e + \mathbf{P}_e[\mathbf{M}_e\mathbf{h}_2 - \mathbf{f}_e] + \partial_{\mathbf{q}_e}\psi + \mathbf{B}\dot{\mathbf{q}}_e$ can be interpreted as the combined force in the geometric fabric system.

3.3.1 End-Effector Attractor

The purpose of the end-effector attractor is to move the current end-effector position $\mathbf{x}$ to a desired end-effector position $\mathbf{x}_d$, where the task space of this fabric the robot's end-effector space.

We specify a potential function $\psi(\mathbf{x}) = k\log(\cosh(\alpha_\psi(||\mathbf{x}_d - \mathbf{x}||)))$ and design the geometric sub-policy as $\pi_i = -||\dot{\mathbf{x}}||^2\partial_{\dot{\mathbf{x}}}\psi(\mathbf{x})$, where $k, \alpha_\psi \in \mathbb{R}^+$ are constants that control the scaling and the sharpness of $\psi(\mathbf{x})$ respectively. In essence, $\psi(\mathbf{x})$ decreases towards 0 as $\mathbf{x}$ approaches $\mathbf{x}_d$ and the sub-policy assigns an end-effector acceleration in the direction that decreases $\psi(\mathbf{x})$, hence attracting $\mathbf{x}$ towards $\mathbf{x}_d$. Note that the sub-policy π_i is HD2 in $\dot{\mathbf{x}}$, hence satisfying the requirement of being a geometric generator.

We design the Finsler energy as $\mathcal{L}_e = \|\dot{\mathbf{x}}\|^2 \left(\frac{1}{2}(\overline{m} - \underline{m})\left(\tanh\left(-\alpha\left\|\mathbf{x}_d - \mathbf{x}\right\|\right) + 1\right) + \underline{m}\right)$, where $\overline{m}, \underline{m}, \alpha \in \mathbb{R}^+$ are the maximum isotropic mass, the minimum isotropic mass, and the sharpness of the hyperbolic tangent respectively. The induced metric from the Finsler energy is thus as follows: $\mathbf{M}_e = \left((\overline{m} - \underline{m})\left(\tanh\left(-\alpha\left\|\mathbf{x}_d - \mathbf{x}\right\|\right) + 1\right) + \underline{m}\right)\mathbf{I}$, where the priority of the geometry increases as $\mathbf{x}$ approaches $\mathbf{x}_d$.

In addition to the geometric sub-policy, we also separately deploy the potential $\psi(\mathbf{x})$ with a different tuning from the one used in the geometric policy. This potential is generally less potent than the geometric policy, but it helps with convergence.

We note that in order to achieve end-effector orientation alignment in conjunction with position alignment, we convert the desired end-effector pose to the positions of three non-collinear points on the end-effector and subsequently apply an end-effector attractor to each of the three points. We expose the desired end-effector pose $\mathbf{T}_d$ as a controllable parameter that the RL policy can modify.

3.3.2 Joint-Space Attractor

The purpose of the joint-space attractor is to control the null-space of a manipulator in order to posture the arm in addition to controlling its end-effector pose. Specifically, the goal of the joint-space attractor is to bring the arm's joint configuration $\mathbf{q}$ as close to a desired joint configuration $\mathbf{q}_d$ as possible without impeding on the end-effector attractor.

We define a potential $\psi(\mathbf{q}) = k\log\cosh(\alpha_\psi(||\mathbf{q}_d - \mathbf{q}||)$, where $k, \alpha_\psi \in \mathbb{R}^+$ are constants that control the scaling and the sharpness of $\psi(\mathbf{q})$ respectively. Similarly to the end-effector attractor,

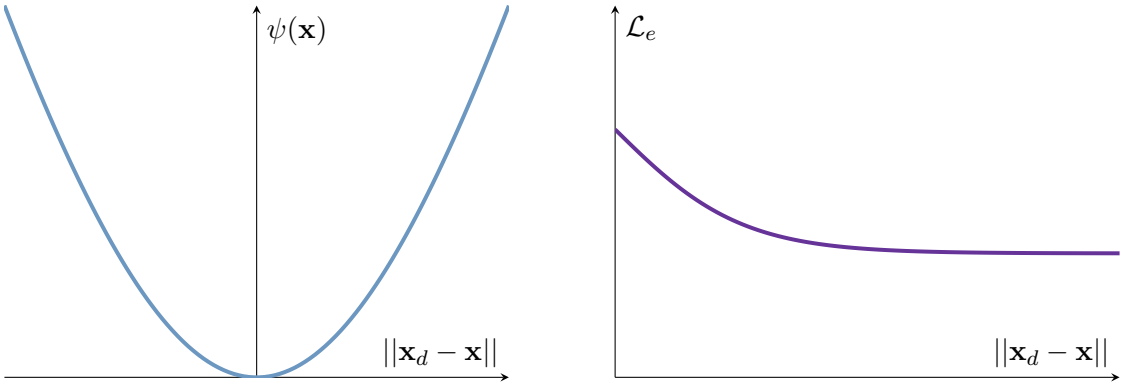

Figure 3.2: The potential (left) and the Finsler energy (right) used for the end-effector attractor.

we design the geometric sub-policy as $\pi_i = -||\dot{\mathbf{q}}||^2 \partial_{\dot{\mathbf{q}}} \psi(\mathbf{q})$. We design the Finsler energy as $\mathcal{L}_e = \frac{m}{2}||\dot{\mathbf{q}}||^2$, where $m \in \mathbb{R}^+$ is the isotropic mass. The induced metric from the Finsler energy is thus $\mathbf{M}_e = m\mathbf{I}$. We also expose the desired joint-space target $\mathbf{q}_d$ as a controllable parameter that the RL policy can adjust.

3.3.3 Distance-Space Repulsion

We employ the distance-space repulsion fabric to achieve self-collision avoidance, obstacle avoidance, and joint-limit avoidance. Specifically, the task-space of the distance-space repulsion fabric is the signed distance to a barrier, denoted generally as x. For self-collision avoidance, x represents the distance from a robot link to anther robot link. For obstacle avoidance, x represents the distance from a robot link to an obstacle. For joint-limit avoidance, x represents the distance from the current joint position to either the upper or the lower joint limit. Therefore, the distance-space repulsion fabric aims to move x away from 0.

We define a potential $\psi(x) = \frac{k_b}{x} + \frac{k_r}{\alpha} \log\left(1 + e^{-\alpha(x - x_o)}\right)$, where $k_b, k_r \in \mathbb{R}^+$ control the scaling, α controls the sharpness, and x_o is an activation offset. Similarly to the others, we design the geometric sub-policy as $\pi_i = -\dot{x}^2 \partial_x \psi(x)$.

We design the Finsler energy as $\mathcal{L}_e = \frac{k}{2x} s(\dot{x}) \dot{x}^2$, where $k \in \mathbb{R}^+$ controls scaling and $s(\dot{x}) = 1$ if $\dot{x} < 0$ and $s(\dot{x}) = 0$ otherwise. The metric induced from the Finsler energy is $\mathbf{M}_e = \frac{k}{x}\mathbf{I}$ if $\dot{x} < 0$ and $\mathbf{M}_e = 0$ otherwise. Effectively, the distance-space repulsion fabric is disabled if the system is moving away from the barrier and the priority of this fabric increases as the system moves towards the barrier, which demonstrates the benefits of velocity-dependent metrics.

We also deploy the potential $\psi(x)$ alongside the geometric sub-policy for numerical stability purposes [46]. Note that this potential is weighted significantly less than the geometric sub-policy.

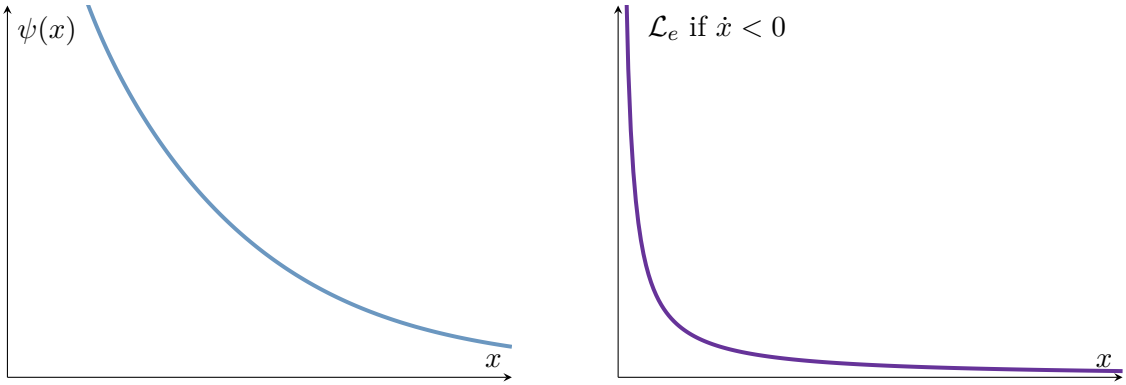

Figure 3.3: The potential (left) and the Finsler energy (right) used for the distance-space repulsion.

3.3.4 Satisfying Joint Jerk and Acceleration Constraints

We explicitly handle joint jerk and acceleration constraints imposed by the robot hardware to ensure that the trajectories generated by geometric fabrics are physically feasible. We denote $\overline{\ddot{\mathbf{q}}}$ as the maximum acceleration magnitude and $\overline{\dddot{\mathbf{q}}}$ as the maximum jerk magnitude.

