STRUCTURED LEARNING AND INFERENCE FOR ROBOT MOTION GENERATION

A Dissertation
Presented to
The Academic Faculty

By

Mustafa Mukadam

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in Robotics

School of Electrical and Computer Engineering
Georgia Institute of Technology

August 2019

STRUCTURED LEARNING AND INFERENCE FOR ROBOT MOTION GENERATION

Approved by:

Dr. Byron Boots, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Frank Dellaert
School of Interactive Computing
Georgia Institute of Technology

Dr. Sonia Chernova
School of Interactive Computing
Georgia Institute of Technology

Dr. Evangelos Theodorou
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Nathan Ratliff
NVIDIA Research

Date Approved: July 10, 2019

If I have seen further it is by standing on the shoulders of giants.

Isaac Newton

To my parents, my sister, and my wife.

ACKNOWLEDGMENTS

It has been a long and strenuous journey that would not have been possible to complete without the help and support of many many people along the way. Here, I would like to express my gratitude towards them, even though words nevertheless may prove insufficient.

Foremost, I want to thank my advisor Prof. Byron Boots. I am grateful to him for providing me the environment and freedom to pursue topics in which I was interested, and transforming my sometimes preposterous ideas into tangible research directions. He taught me the value of collaboration, introduced me to machine learning, and gave me countless invaluable advice on career and all things life. I am forever in indebted to him for his constant guidance and support, and for shaping me to become an independent researcher. It has been an absolute privilege being a part of his lab.

I am honored to have exceptional committee members Prof. Frank Dellaert, Prof. Sonia Chernova, Prof. Evangelos Theodorou, and Dr. Nathan Ratliff, and I am thankful for their extremely valuable guidance and insightful feedback on my research. I am very grateful to Frank, Sonia, Nathan, and Prof. Dieter Fox for being awesome collaborators throughout the major beats of this thesis. I have learned a tremendous deal from them and much of this work would not be possible without their mentorship. I also want to thank Prof. Tim Bretl and Prof. Seth Hutchinson for introducing me to the world of motion planning and robotics research, and for inspiring me to pursue a Ph.D.

During this time, I have had the opportunity to intern at some amazing places, where I gained a great deal of valuable experience. I want to thank Dieter and Nathan at NVIDIA Research, Dr. Joey Durham and Dr. Jong Jin Park at Amazon Robotics, and Dr. Alireza Nakhaei and Dr. Akansel Cosgun at Honda Research Institute for their mentorship.

I have had the pleasure of collaborating with many excellent researchers, colleagues, and friends. I want to thank Jing Dong, Asif Rana, Ching-An Cheng, Xinyan Yan, Mohak Bhardwaj, Anqi Li, Reza Ahmadzadeh, Harish Ravichandar, Sasha Lambert, Eric Huang,

Shray Bansal, and others for the fruitful projects, discussions, and mutual learning. I am also thankful to Jing for teaching me how to write better code and about software engineering in general. Many thanks also to NSF and NVIDIA for providing funding support.

With RoboGrads and IRIM, I have been exceedingly lucky to have an amazing community and network at Georgia Tech and the time I spent here has been one of the most rewarding experiences of my life. I would especially like to thank my labmates in the Georgia Tech Robot Learning Lab and the Robot Autonomy and Interactive Learning Lab.

Finally, I want to thank my family for their incredible support and unconditional love. Thank you to my parents Mubina and Huseini, my sister Alifya, and my wife Zaitun for all their infinite sacrifices and for believing in me every step of the way. Thank you for being my pillars and celebrating with me through the highs of this journey, but also keeping me motivated and inspired through the lows. I would not be where I am today without them.

TABLE OF CONTENTS

Acknowledgments v

List of Tables xiii

List of Figures xiv

Chapter 1: Introduction 1

I Planning as Inference **4**

Chapter 2: Gaussian Process Motion Planning 5

2.1 Introduction 5

2.2 Related work 7

2.3 Motion planning as trajectory optimization 10

2.4 Gaussian processes for continuous-time trajectories 11

2.4.1 The GP prior 11

2.4.2 A Gauss-Markov model 12

2.4.3 Gaussian process interpolation 14

2.5 Motion planning with Gaussian processes 17

2.5.1 Cost functionals 17

2.5.2 Optimization 18

2.5.3 Compact trajectory representations and faster updates 20
2.6 Motion planning as probabilistic inference 22
2.7 Structure with factor graphs 24
2.8 Incremental inference for fast replanning 29
2.9 Implementation details 31
2.9.1 GPMP 31
2.9.2 GPMP2 and iGPMP2 33
2.10 Evaluation 37
2.10.1 Batch planning benchmark 38
2.10.2 Incremental planning benchmark 43
2.11 Discussion 45

II Learning on Factor Graphs 48

Chapter 3: Combining LfD and Motion Planning 49

3.1 Introduction 49
3.2 Related work 50
3.3 The trajectory prior as a skill model 52
3.3.1 Structured heteroscedastic GPs 53
3.3.2 A combined prior 55
3.3.3 Learned workspace prior 56
3.4 Efficient inference via factor graphs 57
3.4.1 Prior factors 57
3.4.2 Likelihood factors 58

3.4.3 Skill reproduction 59
3.5 Evaluation 59
3.6 Discussion 62

Chapter 4: Differentiable Inference-based Planning 64
4.1 Introduction and related work 64
4.2 Sensitivity to objective function parameters 66
4.3 A computational graph for planning 68
4.4 Evaluation 71
4.4.1 Implementation details 71
4.4.2 Learning on complex distributions 73
4.4.3 Planning with velocity constraints 75
4.5 Discussion 76