To handle the jerk constraint, we leverage the following time-discretized jerk model:

$$\dddot{\mathbf{q}}_e^t = \frac{\ddot{\mathbf{q}}_e^{t+1} - \ddot{\mathbf{q}}_e^t}{\Delta t} \tag{3.4}$$

The maximum possible jerk that the system would experience is if $\ddot{\mathbf{q}}_e^{t+1} = \overline{\ddot{\mathbf{q}}}$ and $\ddot{\mathbf{q}}_e^t = -\overline{\ddot{\mathbf{q}}}$, resulting in the following expression for the maximum possible jerk:

$$\dddot{\mathbf{q}}_{e,max}^t = \frac{2\overline{\ddot{\mathbf{q}}}}{\Delta t} \tag{3.5}$$

Since we need to ensure $\dddot{\mathbf{q}}_{e,max}^t < \overline{\dddot{\mathbf{q}}}$, we can derive the following requirement of $\overline{\ddot{\mathbf{q}}}$ in terms of $\overline{\dddot{\mathbf{q}}}$:

$$\frac{2\overline{\ddot{\mathbf{q}}}}{\Delta t} < \overline{\dddot{\mathbf{q}}} \tag{3.6}$$

$$\overline{\ddot{\mathbf{q}}} < \frac{\Delta t \overline{\dddot{\mathbf{q}}}}{2} \tag{3.7}$$

Therefore, we can define a new maximum acceleration $\overline{\ddot{\mathbf{q}}}'$ such that ensuring $|\ddot{\mathbf{q}}_e| \leq \overline{\ddot{\mathbf{q}}}'$ simultaneously satisfies the maximum jerk constraint:

$$\overline{\ddot{\mathbf{q}}}' = \min\left(\overline{\ddot{\mathbf{q}}}, \frac{\Delta t \overline{\dddot{\mathbf{q}}}}{2}\right) \tag{3.8}$$

In order to ensure $|\ddot{\mathbf{q}}_e| \leq \overline{\ddot{\mathbf{q}}}'$, we make the following modification to Equation (3.3):

$$(\mathbf{M}_e + \alpha\mathbf{I})\ddot{\mathbf{q}}_e + \mathbf{f}_f = \mathbf{0} \tag{3.9}$$

$$\ddot{\mathbf{q}}_e = -(\mathbf{M}_e + \alpha\mathbf{I})^{-1}\mathbf{f}_f \tag{3.10}$$

where α is a parameter we can set to make $\ddot{\mathbf{q}}_e$ arbitrarily small since $\alpha \to \infty$ implies $\ddot{\mathbf{q}}_e \to \mathbf{0}$. Therefore, we can solve for α that guarantees $|\ddot{\mathbf{q}}_e| \leq \overline{\ddot{\mathbf{q}}}'$, hence satisfying both the joint jerk and acceleration constraints.

3.3.5 Interactions with Real Dynamics

Equation (3.10) can be interpreted as an acceleration policy that specifies the dynamics that we desire the robot to follow, referred to as behavioural dynamics. However, the real dynamics of a serial-link manipulator system is governed by the following equation of motion:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \tag{3.11}$$

where $\mathbf{M}(\mathbf{q})$ is the robot's generalized mass matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ is the Coriolis forces, $\mathbf{g}(\mathbf{q})$ is the gravitational forces, and $\boldsymbol{\tau}$ is the generalized joint forces. We can make the robot follow the desired behavioural dynamics using PD control with inverse dynamics compensation:

$$\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{q}_e, \dot{\mathbf{q}}_e, \ddot{\mathbf{q}}_e) = \mathbf{M}(\mathbf{q})(\ddot{\mathbf{q}}_e + \mathbf{K}_p(\mathbf{q}_e - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_e - \dot{\mathbf{q}})) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) \tag{3.12}$$

where we treat $\mathbf{q}_e$ and $\dot{\mathbf{q}}_e$ from the behavioural dynamics policy as the desired joint position and velocity targets for the inverse dynamics controller.

To obtain $\mathbf{q}_e$ and $\dot{\mathbf{q}}_e$, we forward integrate $\ddot{\mathbf{q}}_e$, computed from the behavioural dynamics acceleration policy. We employ an approximate RK2 numerical integration method to calculate $\mathbf{q}_e$ and $\dot{\mathbf{q}}_e$ as follows:

$$\mathbf{q}_e^{t+1} = \mathbf{q}_e^t + \Delta t\dot{\mathbf{q}}_e^t + \frac{1}{2}\Delta t^2\ddot{\mathbf{q}}_e^t \tag{3.13}$$

$$\dot{\mathbf{q}}_e^{t+1} = \dot{\mathbf{q}}_e^t + \Delta t\ddot{\mathbf{q}}_e^t \tag{3.14}$$

3.4 Geometric Fabrics Guided Reinforcement Learning

We formulate the problem of generating collision-free motion generation for a manipulator as a discrete-time Markov Decision Process, where the agent interacts with its environment by sequentially observing and applying actions. In order for the agent to maximize the expected discounted return $J(\pi_\theta) = \mathbb{E}_{\mathbf{g}\sim P(\mathbf{g})}\mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$, we use Proximal Policy Optimization (PPO) [39] to learn a goal-conditioned stochastic actor policy π_θ that maps state $\mathbf{s} \in S$ and goal $\mathbf{g} \in G$ to action $\mathbf{a} \in A$. PPO also learns a critic $V_\phi(\mathbf{s})$ to approximate the on-policy value function.

We apply PPO in Isaac Gym [20], a GPU-based physics simulator, due to the fact that PPO is highly parallizable, allowing it to scale effectively with GPU-accelerated vectorized simulation [8, 16, 22, 35]. Figure 3.4 presents a high level overview of the training pipeline of geometric fabrics guided reinforcement learning. We note that the actor and the critic networks run at 2 Hz while geometric fabrics run at 60 Hz.

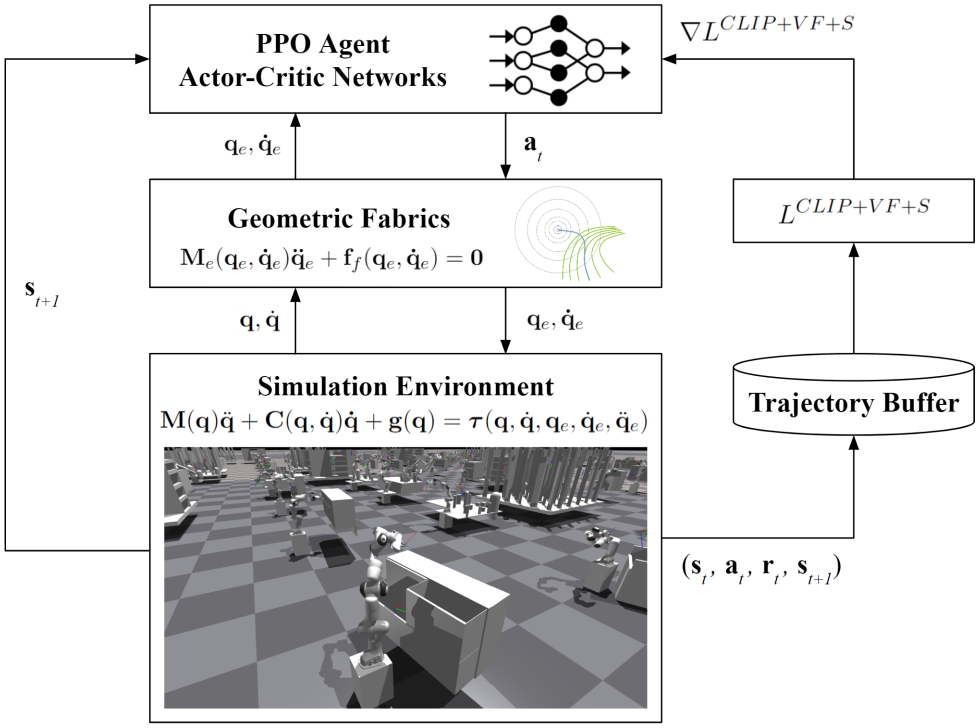

Figure 3.4: High level PPO training pipeline of geometric fabrics guided reinforcement learning.

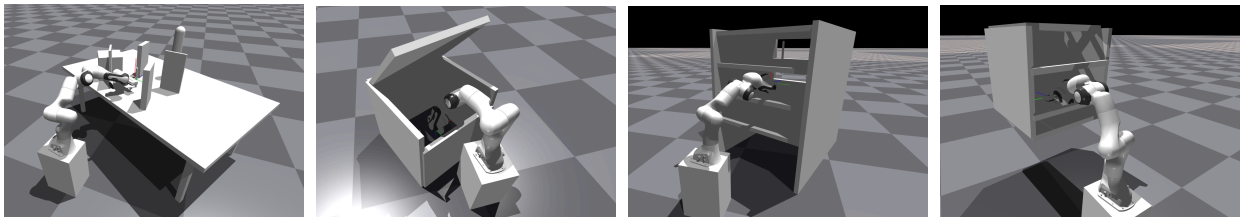

Figure 3.5: MotionBenchMaker Scenes: Table, Box, Bookshelf, and Cage.

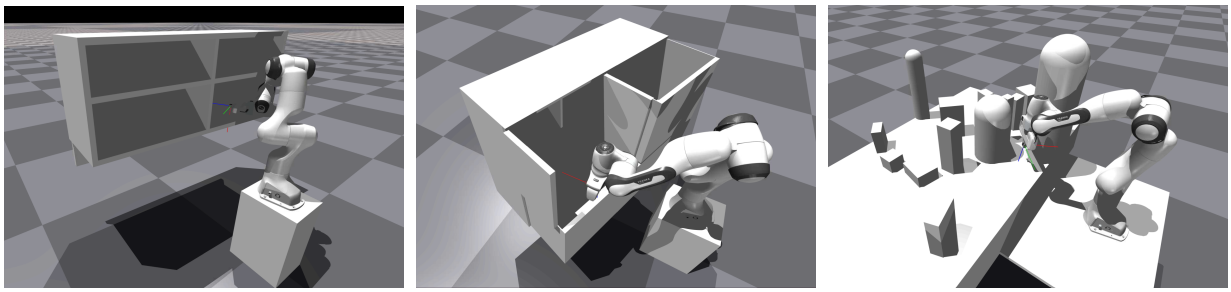

Figure 3.6: MπNets Scenes: Cubby, Dresser, and Tabletop.

3.4.1 Simulation Environment

We implement two main types of obstacle scenes in our Isaac Gym simulation environment to train a collision-free motion generation policy: dataset scenes and procedurally generated scenes.