III Reactive Policy Synthesis 77

Chapter 5: Riemannian Policies for Reactive Motion 78
5.1 Introduction 78
5.2 Motion generation and control 80
5.3 Related work 82
5.4 From operational space control to geometric control 83
5.5 Structure with task-map trees 97
5.5.1 Task-maps 97
5.5.2 Riemannian motion policies 98
5.5.3 RMP-tree 99

5.6 Automatic motion policy generation . . . 99
5.6.1 RMP-algebra . . . 99
5.6.2 Algorithm . . . 101
5.6.3 Example RMPs . . . 102
5.7 Theoretical analysis of RMPflow . . . 103
5.8 Applying RMPflow on a simple example problem . . . 107
5.9 Evaluation . . . 109
5.9.1 Controlled experiments . . . 109
5.9.2 System experiments . . . 111
5.10 Discussion . . . 118

IV Learning on Task-Map Trees 121

Chapter 6: Learnable Policy Fusion . . . 122
6.1 Introduction and related work . . . 122
6.2 Modifying RMPflow with multiplicative weight functions . . . 123
6.3 End-to-end learning of weight functions . . . 128
6.4 Evaluation . . . 130
6.4.1 2D robot . . . 130
6.4.2 Franka robot . . . 135
6.5 Discussion . . . 139

Chapter 7: Learning Reactive Motion Policies . . . 140
7.1 Introduction . . . 140
7.2 Related work . . . 141

7.3 Skill reproduction via RMPflow . . . 143
7.3.1 Human-guided Riemannian motion policies . . . 144
7.3.2 Learning warped potentials from demonstration . . . 146
7.3.3 Admissible metric for policy resolution . . . 149
7.4 Evaluation . . . 150
7.5 Discussion . . . 154

Chapter 8: Conclusion . . . 155

Appendix A: Prior and Sparsity in Gaussian Process Motion Planning . . . 160
A.1 The trajectory prior . . . 160
A.2 Sparsity of the likelihood . . . 162

Appendix B: Geometric Dynamical Systems . . . 164
B.1 From geometric mechanics to GDSs . . . 164
B.2 Degenerate GDSs . . . 164
B.3 Geodesic and stability . . . 165
B.4 Curvature term and Coriolis force . . . 165

Appendix C: Proofs of RMPflow Analysis . . . 167
C.1 Proof of Theorem 1 . . . 167
C.2 Proof of Proposition 1 . . . 173
C.3 Proof of Theorem 2 . . . 175

Appendix D: RMPflow and Recursive Newton-Euler . . . 177

Appendix E: Proofs of RMPfusion Analysis 179

References 197

LIST OF TABLES

2.1 Results for 24 planning problems on the 7-DOF WAM arm. 41

2.2 Results for 198 planning problems on PR2's 7-DOF right arm. 41

2.3 Average number of optimization iterations on successful runs. 42

2.4 Results for 72 replanning problems on WAM. 44

2.5 Results for 54 replanning problems on PR2. 44

4.1 Comparison of dGPMP2 versus GPMP2 with fixed hand tune covariances. dGPMP2 learns the obstacle covariance σ_{obs} using training set of 5000 environments. $\mathbf{Q}_C = 0.5 \times I$ for all. 75

4.2 Performance of dGPMP2 with velocity constraints on different combinations of training and testing. Mild constraints are $v_{xmax} = 1.5m/s$, $v_{ymax} = 1.5m/s$, and $time = 15s$, tight constraints are $v_{xmax} = 1.0m/s$, $v_{ymax} = 1.0m/s$, and $time = 10s$ for the same start and goal. 75

LIST OF FIGURES

1.1 State-of-the-art robotic systems designed for diverse applications operating in their respective domains. (from left to right) Atlas humanoid (source: Boston Dynamics), Curiosity Mars rover (source: NASA JPL), da Vinci Surgical System (source: Intuitive Surgical), Kiva Drives (source: Amazon Robotics). 2

2.1 Optimized trajectory found by GPMP2 is used to place a soda can on a shelf in simulation (left) and with a real WAM arm (middle left). Examples of successful trajectories generated by GPMP2 are shown in the countertop (middle right) and lab (right) environments with the PR2 and WAM robots respectively. 6

2.2 An example GP prior for trajectories. The dashed line is the mean trajectory $\boldsymbol{\mu}(t)$ and the shaded area indicates the covariance. The 5 solid lines are sample trajectories $\boldsymbol{\theta}(t)$ from the GP prior. 12

2.3 An example that shows the trajectory at different resolutions. Support states parameterize the trajectory, collision cost checking is performed at a higher resolution during optimization and the output trajectory can be up-sampled further for execution. 15

2.4 An example showing how GP interpolation is used during optimization. (a) shows the current iteration of the trajectory (black curve) parameterized by a sparse set of support states (black circles). GP regression is used to densely up-sample the trajectory with interpolated states (white circles). Then, in (b) cost is evaluated on all states and their gradients are illustrated by the arrows. Finally, in (c) the cost and gradient information is propagated to just the support states illustrated by the larger arrows such that only the support states are updated that parameterize the new trajectory (dotted black curve). 16

2.5 A factor graph of an example trajectory optimization problem showing support states (white circles) and four kinds of factors (black dots), namely prior factors on start and goal states, GP prior factors that connect consecutive support states, obstacle factors on each state, and interpolated obstacle factors between consecutive support states (only one shown here for clarity, any number of them may be present in practice). 25

2.6 Example of a Bayes Tree with its corresponding factor graph. 30

2.7 Replanning examples using Bayes Trees. Dashed boxes indicate parts of the factor graphs and Bayes Trees that are affected and changed while performing replanning. 31

2.8 The WAM arm is represented by multiple spheres (pink), which are used during collision cost calculation. 33

2.9 The likelihood function $\mathbf{h}$ in a 2D space with two obstacles and $\epsilon = 0.1\text{m}$. Obstacles are marked by black lines and darker area has higher likelihood for no-collision. 35

2.10 Environments used for evaluation with robot start and goal configurations showing the WAM dataset (left), and a subset of the PR2 dataset (*bookshelves* (center) and *industrial* (right)). 37

2.11 (a) shows a successful trajectory with a good selection of σ_{obs}; (b) shows failure, where the trajectory collides with the top part of the shelf, when σ_{obs} is too large. 39

2.12 Breakdown of average timing per task **per iteration** on all problems in the WAM dataset is shown for CHOMP, GPMP-no-intp, GPMP-intp, GPMP2-no-intp and GPMP2-intp. 42

2.13 Example iGPMP2 results on the WAM and PR2 *industrial*. Red lines show originally planned end-effector trajectories, and green lines show replanned end-effector trajectories. Best viewed in color. 44

3.1 Block diagram showing various components of CLAMP. 53

3.2 Example factor graphs of (a) the prior distribution, and the joint distribution of the prior and the likelihood when the likelihood describes events associated with (b) different start conditions or (c) obstacle avoidance and different start conditions. States $\boldsymbol{\theta}_i$ are shown as white circles. 58

3.3 Demonstration and reproduction of *box-opening* (top) and *drawer-opening* (bottom). 60

3.4 Top: Position workspace priors shown in 3D; Middle: Position workspace priors plotted against time; Bottom: Velcocity workspace priors plotted againt time. The mean is in blue with an envelope showing the 95% confidence. Demonstrations are overlayed. 61

3.5 Reproduced position trajectories in red from different initial states. The obstacle is in yellow and the prior position mean is in blue. 62

4.1 (a)-(b) `tarpit` dataset (robot radius = 0.4m, safety distance = 0.4m). For the same $\mathbf{Q}_C$, a smaller σ_{obs} is required to encourage the planner to navigate around obstacles. (c)-(d) `forest` dataset (robot radius = 0.2m, safety distance = 0.2m). For the same $\mathbf{Q}_C$, a larger σ_{obs} is required to focus on finding solutions near the straight line trajectory. (e)-(f) `multi_obs` dataset (robot radius = 0.4m, safety distance = 0.4m) A small change in obstacle covariance can lead to significant changes in the trajectory. In all figures, the red dashed trajectories are the initializations and the blue trajectories are the optimized solutions. 66

4.2 The computational graph of dGPMP2 where $\boldsymbol{\phi}_F$ represents some user defined planning parameters that are fixed and $\boldsymbol{\phi}_L$ represents the learned planning parameters. See text for details. 69

4.3 Example comparison of (d) dGPMP2 against (b)-(c) GPMP2 (fixed hand tuned covariances) and (a) Expert on `forest` (top row) and `tarpit` (bottom row) datasets. Hand tuned covariances that work well on one distribution of obstacles fail on the other and vice versa. By imitating the expert, dGPMP2 is able to perform consistently across different environment distributions. Green circle is start, cyan is goal, dashed red line is initialization, and $\mathbf{Q}_c = 0.5 \times I$, $r = 0.4m$ for all. Trajectory is in collision if at any state the signed distance between robot center of mass and nearest obstacle is less than or equal to r. 72

5.1 Tree-structured task maps 98

5.2 (a) A 2-DOF planar manipulator tasked with avoiding an obstacle (red) and reaching a goal (green). Task-map tree for the problem in (a) being built starting from (b) the main kinematic chain to (c) all kinematic body point locations to (d) full task-map tree consisting of all abstract task spaces for obstacle avoidance (red) and target attraction (green). 108

5.3 Phase portraits (gray) and integral curves (blue; from black circles to red crosses) of 1D example. (a) Desired behavior. (b) With curvature terms. (c) Without curvature terms. (d) Without curvature terms but with nonlinear damping. 110

5.4 2D example; initial positions (small circle) and velocities (arrows). (a-d) Obstacle (circle) avoidance: (a) w/o curvature terms and w/o potential. (b) w/ curvature terms and w/o potential. (c) w/o curvature terms and w/ potential. (d) w/ curvature terms and w/ potential. (e) Combined obstacle avoidance and goal (square) reaching. 110

5.5 Task-map tree used in the system experiments. 112

5.6 Two of the six simulated worlds in the reaching experiments (left), and the two physical dual-arm platforms in the full system experiment (right). . . . 114

5.7 Results for reaching experiments. Though some methods achieve a shorter goal distance than RMPflow in successful trials, they end up in collision in most the trials. 115

5.8 From left to right using RMPflow the YuMi robot opens a drawer with one arm and with the other arm picks and places a banana in that drawer. 117