3.4.1.1 Dataset Scenes

Dataset scenes are obstacle scenes from existing datasets. Specifically, we import 1800 unique scenes from the MotionBenchMaker (MBM) dataset [7] which we use during training. These MBM scenes consist of four categories, illustrated in Figure 3.5: table, box, bookshelf, and cage. In addition, we import 800 unique scenes from the MπNets dataset [9], which we use during testing to evaluate and compare our policies since these scenes are outside of the training distribution. These MπNets scenes consist of the following three categories, illustrated in Figure 3.6: cubby, dresser, and tabletop.

Each of these dataset scenes has a pre-defined initial robot configuration and end-effector goal, which specifies a challenging motion generation problem. In addition to these pre-defined problems, we generate more collision-free initial robot configurations and end-effector goals online during training via rejection sampling. Specifically, we repeatedly uniformly sample joint configurations within joint limits, perform forward kinematics based on the sampled joint configurations, and check for collisions until we yield two collision-free joint configurations. We can then use one of

the configurations as the initial configuration and the resulting end-effector pose from the other configuration as the end-effector goal.

Therefore, for each environment reset, we can set the initial configuration to either the pre-defined configuration or a generated configuration via rejection sampling, and we can set the goal to either the pre-defined end-effector goal pose or a generated one, resulting in four possibilities. We then sample one of the four modes per environment reset in accordance with a categorical distribution. Table 3.1 delineates the probabilities associated with each of the four modes that we use during training.

Initial Configuration	**End-Effector Goal Pose**	**Probabilities (%)**
Pre-defined	Pre-defined	55
Pre-defined	Rejection Sampling	15
Rejection Sampling	Pre-defined	15
Rejection Sampling	Rejection Sampling	15

Table 3.1: Categorical distribution for sampling one of the four possible modes at environment resets for dataset scenes.

3.4.1.2 Procedurally Generated Scenes

In order to increase the diversity and coverage of obstacle scenes used during training, we procedurally generate obstacle scenes on the fly at the start of each training run in addition to the dataset scenes. In this work, we include three types of procedurally generated scenes, depicted in Figure 3.7: cuboid forest, shelves, and flying cuboids.

Cuboid Forest: Randomly vertically-stretched cuboids are placed in a grid-like pattern with small random perturbations applied to the base positions. Each cuboid is also randomly rotated about the z-axis. Cuboids near the robot base are removed to prevent interpenetration between the scene and the robot.

Shelves: The robot manipulator is surrounded by four tall shelves, where the spacing between each level of a shelf is randomized between 0.2 meters and 0.4 meters.

Flying Cuboids: Ten to twelve cuboids of various dimensions are placed around the robot manipulator, where their positions are uniformly sampled from a sphere of radius between 0.3 meters and 1.2 meters, and their orientations are uniformly sampled.

We note that the initial configurations and end-effector goal poses of these procedurally generated scenes are set specified the same rejection sampling process delineated in Section 3.4.1.1. Due to the random nature of these scenes, there is no guarantee that every generated motion generation

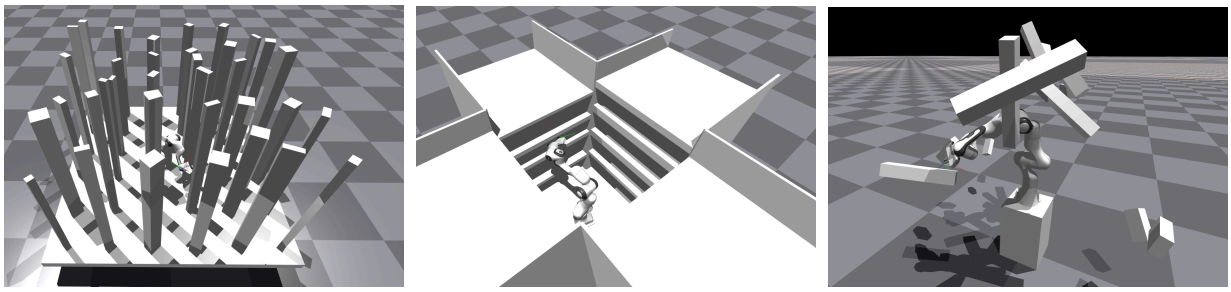

Figure 3.7: Procedurally Generated Scenes: Cuboid Forest, Shelves, and Flying Cuboids.

problem has a solution; in practice, we find that our trained policy can solve more than 95% of these problems.

3.4.2 Observation Space

The observation space of the goal-conditioned actor policy π_θ and the critic network V_ϕ is identical, comprising of three main components: the goal, robot proprioception, observation of the scene, and the exploration map. The details regarding the observation space is delineated in Table 3.2.

Observation Category	**Observation Name**	**Shape**
Goal	End-effector goal pose, represented as keypoints	$(8 \times 3,)$
Proprioception	Joint positions	$(7,)$
	End-effector pose, represented as keypoints	$(8 \times 3,)$
	Fabric joint positions	$(7,)$
	Fabric joint velocities	$(7,)$
	Fabric joint accelerations	$(7,)$
Scene Observation	Static basis point set (vectors)	$(16, 16, 16, 3)$
	Dynamic basis point set (vector magnitudes)	$(62,)$
Exploration Map	Exploration map array	$(20, 20, 20, 1)$
Miscellaneous	Fabrics switch	$(1,)$

Table 3.2: Observations of the actor policy and the critic network for the Franka Panda manipulator.

Goal: We represent the desired end-effector goal pose as a collection of eight keypoint positions placed at the vertices of the oriented bounding box surrounding the end-effector, visualized in Figure 3.8: $\{\mathbf{x}_{g,1}, \mathbf{x}_{g,2}, \ldots, \mathbf{x}_{g,8}\} \in \mathbb{R}^{8\times 3}$. We elect to represent the goal pose using keypoints instead of a position and a quaternion because prior work has shown empirically that the keypoint-based representation yields an improved policy performance [3, 27].

Proprioception: Robot proprioception includes robot joint positions $\mathbf{q}$, the fabric's joint positions $\mathbf{q}_e$, the fabric's joint velocities $\dot{\mathbf{q}}_e$, the fabric's joint accelerations $\ddot{\mathbf{q}}_e$, and the robot's

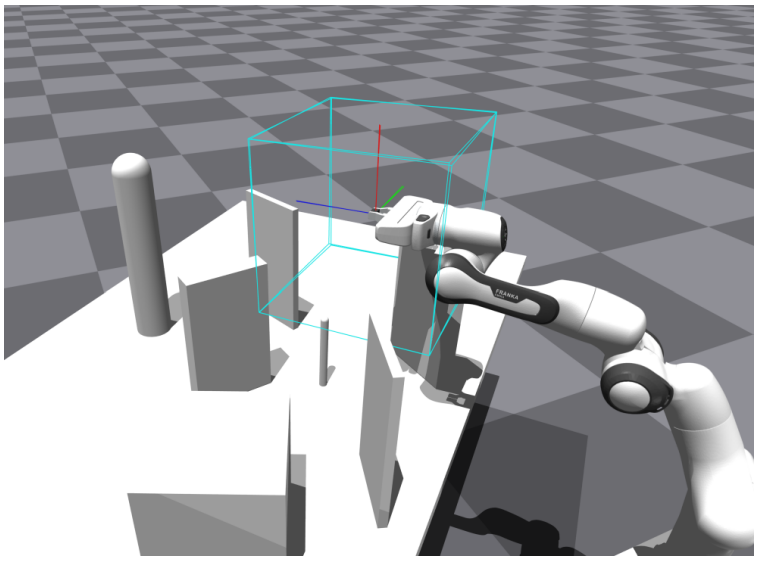

Figure 3.8: End-effector goal pose is represented using the eight keypoints at the vertices of the oriented bounding box surrounding the end-effector (in light blue).

end-effector pose, represented using the eight keypoints.

Scene Observation: In order for the policy to produce collision-free trajectories, the policy must be conditioned on some representation of the obstacle scene. We elect to use a basis point set (BPS) to encode spatial information of scene [28]. In comparison to other scene representation methods, such as point clouds or voxel grids, BPS is particularly suited for deep learning due to its superior efficiency in terms of memory and computation without degrading the network performance [28, 45]. Specifically, we implement BPS as a set of points arranged on a $16 \times 16 \times 16$ rectangular grid fixed with respect to the robot base frame, spanning from $(x_{\min}, y_{\min}, z_{\min}) = (-0.75, -1.0, -0.1)$ to $(x_{\max}, y_{\max}, z_{\max}) = (1.5, 1.0, 1.25)$:

$$B = \{\mathbf{b}_1, \ldots, \mathbf{b}_{16^3}\}, \mathbf{b}_i \in \mathbb{R}^3. \tag{3.15}$$

Each basis point gives a vector pointing from the location of the basis point to its nearest obstacle in the scene, excluding the robot itself, which produces the following scene representation:

$$\mathbf{X}_{BPS} = [(\tilde{\mathbf{x}}_\mathbf{1} - \mathbf{b}_1), \ldots, (\tilde{\mathbf{x}}_{\mathbf{16^3}} - \mathbf{b}_{16^3})] \in \mathbb{R}^{16 \times 16 \times 16 \times 3} \tag{3.16}$$

where $\tilde{\mathbf{x}}_\mathbf{i}$ denotes the position of the closest obstacle to basis point $\mathbf{b}_i$. Figure 3.9 provides a visualization of the array of basis points as well as the associated vectors pointing to the nearest obstacles.