6.1 Franka robot navigating around an obstacle using RMPfusion. 123

6.2 (a) Shows the network used for learning with RMPfusion, specifically for any node i on the RMP-tree*, with children $c_0, \ldots, c_j$. If i is a leaf node, then it is evaluated from the designed RMP policy. The global policy is obtained by applying `resolve` on the root node RMP. RMP-tree* used in experiments for (a) `2d1level` and (b) `2d2level`. 129

6.3 Trajectories generated in (a) `2d1level` and in `2d2level` by (b) `learner-rmp` and (e) `learner-un`, compared to the expert are shown. Initial state is a black circle for position and black arrow for velocity. The environment has obstacles (red and blue) and goal (orange square). (c) shows the corresponding energy function for `learner-rmp` trajectories in (b). (d) shows the learning curve for `learner-rmp`. 132

6.4 Improvement of the behavior produced by `learner-rmp` at various stages during training for `2d2level`. The top row shows the trajectories and the bottom row shows the corresponding energy function. From left to right these plots correspond to the red dots from left to right on the training curve in Figure 6.3d. 132

6.5 (b) Trajectories generated in `2d2level` by `learner-rmp-large` compared to the expert is shown. Initial state is a black circle for position and black arrow for velocity. The environment has obstacles (red and blue) and goal (orange square). Learning curves for (a) `learner-rmp` and (c) `learner-rmp-large` on `2d2level` is also shown. 134

6.6 Trajectories produced by `learner-un` at various stages during training for `2d2level`. From left to right these plots correspond to the red dots from left to right on the training curve in Figure 6.5a. 134

6.7 Trajectories produced by `learner-un-large` at various stages during training for `2d2level`. From left to right these plots correspond to the red dots from left to right on the training curve in Figure 6.5c. 134

6.8 RMP-tree* used for the Franka robot. Gray nodes show task spaces, blue nodes show subtask RMPs, and weight functions are shown on the respective edges where they are defined. See text for more details. 135

6.9 An example from the test and training dataset (left) and the validation dataset (right). The robot is shown in its start configuration with an obstacle (cylinder) and a goal (sphere). 136

6.10 Learner's performance with respect to the expert on the validation dataset for the experiments with the Franka robot. 137

6.11 (a)-(d) An example execution (left to right) from the validation dataset, comparing (a) the expert with (b) `learner-0`, (c) `learner-300`, and (d) `learner-1200`. (e) The respective total energy profiles of the learners' trajectories (`learner-0` (left), `learner-300` (middle), `learner-1200` (right)). 138

7.1 Sawyer robot performing a complex manipulation task in a constrained environment, by coordinating the movements of different parts of its arm. . . . 141

7.2 Example RMP-tree with root node as the configuration space q, white nodes are other task spaces on the kinematic tree. There two types of leaf nodes with learned (blue) and hand designed (grey) RMPs. 143

7.3 *Left*: Visualization of *warped potential* learned from a demonstration overlayed on top in red. The arrows show the negative gradient direction. *Right*: Isocontours representing points equidistant to the target on the underlying manifold defined by the metric that stretches space in the direction of the potential gradient. 146

7.4 Sequence of images illustrating qualitative differences between learning leaf RMPs (a) only for the end-effector (*top row*), and (b) for all the robot link reference frames (*bottom row*) when starting from similar configurations. 151

7.5 Time evolution of mean squared errors between reproduced link frame trajectories and the corresponding demonstrations over time for three different robot initial configurations (init-0 through init-1). The *top-row* visualizes errors for an RMP-tree with a human-guided leaf RMP defined solely on the end-effector, while *bottom-row* corresponds to an RMP-tree with human-guided leaf RMPs defined on all the robot link reference frames. 152

7.6 The combination of obstacle avoidance RMP and human-guided RMPs results in motions that are guaranteed to avoid collisions while simultaneously attempting to satisfy the task constraints. 153

SUMMARY

The ability to generate motions that accomplish desired tasks is fundamental to any robotic system. Robots must be able to generate such motions in a safe and feasible manner, sufficiently quickly, and in dynamic and uncertain environments. In addressing these problems, there exists a dichotomy between traditional methods and modern learning-based approaches. Often both of these paradigms exhibit complementary strengths and weaknesses, for example, while the former are interpretable and integrate prior knowledge, the latter are data-driven and flexible to design. In this thesis, I present two techniques for robot motion generation that exploit structure to bridge this gap and leverage the best of both worlds to efficiently find solutions in real-time. The first technique is a planning as inference framework that encodes structure through probabilistic graphical models, and the second technique is a reactive policy synthesis framework that encodes structure through task-map trees. Within both frameworks, I present two strategies that use said structure as a canvas to incorporate learning in a manner that is generalizable and interpretable while maintaining constraints like safety even during learning.

CHAPTER 1
INTRODUCTION

Across any domain, like a house, hospital, or even the surface of Mars, robots must posses the capability to autonomously navigate and manipulate their surroundings by generating motions (Figure 1.1). We use *motion generation* as a blanket term to refer to approaches that utilize intrinsic, extrinsic, and task relevant information to compute and execute motions. Finding desirable and practical solutions to motion generation can be a monstrous challenge. It requires many considerations like, high-degree-of-freedom spaces, model inaccuracy, execution stochasticity, noisy sensors, dynamic environments, task and robot dependent constraints, in the face of limited onboard computational resources.