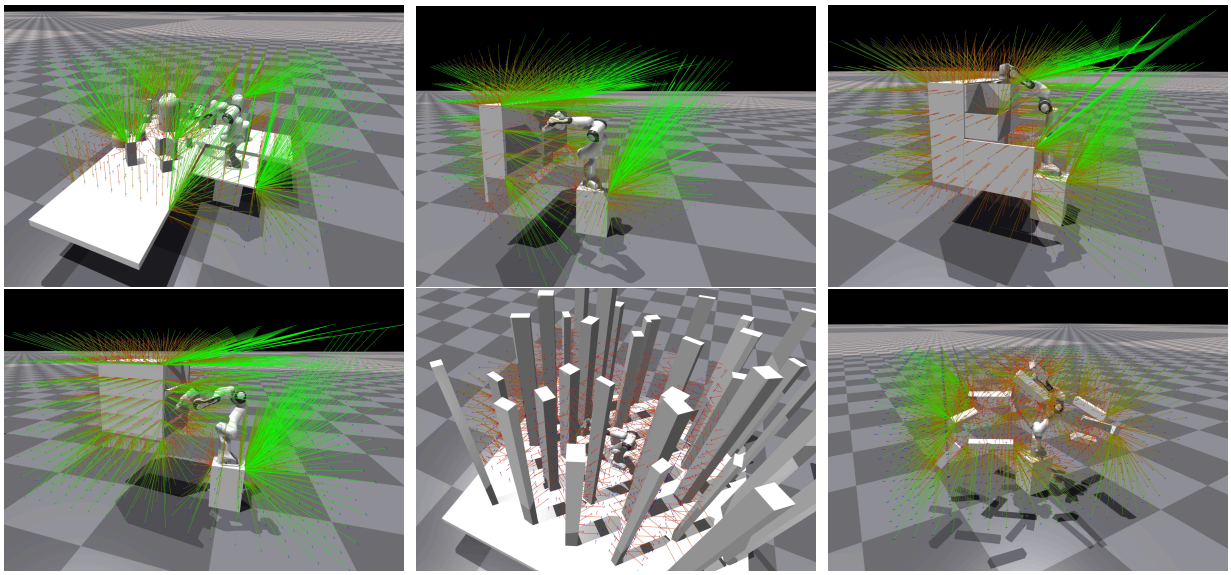

Figure 3.9: Visualization of the static basis point sets and their associated vector features, displayed at half density.

In addition to the aforementioned set of fixed basis points, we also implement a set of 62 dynamic basis points that are attached to the links of the manipulator. Note that for each dynamic basis point, we only pass in the magnitude of the associated vector since the magnitude is a smoothly changing value, whereas the vector itself is noisy due to due to the dynamic nature of these basis points. These basis points are illustrated in Figure 3.10.

Exploration Map: The exploration map $\mathbf{X}_{MAP} \in \mathbb{R}^{20\times20\times20}$ is a 3D array that discretizes the environment space into a $20 \times 20 \times 20$ grid, where each grid contains a binary value that indicates whether or not the end-effector has explored that grid in that episode. We include $\mathbf{X}_{MAP}$ as part of the observation space to inform the agent regarding the previously visited parts of the state space, which we empirically confirmed to lead to better exploration and recovery from local minima.

3.4.3 Action Space

The action space of the actor policy network π_θ consists of the following three components: end-effector control, joint-space control, and the fabrics switch. The details regarding the action space is presented in Table 3.3.

End-Effector Control: The policy network outputs a transformation relative to the current end-effector pose, consisting of a 3 DOF relative translation and a 3 DOF relative axis-angle rotation. This relative transformation is then applied to the current end-effector pose to yield

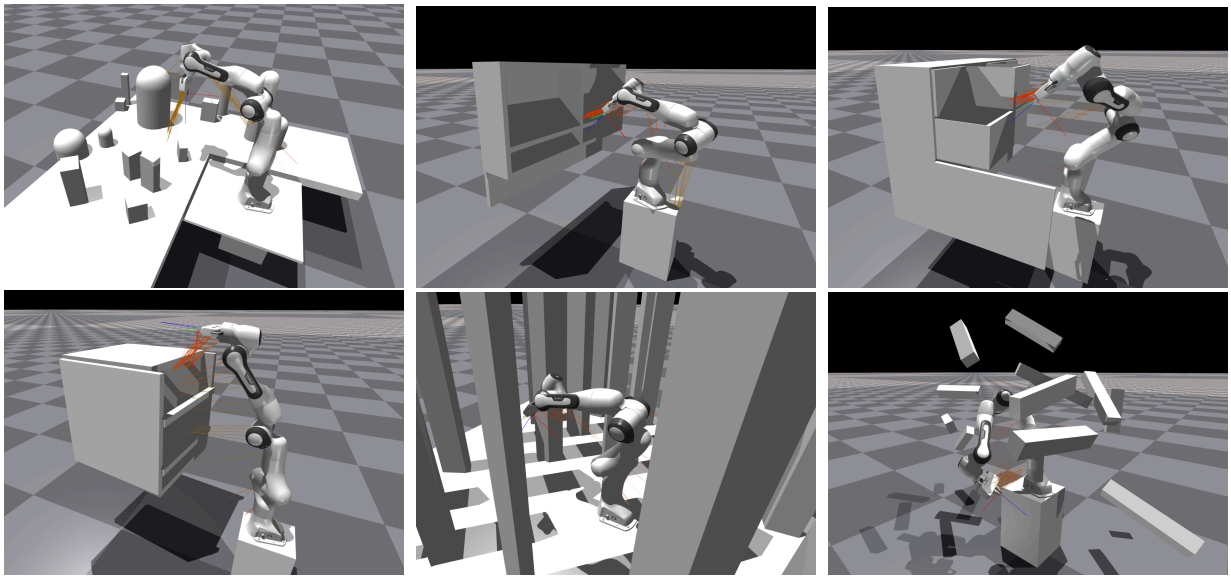

Figure 3.10: Visualization of the dynamic basis points and their associated vector features.

Action Category	**Action Name**	**Shape**
End-Effector Control	Relative end-effector translation	$(3,)$
	Relative end-effector axis-angle rotation	$(3,)$
Joint-Space Control	Joint-space targets	$(7,)$
Fabrics Switch	Fabrics switch	$(1,)$

Table 3.3: Action space of the actor policy network for the Franka Panda manipulator.

the target end-effector pose $\mathbf{T}_d$ for the geometric fabrics controller. We choose to interpret the policy's output as a relative end-effector transformation because prior work have demonstrated that this action formulation consistently resulted in higher performance and smoother trajectories than others for learning manipulation tasks [23, 44].

Joint-Space Control: The policy network outputs a joint-space target $\mathbf{q}_d$ for the geometric fabrics controller. This joint-space target serves to posture the manipulator primarily within the null-space of higher-priority objectives such as end-effector control or obstacle avoidance.

Fabrics Switch: The policy network outputs outputs a continuous value $[0, 1]$, which gets rounded to a binary output that we interpret as the fabrics switch. If the fabrics switch is 0, then the geometric fabrics controller takes in end-effector pose target $\mathbf{T}_d$ specified from the actor policy. However, if the policy sets the fabrics switch to 1, then the geometric fabrics controller disables its joint-space attractor and automatically sets its end-effector pose target to the end-effector goal pose for the remainder of the episode.

We observe that geometric fabrics on its own can reach most targets, given that the manipulator

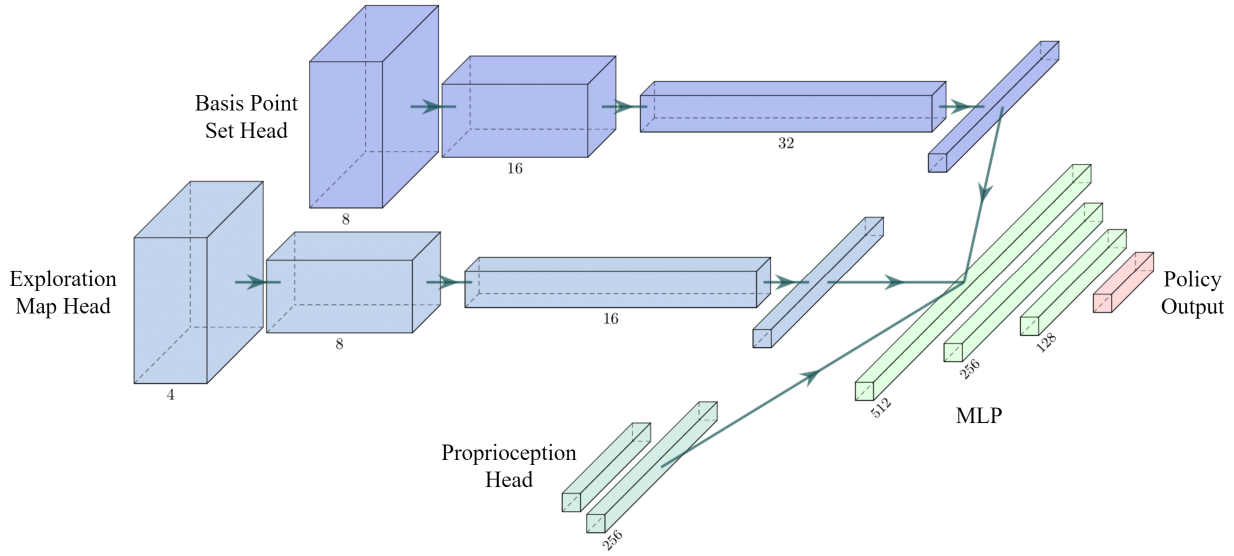

Figure 3.11: Actor policy network architecture.

is not initially trapped at a local minima state. Therefore, we do not need the RL actor policy to drive the arm all the way to the goal; instead, the RL policy merely needs to steer the arm away from local minima states and then simply rely on geometric fabrics to finish the task. In consideration of this observation, we implement this switch mechanism to take full advantage of the useful priors provided by geometric fabrics, which significantly decreases the difficulty of the learning task for the actor policy.

3.4.4 Network Architecture

Figure 3.11 depicts the architecture for both the actor policy π_θ and the critic network. The network has three input heads: the basis point set head, the exploration map head, and the proprioception head. The basis point set head expects basis point set features of shape $(16, 16, 16, 3)$. Three consecutive 3D convolutions + ReLU activations are performed. Similarly, the exploration map head expects a 3D array of shape $(20, 20, 20, 1)$, for which three consecutive 3D convolutions + ReLU activations are performed. The proprioception head is a single fully-connected layer that takes in the rest of features in the observation. The outputs of the three heads are flattened and concatenated, which is then passed into three consecutive fully-connected layers + ELU activations to produce the final output. Table 3.4 delineates the parameters of the network.