Within the planning and control literature many methods have been proposed to tackle this problem by dealing with various subsets of the considerations above and utilizing techniques that can be broadly grouped into search [1, 2, 3], sampling [4, 5, 6], trajectory optimization [7, 8, 9, 10, 11, 12], and inference [13, 14, 15, 16]. These approaches generally rely on domain knowledge, models, and problem constraints to offer interpretability and safety. However, they can be hard to design for complex tasks and are limited by the fact that they often do not utilize past experience or computation. Thus, they require hand engineering and expert oversight to apply them to settings beyond their initial problem abstraction.

Over the last few years, machine learning has led to many successes in the areas of computer vision and natural language, as a result of advances in deep learning, access to large amounts of data, and increasingly powerful computing hardware. Naturally, this has generated heavy interest in applying these tools, successful in other domains, to solve problems in robotics. Early work begun extending these ideas largely with a revisitation of reinforcement learning for playing games [17, 18] and later to move robots [19, 20, 21,

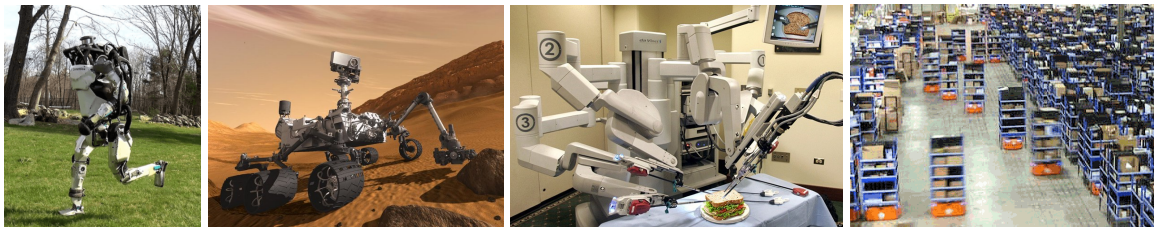


Figure 1.1: State-of-the-art robotic systems designed for diverse applications operating in their respective domains. (from left to right) Atlas humanoid (source: Boston Dynamics), Curiosity Mars rover (source: NASA JPL), da Vinci Surgical System (source: Intuitive Surgical), Kiva Drives (source: Amazon Robotics).

22, 23, 24]. Being inherently data-driven, learning offers opportunities to set up systems that are more broadly applicable and can utilize prior computation. It can address complex tasks that are difficult to program by hand, and can allow the system to evolve and adapt to new situations on the fly. However, much of the current approaches have been applied only in simulation [22, 20]. This limitation is being addressed with work in transferring from simulation to the real world [25, 26], but that brings with it its own set of unique problems. Other work that is applied on real robots commonly relies on networks that transform raw inputs, like images, directly to action commands [21, 24]. Recent work, has also revealed challenges in reproducing and benchmarking common deep reinforcement learning methods [27].

The progress currently achieved in the context of robotics with a naive application of modern machine learning tools mostly seems to be exhausted. Interpretability is usually only available in the form of low-level image features from convolution networks and therefore the decision making of complex robotic systems with sophisticated policy behaviors can be difficult to understand and analyze. Ensuring safety constraints becomes challenging when the underlying problem structure is neglected. Inability to incorporate physics, prior knowledge, and constraints leads to data inefficiency and the learning process becomes slower, made doubly worse by the fact that acquiring large amounts of data on real robotic systems is often infeasible. These tools for learning are powerful, but what's lacking are principled and systematic ways of leveraging them in robotics.

Recognizing the complementary strengths and weaknesses of traditional methods with built in domain knowledge and strong priors, and modern methods with the power and flexibility of machine learning, by employing ***structure*** *as the pivotal component, in the context of robot motion generation, we can bridge the gap between the two paradigms.*

The role of structure has been a topic of recent interest among many other communities like optimization [28, 29, 30], control [31, 32] and perception [33, 34] stemming from a similar dichotomy between the two paradigms.

In this thesis, I present two novel overarching techniques for efficient robot motion generation with systematic integration of learning for real world systems. First is a planning as inference framework in Part I, where structure manifests with probabilistic graphical models that are utilized to incorporate learning in two distinct ways in Part II. Second is a reactive policy synthesis framework in Part III, where structure manifests with task-map trees that are utilized to incorporate learning in two distinct ways in Part IV.

In both of these frameworks, the structure is used to encode domain knowledge and problem constraints such that with the modularity induced therein, we can selectively learn to represent parts of the problem. Effectively, the existing framework can still be utilized to solve the full problem, and the non-learning and learning components can work in synergy. Consequently, I show that constraints like safety can be enforced by the non-learning components even when learning is in progress. Along similar lines, broader generalization is possible, wherein the learning components can easily transfer to new scenarios with the help from the non-learning components. I also show that these representations are highly interpretable and thus provide full transparency into how the system's learning progresses and how it eventually makes decisions.

Part I

Planning as Inference

CHAPTER 2

GAUSSIAN PROCESS MOTION PLANNING

2.1 Introduction

Motion planning is a key tool in robotics, used to find trajectories of robot states that achieve a desired task. While searching for a solution, motion planners evaluate trajectories based on two criteria: *feasiblity* and *optimality*. The exact notion of feasibility and optimality can vary depending on the system, tasks, and other problem-specific requirements. In general, feasibility evaluates a trajectory based on whether or not it respects the robot or task-specific constraints such as avoiding obstacles, while reaching the desired goal. In other words, feasibility is often binary: a trajectory is feasible or it is not. In contrast with feasibility, optimality often evaluates the quality of trajectories without reference to task-specific constraints. For example, optimality may refer to the smoothness of a trajectory and encourage the motion planner to minimize dynamical criteria like velocity or acceleration. A variety of motion planning algorithms have been proposed to find trajectories that are both feasible and optimal.

In this chapter, we adopt a continuous-time representation of trajectories; specifically, we view trajectories as functions that map time to robot state. We assume these functions are sampled from a Gaussian process (GP) [35]. We will show that GPs can inherently provide a notion of trajectory optimality through a *prior*. Efficient structure-exploiting GP regression (GPR) can be used to query the trajectory at any time of interest in $O(1)$. Using this representation, we develop a gradient-based optimization algorithm called GPMP (Gaussian Process Motion Planner) [36] that can efficiently overcome the large computational costs of fine discretization while still maintaining smoothness in the result.

Through the GP formulation, we can view motion planning as probabilistic inference [15,

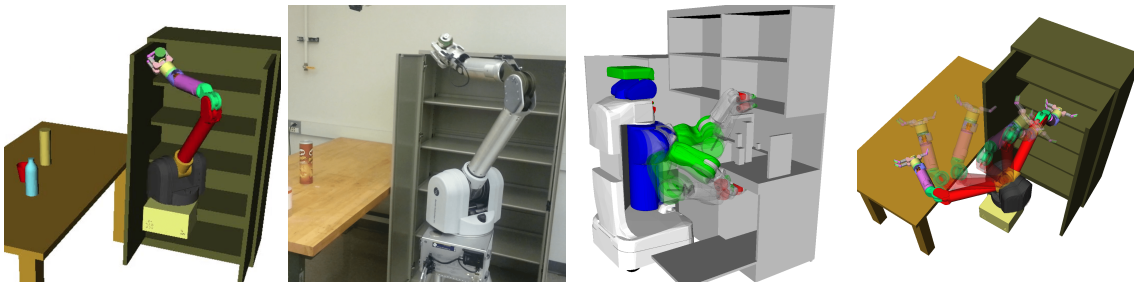


Figure 2.1: Optimized trajectory found by GPMP2 is used to place a soda can on a shelf in simulation (left) and with a real WAM arm (middle left). Examples of successful trajectories generated by GPMP2 are shown in the countertop (middle right) and lab (right) environments with the PR2 and WAM robots respectively.

37]. Similar to how the notion of trajectory optimality is captured by a *prior* on trajectories, the notion of feasibility can also be viewed probabilistically as well and encoded in a likelihood. We encode structure in to the inference problem using a probabilistic graphical model called factor graphs [38]. We illustrate the duality between inference and optimization and show how to solve the *maximum a posteriori* (MAP) inference problem. We will see how this optimization can be done in an efficient manner by exploiting the underlying problem structure encoded through factor graphs. Similar techniques have been used to solve large-scale Simultaneous Localization and Mapping (SLAM) problems [39]. With this key insight we can leverage preexisting efficient optimization tools developed by the SLAM community, and use them in the context of motion planning. With this we provide the GPMP2 algorithm [40, 41], which is more efficient than previous motion planning algorithms. Another advantage of GPMP2 is that we can easily extend the algorithm using techniques designed for incremental inference on factor graphs developed in the context of SLAM. For example, Incremental Smoothing and Mapping (iSAM) [42, 43] can be adapted to efficiently solve replanning problems. In Chapters 3 and 4, this structure (via factor graphs) will serve a central role in integrating learning in to this framework.

We conduct benchmarks and compare GPMP and GPMP2 against leading trajectory optimization-based motion planning algorithms [44, 45] as well as sampling-based motion planning algorithms [5, 46, 47] in multiple reaching tasks (Figure 2.1). Our results show

GPMP2 to be several times faster than the state-of-the-art with higher success rates. We also benchmark GPMP2 against our incremental planner, iGPMP2, on replanning tasks and show that iGPMP2 can incrementally solve replanning problems an order of magnitude faster than GPMP2 solving from scratch.

2.2 Related work

Most motion planning algorithms are broadly classified into sampling-based algorithms or trajectory optimization-based algorithms. Sampling-based planners such as probabilistic roadmaps (PRMs) [4] construct a dense graph from random samples in obstacle free areas of the robot's configuration space. PRMs can be used for multiple queries by finding the shortest path between a start and goal configuration in the graph. Rapidly exploring random trees (RRTs) [5, 6] find trajectories by incrementally building space-filling trees through directed sampling. RRTs are very good at finding feasible solutions in highly constrained problems and high-dimensional search spaces. Both PRMs and RRTs offer probabilistic completeness, ensuring that, given enough time, a feasible trajectory can be found, if one exists. However, they often result in jerky and redundant motion and therefore require post processing to address optimality. Although optimal planners [48] have been proposed, they are computationally inefficient on high-dimensional problems with challenging constraints. Despite guarantees, sampling-based algorithms may be difficult to use in real-time applications due to computational challenges. Often computation is wasted exploring regions that may not lead to a solution. Recent work in informed techniques [49] combat this problem by biasing the sampling approach to make search more tractable.