Sub-Network	Layers	Parameters
Basis Point Set	3D Convolution + ReLU 1	Filters: 8, Kernel Size: 4, Strides: 2
	3D Convolution + ReLU 2	Filters: 16, Kernel Size: 3, Strides: 2
	3D Convolution + ReLU 3	Filters: 32, Kernel Size: 2, Strides: 1
Exploration Map	3D Convolution + ReLU 1	Filters: 4, Kernel Size: 4, Strides: 2
	3D Convolution + ReLU 2	Filters: 8, Kernel Size: 3, Strides: 2
	3D Convolution + ReLU 3	Filters: 16, Kernel Size: 3, Strides: 1
Proprioception	Fully Connected Layer + ELU	Units: 256
MLP	Fully Connected Layer + ELU 1	Units: 512
	Fully Connected Layer + ELU 2	Units: 256
	Fully Connected Layer + ELU 3	Units: 128

Table 3.4: Parameters of the actor and the critic network.

3.4.5 Reward Function and Curriculum

The reward function merely consists of two terms: the sparse task reward and the intrinsic reward, summarized in Table 3.5.

The sparse task reward is given when the end-effector pose distance from the goal pose is less than 0.08. We compute the end-effector pose distance as the mean keypoints distance error $d = \sum_{i=1}^{8}(\mathbf{x}_{g,i} - \mathbf{x}_{k,i})^2$, where $\mathbf{x}_{g,i}$ and $\mathbf{x}_{k,i}$ are the goal end-effector pose and current end-effector pose keypoints respectively.

Since we are using a sparse reward formulation, we also employ a count-based intrinsic reward to encourage exploration [4]. We leverage the exploration map described in Section 3.4.2, which discretizes the environment space into a $20 \times 20 \times 20$ grid, where each grid counts the number of times the end-effector has visited that grid in an episode. We denote the visit count for grid ijk as n_{ijk} and calculate the intrinsic reward as follows:

$$r_{intrinsic} = \sum_{ijk \in V} w_{ijk}\ e^{-10(n_{ijk}-1)} \tag{3.17}$$

$$w_{ijk} = \frac{1}{d_{ijk} + 0.25} \tag{3.18}$$

where V is the set of grids that the end-effector has visited since the last policy step and d_{ijk} is the distance between grid ijk from the end-effector goal position. In essence, the reward-per-grid decreases exponentially as its visit count increases. The overall intrinsic reward $r_{intrinsic}$ is computed as a weighted sum of reward-per-grid, where grids that are positioned closer to the goal are weighted more. This goal-weighted intrinsic reward results in behaviour where the robot is

Reward Term	**Formula**	**Weight**
Sparse Task Reward	$\begin{cases} 50 & \text{if } d < 0.08 \\ 0 & \text{otherwise} \end{cases}$	1
Intrinsic Reward	$\sum_{ijk \in V} w_{ijk}\ e^{-10(n_{ijk}-1)}$	0.3

Table 3.5: The reward function is computed by multiplying each reward term by their weight and summing them up at each policy step.

biased to explore the broad region around the goal instead of wasting time exploring other irrelevant regions in the environment.

We employ a two-stage curriculum during training. In the first stage, only the intrinsic reward is active and the fabrics switch is manually set to 0; consequently, the actor policy has complete control over the robot's end-effector trajectory. In this stage, the policy is encouraged to explore its environment, forcing it to learn recovery behaviour to prevent getting trapped at local minima in order to reach more unvisited grids. This stage serves to instill the policy with useful priors relevant to the main task objective. The second stage of the curriculum is triggered once the unique grid visit count per episode reaches a certain threshold. Both the intrinsic reward and the sparse task reward are active, and the fabrics switch is allowed to be set to 1. Intuitively, in this stage, the policy mainly learns when to enable the fabrics switch that would allow geometric fabrics to converge to the goal.

It is worth noting that because the policy does not directly output joint actions for the robot but rather sends high-level commands for geometric fabrics, we do not need to add any motion regularization terms, such as joint velocity penalties, in the reward function since any motion produced by geometric fabrics is smooth and physically feasible on the robot hardware. As a result, leveraging geometric fabrics as a guiding medium for reinforcement learning significantly simplifies the process of reward function tuning for sim-to-real.

PPO Hyperparameter	Value
Clipping Parameter ϵ	0.2
Discount Factor γ	0.99
GAE λ	0.95
Learning Rate	5.00×10^{-4}

Table 3.6: Proximal Policy Optimization parameters.

3.4.6 PPO Training Setup

We use a high-performance PPO implementation from rl-games [19] in Isaac Gym with 4096 parallelized environments, trained on a single NVIDIA RTX 3090 GPU. We note that of these environments, half are dataset scenes and half are procedurally generated scenes. Table 3.6 outlines some of the PPO hyperparameters used in our training.

Chapter 4

Results and Discussion

We evaluate our neural motion generation method on 800 unique motion generation problems from the MπNets dataset. Each of these problems contains a pre-defined initial configuration for the Franka Panda manipulator and an end-effector goal pose with at least one guaranteed solution. The policies we employ in our evaluations and comparisons are trained for the Franka Panda manipulator with seven degrees-of-freedom, but our method generalizes to other manipulators with different kinematics, as demonstrated in Section 4.4.7 where we train and deploy a policy on a six degrees-of-freedom industrial arm. We note that all of our policies under evaluation were not trained on the MπNet dataset scenes.

4.1 Quantitative Metrics

We evaluate our method against other methods by rolling out a policy until the manipulator's mean keypoints distance error, as defined in Section 3.4.5, reaches below the threshold of 0.08 or the policy has been executed for more than 20 seconds. At the end of each roll out, we measure the following three quantitative metrics: completeness, collision rate, and reactiveness.

Completeness: Measured as the success rate of the policy on the 800 problems from the MπNets dataset, where the policy is deemed successful for a problem if the manipulator reaches a mean keypoints distance error of below 0.08 without resulting in any environment collisions or self-collisions.

Collision Rate: Measured as the percentage of problems in the MπNets dataset where rolling out a policy results in collisions with either the environment or the manipulator itself.

Reactiveness: Measured as the time it takes for a policy to generate an action for the manipulator given a new obstacle scene.

	Success Rate (%)	**Collision Rate (%)**	**Reaction Time (s)**
AIT*	100.0	0.0	16.46 ± 0.90
Geometric Fabrics	38.44	8.61	0.00024 ± 0.00003
MπNets	75.78	13.78	0.0068 ± 0.00007
CuRobo	99.92	0.0	0.089 ± 0.122
Ours	99.15	0.64	0.00279 ± 0.0001

Table 4.1: Performance comparison of methods on the MπNets dataset.

4.2 Quantitative Results and Comparisons

We compare the performance of our method against the following four methods: AIT*, Geometric Fabrics, MπNets, and CuRobo. AIT* is a state-of-the-art sampling-based global planner [40]. Geometric Fabrics is a state-of-the-art local reactive controller [46]. MπNets is a state-of-the-art neural motion generation policy [9]. CuRobo is a state-of-the-art optimization-based global planner [41]. Results of all five methods are summarized in Table 4.1.

Success Rate: Our method attains a success rate of 99.15% on the evaluation dataset. This result indicates that our method pertains some degree of global awareness as it yields a comparable performance to global planners such as AIT* and CuRobo while a locally-reactive controller, Geometric Fabrics, could only solve 38.44% of the problems. This result further indicates that our method of using basis point set to encode spatial information of the scene exhibits some capability of generalization to environments and goals previously unseen during training. On the other hand, the previous state-of-the-art neural motion generation method MπNets only achieves a success rate of 75.78%. Therefore, our method offers a significant improvement in the success rate metric over MπNets, hence achieving state-of-the-art performance in terms of completeness for neural motion generation methods. Figure 4.1 illustrates the success rate comparison.

Collision Rate: Our method achieves a collision rate of 0.64% on the evaluation dataset, whereas MπNets has a significantly higher collision rate of 13.78%. During RL training, we reset an environment if any collision occurs and applies a penalty accordingly in the reward function, which deters the policy from generating actions that result in collisions. We note that pure Geometric Fabrics has a collision rate of 8.61%, where the collisions occur mostly due to the lack of global motion planning, potentially moving the manipulator to regions with complex obstacle arrangements where Geometric Fabrics' local obstacle avoidance has a higher likelihood of failure. However, with our method, the RL policy guides Geometric Fabrics to avoid those high-risk regions in the environment since the policy possesses some degree of global awareness, which drastically

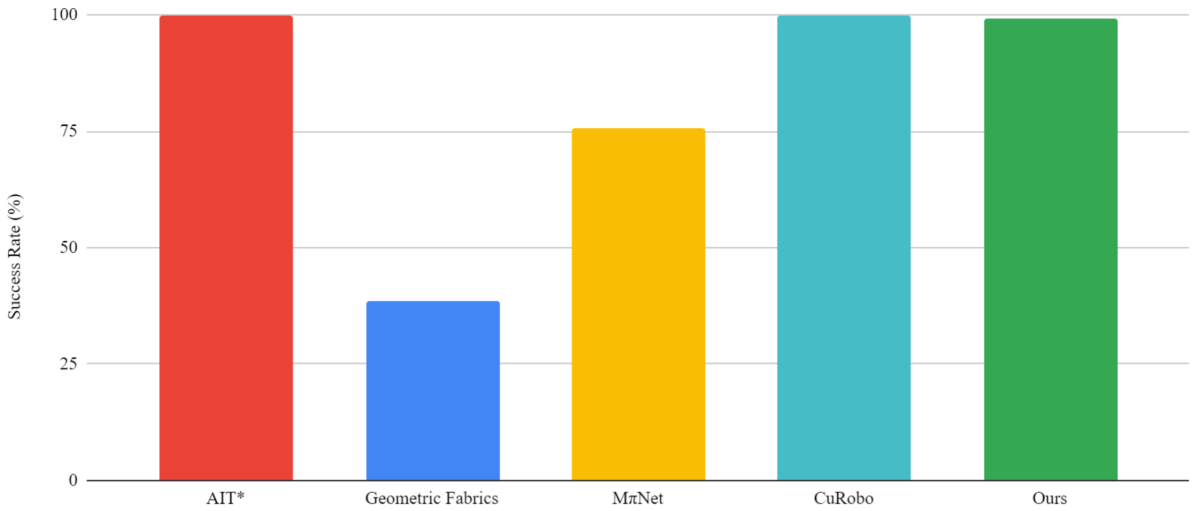

Figure 4.1: Success rate comparison between AIT*, Geometric Fabrics, MπNets, CuRobo, and our method.

reduces the method's collision rate. Figure 4.2 illustrates the success rate comparison.