In contrast with sampling-based planners, trajectory optimization starts with an initial, possibly infeasible, trajectory and then optimizes the trajectory by minimizing a cost function. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) and related methods [9, 44, 50, 51, 52] optimize a cost functional using covariant gradient descent, while Stochastic Trajectory Optimization for Motion Planning (STOMP) [53] opti-

mizes non-differentiable costs by stochastic sampling of noisy trajectories. TrajOpt [11, 45] solves a sequential quadratic program and performs convex continuous-time collision checking. It achieves reduced computational costs by parameterizing the trajectory with a small number of states and employing continuous-time collision checking. However, due to the discrete-time representation of the trajectory, a sparse solution may need post-processing for execution and may not remain collision-free. In other words, a fine discretization may still be necessary on problems in complex environments. In contrast to sampling-based planners, trajectory optimization methods are very fast, but only find locally optimal solution. The computational bottleneck results from evaluating costs on a fine discretization of the trajectory or, in difficult problems, repeatedly changing the initial conditions until a feasible trajectory is discovered.

Continuous-time trajectory representations can overcome the computational cost incurred by finely discretizing the trajectory. Linear interpolation [54, 55, 56], splines [57, 58, 59, 60, 61, 62], and hierarchical wavelets [63] have been used to represent trajectories in filtering and state estimation. B-Splines [64] have similarly been used to represent trajectories in motion planning problems. Compared to parametric representations like splines and wavelets, Gaussian processes provide a natural notion of uncertainty on top of allowing a sparse parameterization of the continuous-time trajectory. A critical distinction in motion planning problems is that even with a sparse parameterization, the collision cost has to be evaluated at a finer resolution. Therefore, if the interpolation procedure for a chosen continuous-time representation is computationally expensive, the resulting speedup obtained from a sparse representation is negligible and may result in an overall slower algorithm. Recent work by [52] works to optimize trajectories in RKHS with RBF kernels, but ignores the cost between sparse waypoints. Even without interpolation, these dense kernels result in relatively computationally expensive updates. In this work, we use structured Gaussian processes (GPs) that allow us to exploit the underlying sparsity in the problem to perform efficient inference. We are able to use fast GP regression to interpolate

the trajectory and evaluate obstacle cost on a finer resolution, while the trajectory can be parameterized by a small number of support states. We also show in this work that the probabilistic representation naturally allows us to represent the motion planning problem with a factor graph and the GP directly corresponds to the system dynamics or motion model thus giving it a physical meaning.

GPs have been used for function approximation in supervised learning [65, 66], inverse dynamics modeling [67, 68], reinforcement learning [69], path prediction [70], simultaneous localization and mapping [71, 72], state estimation [73, 74], and controls [75], but to our knowledge GPs have not been used in motion planning.

We also consider motion planning from the perspective of probabilistic inference. Early work by [13] uses inference to solve Markov decision processes. More recently, solutions to planning and control problems have used probabilistic tools such as expectation propagation [15], expectation maximization [14, 76], and KL-minimization [77]. We exploit the duality between inference and optimization to perform inference on factor graphs by solving nonlinear least square problems. While this is an established and efficient approach [39] to solving large scale SLAM problems, we introduce this technique in the context of motion planning. Incremental inference can also be performed efficiently on factor graphs [42, 43], a fact we take advantage of to solve replanning problems.

Replanning involves adapting a previously solved solution to changing conditions. Early replanning work like D* [78] and Anytime A* [2] need a finely discretized state space and therefore do not scale well with high-dimensional problems. Recent trajectory optimization algorithms inspired from CHOMP [9] like incremental trajectory optimization for motion planning (ITOMP) [79] can fluently replan using a scheduler that enforces timing restrictions but the solution cannot guarantee feasibility. GPUs have been suggested as a way to increase the speed of replanning [80], with some success. Our algorithm is inspired from the incremental approach to SLAM problems [43] that can efficiently update factor graphs to generate new solutions without performing redundant calculations. During planning, we

use this method to update the trajectory only where necessary, thus reducing computational costs and making fast replanning possible.

2.3 Motion planning as trajectory optimization

The goal of motion planning via trajectory optimization is to find trajectories, $\boldsymbol{\theta}(t) : t \rightarrow \mathbb{R}^D$, where D is dimensionality of the state, that satisfy constraints and minimize cost [44, 53, 45]. Motion planning can therefore be formalized as

$$\begin{aligned} \text{minimize} \quad & \mathcal{F}[\boldsymbol{\theta}(t)] \\ \text{subject to} \quad & \mathcal{G}_i[\boldsymbol{\theta}(t)] \leq 0,\ i = 1, \ldots, m_{ineq} \\ & \mathcal{H}_i[\boldsymbol{\theta}(t)] = 0,\ i = 1, \ldots, m_{eq} \end{aligned} \tag{2.1}$$

where the trajectory $\boldsymbol{\theta}(t)$ is a continuous-time function, mapping time t to robot states, which are generally configurations (and possibly higher-order derivatives). $\mathcal{F}[\boldsymbol{\theta}(t)]$ is an objective or cost functional that evaluates the quality of a trajectory and usually encodes *smoothness* that minimizes higher-order derivatives of the robot states (for example, velocity or acceleration) and collision costs that enforces the trajectory to be *collision-free*. $\mathcal{G}_i[\boldsymbol{\theta}(t)]$ are inequality constraint functionals such as joint angle limits, and $\mathcal{H}_i[\boldsymbol{\theta}(t)]$ are task-dependent equality constraints, such as the desired start and end configurations and velocities, or the desired end-effector orientation (for example, holding a cup filled with water upright). The number of inequality or equality constraints may be zero, depending on the specific problem. Based on the optimization technique used to solve Eq. (2.1), collision cost may also appear as an obstacle avoidance inequality constraint [45]. In practice, most existing trajectory optimization algorithms work with a fine discretization of the trajectory, which can be used to reason about thin obstacles or tight navigation constraints, but can incur a large computational cost.