Reaction Time: Our method yields a mean reaction time of 0.00279 seconds, whereas the mean reaction time of MπNets is 0.0068 seconds. We attribute our improved reaction time over MπNets to the fact that our policy neural network architecture has a faster inference speed than that of MπNets. Our policy network only comprises 3D CNNs and MLPs as we elect to use basis point set as our scene representation method. On the other hand, MπNets' network architecture contains PointNet++ [29] since it takes in point clouds as inputs, hence causing it to be more computationally expensive. Note that our method has a faster and a reaction time with less variance than CuRobo, making our method more suitable for unknown dynamic environments. Figure 4.3 illustrates the success rate comparison.

4.3 Qualitative Results

We showcase a representative selection of collision-free motion generation problems solved by our policy trained for a Franka Panda manipulator in the following video:
https://www.youtube.com/watch?v=0mYDXQe0ZJ8.

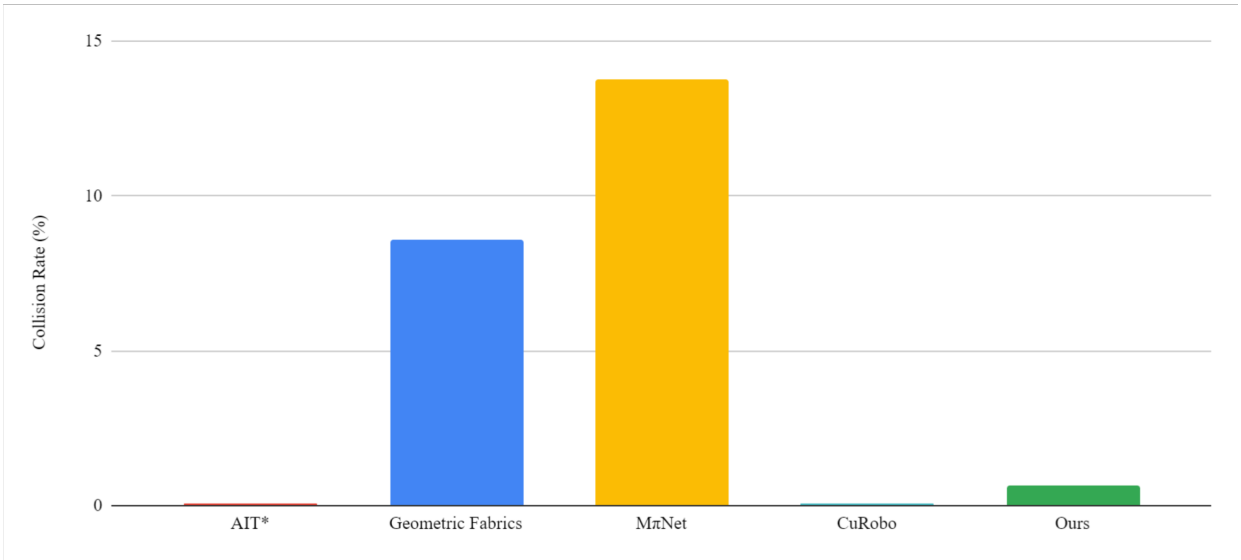

Figure 4.2: Collision rate comparison between AIT*, Geometric Fabrics, MπNets, CuRobo, and our method.

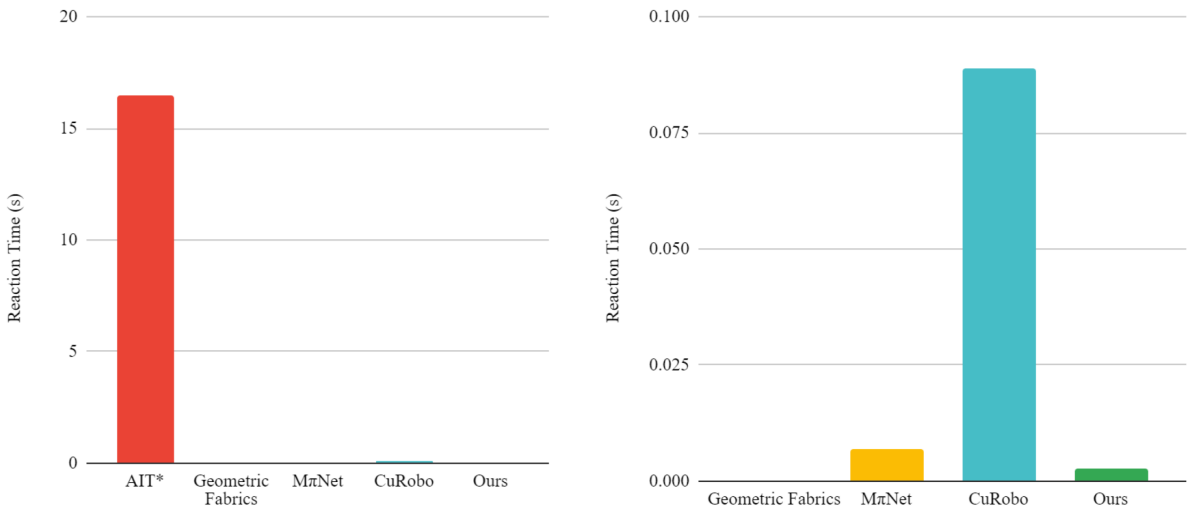

Figure 4.3: Reaction time comparison between AIT*, Geometric Fabrics, MπNets, CuRobo, and our method.

4.4 Ablations

We perform ablations to justify our method design choices regarding the following aspects of the method: task reward formulation, intrinsic reward, training curriculum, neural network architecture, policy-controlled joint-space, and exploration map.

4.4.1 Task Reward Formulation

We compare policies trained with the sparse task reward formulation discussed in Section 3.4.5 and policies trained with a dense task reward formulation defined as follows:

$$r_{dense} = \frac{1}{d + 0.25} \tag{4.1}$$

where d is the mean keypoints distance error as defined in Section 3.4.5. Figure 4.4 compares the two policies on the training set success rate and keypoints distance error, where sparse reward is represented in green and dense reward is represented in orange. We observe that the dense reward formulation fails to yield a policy with high success rate and low keypoints distance error; on the other hand, the sparse reward formulation results in a policy with above 95% training set success rate. This is because a dense task reward formulation would reward the agent despite getting stuck in one state without reaching the goal. This effect is especially problematic in situations where the end-effector of the manipulator is close to the goal in terms of its Euclidean distance, but successfully reaching the goal requires the end-effector to travel a longer path; in these scenarios, the dense reward would still yield a high reward in spite that the agent is not close to solving the task. Therefore, the sparse task reward formulation avoids this issue by only rewarding the agent for succeeding,

4.4.2 Performance Impacts of Intrinsic Reward

We compare policies trained with the sparse task reward with intrinsic reward and policies trained with only the sparse task reward. Figure 4.5 compares the two policies on the training set success rate and keypoints distance error, where policy trained with intrinsic reward is shown in green and the policy trained with only task reward is shown in orange. We observe that intrinsic reward improves the training set success rate of the policy by more than 10% and results in a lower keypoints distance error. This is likely because the intrinsic reward term actively encourages the agent to

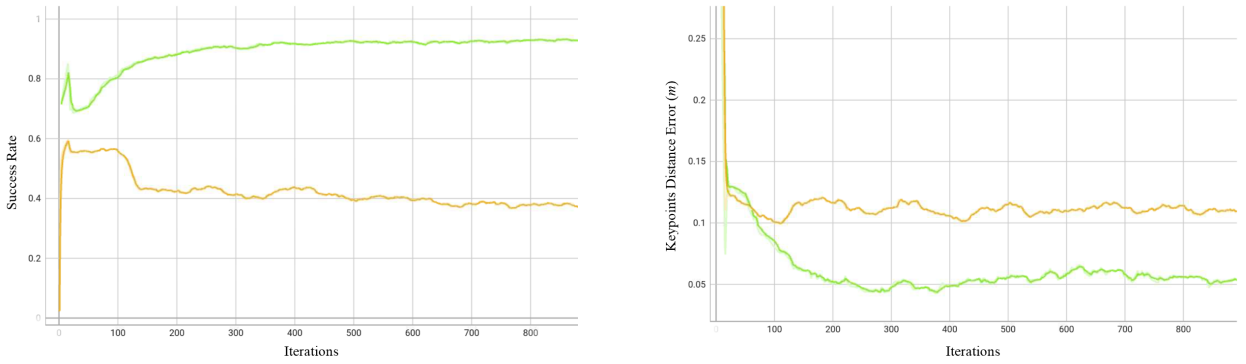

Figure 4.4: Training set success rate (left) and keypoints distance error (right) of policies trained with a sparse task reward (green) vs. a dense task reward (orange).

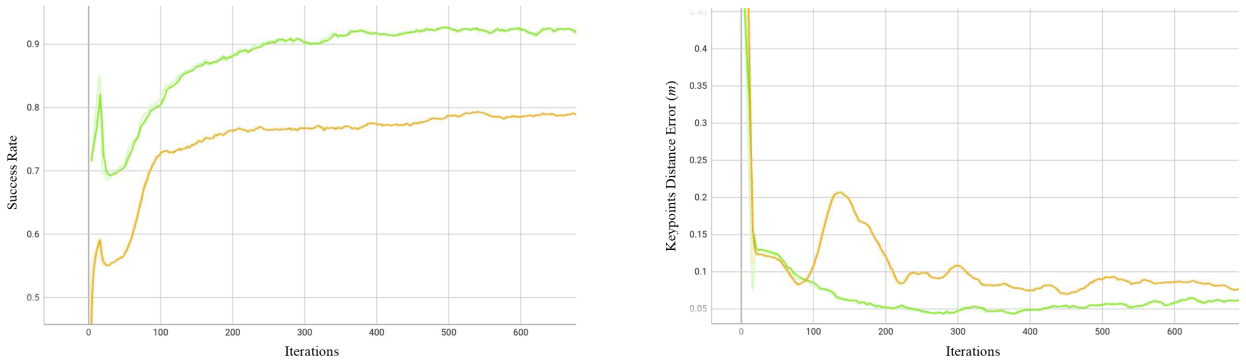

Figure 4.5: Training set success rate (left) and keypoints distance error (right) of policies trained with intrinsic + sparse task reward (green) vs. sparse task reward only (orange).

explore previously unvisited states, which alleviates situations where the manipulator would get stuck at the same state in regions with complex obstacle arrangements.

4.4.3 Performance Impacts of the Training Curriculum

We compare policies trained with the curriculum described in Section 3.4.5 and policies trained without the curriculum. Figure 4.6 compares the two policies on the training set success rate and keypoints distance error, where policy trained with the curriculum is shown in green and the policy trained without the curriculum is shown in orange. We note that the presence of the curriculum slightly improves both the keypoints distance error and the training set success rate of the policy. However, we observed a 5.2% increase in success rate and a 2.14% decrease in collision rate on the evaluation set for policies trained with the curriculum. We hypothesize that this is because the curriculum helps the agent to learn a more effective recovery behaviour by only rewarding the

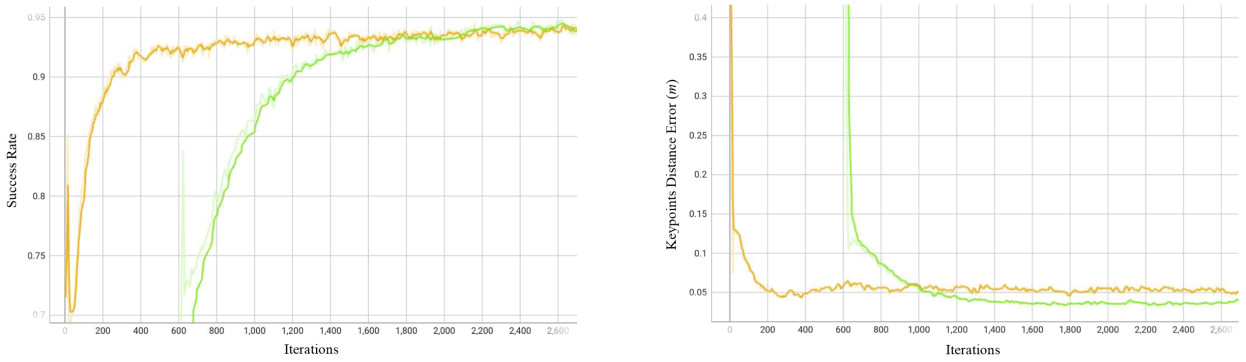

Figure 4.6: Training set success rate (left) and keypoints distance error (right) of policies trained with curriculum (green) vs. without curriculum (orange).

agent for exploring new grids in the first phase of the curriculum, facilitating the agent to escape from local minima.

4.4.4 Network Architecture

We compare policies trained with the multi-headed 3D CNN architecture with policies trained using a vanilla MLP with its parameters delineated in Table 4.2. Figure 4.7 compares the two policies on the training set success rate and keypoints distance error, where policy trained with CNNs is represented in green and the policy trained with a vanilla MLP is represented in orange. We observe a 25% increase in the training set success rate and a 46% increase in the evaluation set success rate with the multi-headed 3D CNN architecture in comparison to the vanilla MLP. We attribute this performance improvement to the inductive biases that CNNs offer, allowing the network to learn scene features more conducive to generalization to novel environments.

Network	**Layers**	**Parameters**
MLP	Fully Connected Layer + ELU 1	Units: 512
	Fully Connected Layer + ELU 2	Units: 512
	Fully Connected Layer + ELU 3	Units: 256
	Fully Connected Layer + ELU 4	Units: 128

Table 4.2: Parameters of the MLP-only actor and the critic network.

4.4.5 Performance Impacts of Policy-Controlled Joint-Space

As described in Section 3.4.3, we design the policy to output a joint-space target $\mathbf{q}_d$ for the geometric fabrics controller in order to posture the manipulator in its null-space. This section investigates the

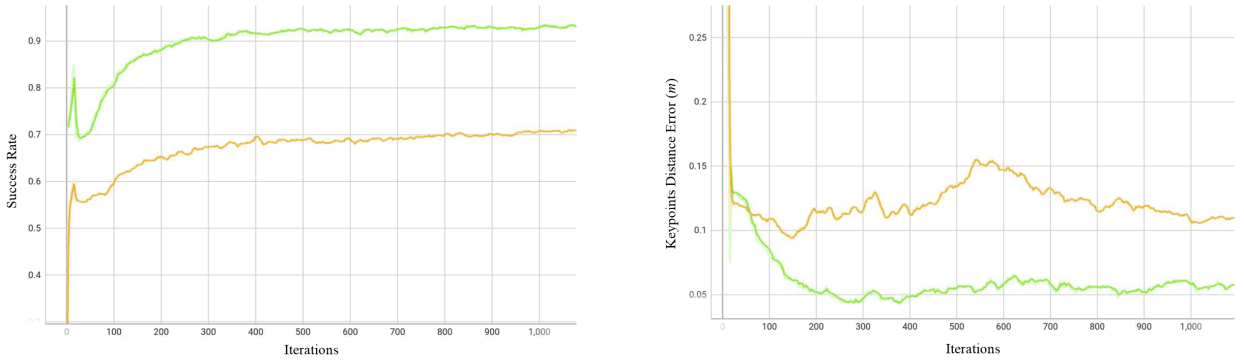

Figure 4.7: Training set success rate (left) and keypoints distance error (right) of policies trained with the multi-headed 3D CNN architecture (green) vs. a vanilla MLP (orange).

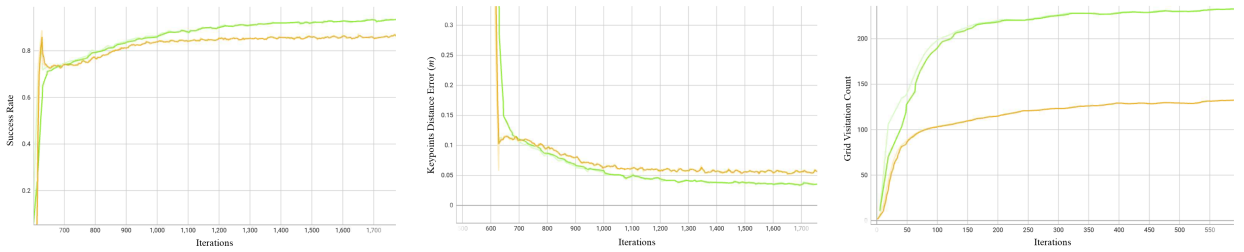

Figure 4.8: Training set success rate (left), keypoints distance error (middle), and grid visitation count (right) of policies that control the joint-space (green) and policies that do not have control over the joint-space (orange).

necessity of the joint-space target action from the policy. We compare the case where $\mathbf{q}_d$ is specified by the RL policy with the case where $\mathbf{q}_d$ is manually set to a default nominal configuration. Figure 4.8 compares the two policies on the training set success rate, the keypoints distance error, and the grid visitation count metrics, where the policy that controls the joint-space is represented in green and the policy that does not directly control the joint-space is represented in orange. Note that we only show the grid visitation count metric for the exploration phase of the curriculum in Figure 4.8. We observe that the policy that controls the joint-space achieves a 10% increase in the training set success rate and a lower keypoints distance error than a manually-defined joint-space target. In addition, we note that during the first phase of the training curriculum where the policy is encouraged to explore as many new states as possible, the policy that controls the joint-space is able to explore 1.75 times more grids than the alternative. This is because the policy that outputs $\mathbf{q}_d$ has more complete control over the configurations of the manipulator, allowing the manipulator to reach places that would otherwise be challenging with only end-effector control.

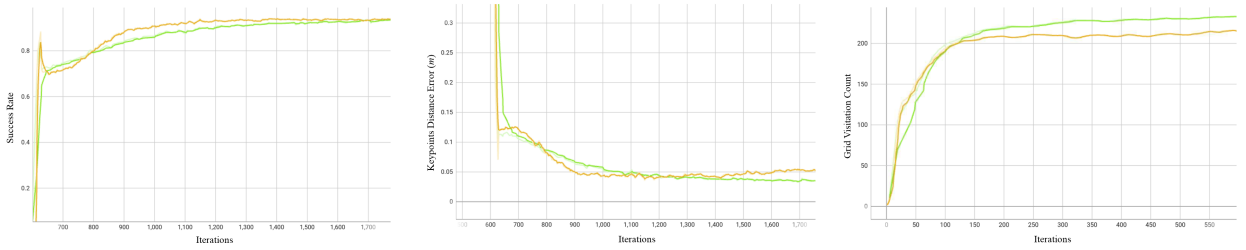

Figure 4.9: Training set success rate (left), keypoints distance error (middle), and grid visitation count (right) of policies trained with the exploration map (green) and policies trained without the exploration map (orange).

4.4.6 Performance Impacts of Using Exploration Map in Observation

We compare policies that include the exploration map as described in Section 3.4.3 as part of its observation with policies that do not take in the exploration map. Figure 4.9 compares the two policies on the training set success rate, the keypoints distance error, and the grid visitation count metrics, where policy trained with the exploration map is represented in green and the policy trained without it is represented in orange. Note that we only show the grid visitation count metric for the exploration phase of the curriculum in Figure 4.9. We observe that the inclusion of the exploration map yields a similar performance in the training set success rate metric, but a slightly lower keypoints distance error. However, on the grid visitation count metric, we observe an 8% improvement in the number of new grids visited during the exploration phase of the curriculum. This suggests that including the exploration map as part of the policy's observation may help with the agent's exploration of new states.

4.4.7 Real Robot Evaluation

In addition to training a neural motion generation policy for the 7-DOF Franka Panda manipulator with our method, we also train a neural motion generation policy for the 6-DOF Yaskawa Motoman NEX10 industrial manipulator. We attempt to deploy the trained policy on the NEX10 robot in the real world given the world model of its environment. Figure 4.10 exhibits the simulation environment and its corresponding real world environment during deployment. We observe some degree of zero-shot sim-to-real transfer without any domain randomization applied during training, but the behaviour generated by the policy is often sub-optimal. We suspect that this is mainly caused by various unmodeled real-world effects which were not accounted for during simulation training, such as observation delay and action delay; the sim-to-real gap induced by these effects can be drastically

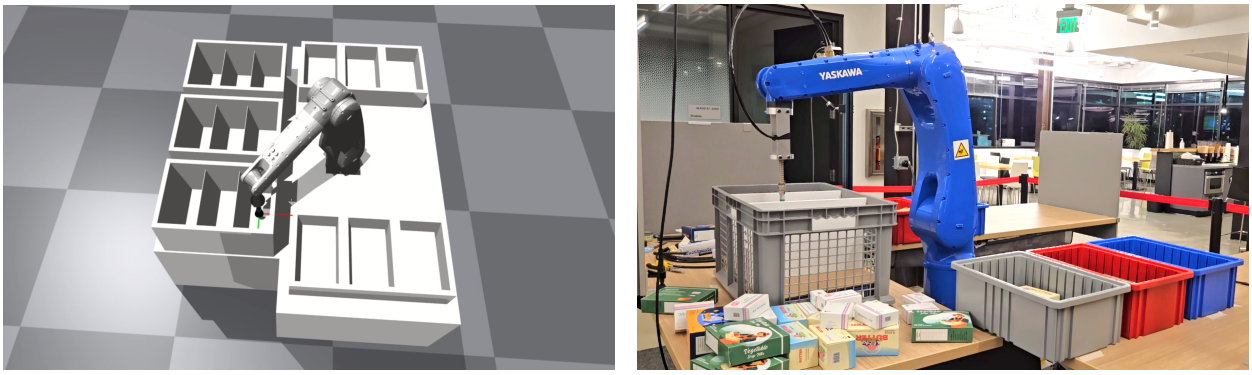

Figure 4.10: Simulation environment (left) and the real-world deployment (right) of the neural motion generation policy trained for the Yaskawa Motoman NEX10 manipulator.

reduced by explicitly modeling these effects during simulation training via domain randomization [3, 10]. Overall, more investigation needs to be conducted on the sim-to-real performance of the NEX10 neural motion generation policy.

Chapter 5

Conclusion

In summary, we have investigated a novel learning-based method for collision-free manipulator global motion generation. Firstly, we analysed the shortcomings of prior neural motion generation methods based on imitation learning, identifying imitation learning's difficulties in generalization to problems out of the distribution of the expert dataset. Secondly, we formulated neural motion generation as a reinforcement learning problem and proposed to train the reinforcement learning policy in the loop with geometric fabrics to reduce the extent for the policy to learn non-task specific behaviour and to ensure physically-feasible motion. Third, we experimented with various design decisions regarding training the reinforcement learning policy, discovering the following key results:

- Basis point set with 3D convolutions is an efficient and effective scene representation method to encode spatial information, enabling the reinforcement learning policy to condition on the scene.
- Count-based intrinsic reward improves the policy's success rate by encouraging exploration of new states, which helps the agent discover solutions to complex motion generation problems during training.
- Levering geometric fabrics as a safe guiding medium for the reinforcement learning policy to operate on ensures smoothness in the resulting motion, which obviates the need for various motion regularization penalties in the reward function, thereby simplifying reward function design and tuning.

Finally, we compare our method to other motion generation methods, finding that our method outperforms the previous state-of-the-art neural motion generation method in success rate, collision rate, and reaction time.

We note that we have not evaluated the policy's performance from taking in point clouds of the scene and converting the point clouds into basis point sets that can be taken in by the policy.

We suspect that in environments where only a partial observation can be obtained from point clouds, the performance of the neural motion generation policy could degrade. Therefore, future work could include training directly with partial observations of the scene or incorporating a scene completion module to reduce the potential sim-to-real gap of the policy.

Overall, this work presents a promising direction in using reinforcement learning with geometric fabrics to produce collision-free global motion generation policies for manipulators, serving as a robust foundation for manipulation tasks in potentially unknown and dynamic environments.

Bibliography

[1] Elie Aljalbout, Felix Frank, Maximilian Karl, and Patrick van der Smagt. On the role of the action space in robot manipulation learning and sim-to-real transfer, 2023.

[2] Arthur Allshire, Roberto Martín-Martín, Charles Lin, Shawn Manuel, Silvio Savarese, and Animesh Garg. Laser: Learning a latent action space for efficient reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6650–6656, 2021.

[3] Arthur Allshire, Mayank MittaI, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11802–11809, 2022.

[4] Arthur Aubret, Laëtitia Matignon, and Salima Hassas. A survey on intrinsic motivation in reinforcement learning. *CoRR*, abs/1908.06976, 2019.

[5] Dimitri Bertsekas. *Reinforcement learning and optimal control.* Athena Scientific, 2019.

[6] Mohak Bhardwaj, Balakumar Sundaralingam, Arsalan Mousavian, Nathan D. Ratliff, Dieter Fox, Fabio Ramos, and Byron Boots. STORM: An integrated framework for fast joint-space model-predictive control for reactive manipulation. In *5th Annual Conference on Robot Learning*, 2021.

[7] Constantinos Chamzas, Carlos Quintero-Peña, Zachary K. Kingston, Andreas Orthey, Daniel Rakita, Michael Gleicher, Marc Toussaint, and Lydia E. Kavraki. Motionbenchmaker: A tool to generate and benchmark motion planning datasets. *CoRR*, abs/2112.06402, 2021.

[8] Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. *arXiv preprint arXiv:2309.14341*, 2023.

[9] Adam Fishman, Adithyavairavan Murali, Clemens Eppner, Bryan Peele, Byron Boots, and Dieter Fox. Motion policy networks. In *6th Annual Conference on Robot Learning*, 2022.

[10] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, and Yashraj Narang. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984, 2023.

[11] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094, 2018.

[12] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5:90 – 98, 1985.

[13] Oussama Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE J. Robotics Autom.*, 3:43–53, 1987.

[14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[15] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

[16] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47):eabc5986, 2020.

[17] J. Parry Lewis. *Homogeneous Functions and Euler's Theorem*, pages 297–303. Palgrave Macmillan UK, London, 1969.

[18] Maxim Likhachev, Dave Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *International Conference on Automated Planning and Scheduling*, 2005.

[19] Denys Makoviichuk and Viktor Makoviychuk. rl-games: A high-performance framework for reinforcement learning. `https://github.com/Denys88/rl_games`, May 2021.

[20] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[21] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 405–421, Cham, 2020. Springer International Publishing.

[22] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023.

[23] Yashraj Narang, Kier Storey, Iretiayo Akinola, Miles Macklin, Philipp Reist, Lukasz Wawrzyniak, Yunrong Guo, Adam Moravanszky, Gavriel State, Michelle Lu, Ankur Handa, and Dieter Fox. Factory: Fast Contact for Robotic Assembly. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022.

[24] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[25] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, 2019.

[26] Xue Bin Peng. *Acquiring Motor Skills Through Motion Imitation and Reinforcement Learning.* PhD thesis, EECS Department, University of California, Berkeley, Dec 2021.

[27] Aleksei Petrenko, Arthur Allshire, Gavriel State, Ankur Handa, and Viktor Makoviychuk. Dexpbt: Scaling up dexterous manipulation for hand-arm systems with population based training, 2023.

[28] Sergey Prokudin, Christoph Lassner, and Javier Romero. Efficient learning on point clouds with basis point sets, 2019.

[29] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

[30] Ahmed Hussain Qureshi, Mayur J. Bency, and Michael C. Yip. Motion planning networks. *CoRR*, abs/1806.05767, 2018.

[31] Nathan Ratliff and Karl Van Wyk. Fabrics: A foundationally stable medium for encoding prior experience, 2023.

[32] Nathan D. Ratliff, Jan Issac, and Daniel Kappler. Riemannian motion policies. *CoRR*, abs/1801.02854, 2018.

[33] Nathan D. Ratliff, Karl Van Wyk, Mandy Xie, Anqi Li, and Muhammad Asif Rana. Generalized nonlinear and finsler geometry for robotics. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10206–10212, 2020.

[34] Nathan D. Ratliff, Karl Van Wyk, Mandy Xie, Anqi Li, and Muhammad Asif Rana. Optimization fabrics. *CoRR*, abs/2008.02399, 2020.

[35] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *5th Annual Conference on Robot Learning*, 2021.

[36] John Schulman, Yan Duan, Jonathan Ho, Alex X. Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33:1251 – 1270, 2014.

[37] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[38] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

[40] Marlin P. Strub and Jonathan D. Gammell. Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics. *CoRR*, abs/2002.06599, 2020.

[41] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. Curobo: Parallelized collision-free robot motion generation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8112–8119, 2023.

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

[43] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[44] Bingjie Tang, Michael A. Lin, Iretiayo Akinola, Ankur Handa, Gaurav S. Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. Industreal: Transferring contact-rich assembly tasks from simulation to reality, 2023.

[45] Johannes Tenhumberg, Darius Burschka, and Berthold Bäuml. Speeding up optimization-based motion planning through deep learning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7182–7189, 2022.

[46] Karl Van Wyk, Mandy Xie, Anqi Li, Muhammad Asif Rana, Buck Babich, Bryan Peele, Qian Wan, Iretiayo Akinola, Balakumar Sundaralingam, Dieter Fox, Byron Boots, and Nathan D. Ratliff. Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior. *IEEE Robotics and Automation Letters*, 7(2):3202–3209, 2022.

[47] Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. Viola: Imitation learning for vision-based manipulation with object proposal priors. *6th Annual Conference on Robot Learning (CoRL)*, 2022.

[48] Matthew Zucker, Nathan Ratliff, Anca Dragan, Mikhail Pivtoraiko, Matthew Klingensmith, Christopher Dellin, J. Andrew (Drew) Bagnell, and Siddhartha Srinivasa. Chomp: Covariant

hamiltonian optimization for motion planning. *International Journal of Robotics Research*, 32(9):1164 – 1193, August 2013.