2.4 Gaussian processes for continuous-time trajectories

A vector-valued Gaussian process (GP) [35] provides a principled way to reason about continuous-time trajectories, where the trajectories are viewed as functions that map time to state. In this section, we describe how GPs can be used to encode a prior on trajectories such that optimality properties like smoothness are naturally encouraged (Section 2.4.1). We also consider a class of structured priors for trajectories that will be useful in efficient optimization (Section 2.4.2), and we provide details about how fast GP interpolation can be used to query the trajectory at any time of interest (Section 2.4.3).

2.4.1 The GP prior

We consider continuous-time trajectories as samples from a vector-valued GP, $\boldsymbol{\theta}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t, t'))$, where $\boldsymbol{\mu}(t)$ is a vector-valued mean function and $\boldsymbol{\mathcal{K}}(t, t')$ is a matrix-valued covariance function. A vector-valued GP is a collection of random variables, any finite number of which have a joint Gaussian distribution. Using the GP framework, we can say that for any collection of times $\boldsymbol{t} = \{t_0, \ldots, t_N\}$, $\boldsymbol{\theta}$ has a joint Gaussian distribution:

$$\boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{\theta}_0 & \ldots & \boldsymbol{\theta}_N \end{bmatrix}^\top \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\mathcal{K}}) \tag{2.2}$$

with the mean vector $\boldsymbol{\mu}$ and covariance kernel $\boldsymbol{\mathcal{K}}$ defined as

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_0 & \ldots & \boldsymbol{\mu}_N \end{bmatrix}^\top, \; \boldsymbol{\mathcal{K}} = [\boldsymbol{\mathcal{K}}(t_i, t_j)]\Big|_{ij, 0 \le i,j \le N}. \tag{2.3}$$

We use bold $\boldsymbol{\theta}$ to denote the matrix formed by vectors $\boldsymbol{\theta}_i \in \mathbb{R}^D$, which are *support states* that parameterize the continuous-time trajectory $\boldsymbol{\theta}(t)$. Similar notation is used for $\boldsymbol{\mu}$.

The GP defines a prior on the space of trajectories:

$$p(\boldsymbol{\theta}) \propto \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\boldsymbol{\mathcal{K}}} \right\} \tag{2.4}$$

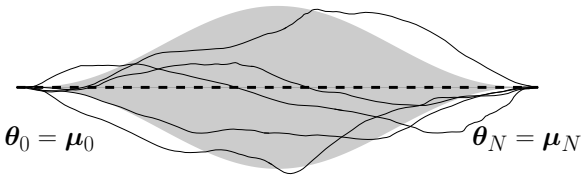


Figure 2.2: An example GP prior for trajectories. The dashed line is the mean trajectory $\boldsymbol{\mu}(t)$ and the shaded area indicates the covariance. The 5 solid lines are sample trajectories $\boldsymbol{\theta}(t)$ from the GP prior.

where $\| \boldsymbol{\theta} - \boldsymbol{\mu} \|^2_{\boldsymbol{\mathcal{K}}} \doteq (\boldsymbol{\theta} - \boldsymbol{\mu})^\top \boldsymbol{\mathcal{K}}^{-1} (\boldsymbol{\theta} - \boldsymbol{\mu})$ is the Mahalanobis distance. Figure 2.2 shows an example GP prior for trajectories. Intuitively this prior encourages *smoothness* encoded by the kernel $\boldsymbol{\mathcal{K}}$ and directly applies on the function space of trajectories. The negative log of this distribution serves as the prior cost functional in the objective (see Section 2.5.1) and penalizes the deviation of the trajectory from the mean defined by the prior.

2.4.2 A Gauss-Markov model

Similar to previous work [81, 71], we use a structured kernel generated by a linear time-varying stochastic differential equation (LTV-SDE)

$$\dot{\boldsymbol{\theta}}(t) = \mathbf{A}(t)\boldsymbol{\theta}(t) + \mathbf{u}(t) + \mathbf{F}(t)\mathbf{w}(t), \tag{2.5}$$

where $\mathbf{u}(t)$ is the known system control input, $\mathbf{A}(t)$ and $\mathbf{F}(t)$ are time-varying matrices of the system, and $\mathbf{w}(t)$ is generated by a white noise process. The white noise process is itself a zero-mean GP

$$\mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_C \delta(t - t')). \tag{2.6}$$

$\mathbf{Q}_C$ is the power-spectral density matrix and $\delta(t - t')$ is the Dirac delta function. The solution to the initial value problem of this LTV-SDE is

$$\boldsymbol{\theta}(t) = \boldsymbol{\Phi}(t, t_0)\boldsymbol{\theta}_0 + \int_{t_0}^{t} \boldsymbol{\Phi}(t, s)(\mathbf{u}(s) + \mathbf{F}(s)\mathbf{w}(s))\, \mathrm{d}s, \tag{2.7}$$

where $\mathbf{\Phi}(t, s)$ is the state transition matrix, which transfers state from time s to time t. The mean and covariance functions of the GP defined by this LTV-SDE are calculated by taking the first and second moments respectively on Eq. (2.7),

$$\widetilde{\boldsymbol{\mu}}(t) = \mathbf{\Phi}(t, t_0)\boldsymbol{\mu}_0 + \int_{t_0}^{t} \mathbf{\Phi}(t, s)\mathbf{u}(s)\,\mathrm{d}s, \tag{2.8}$$

$$\begin{aligned}\widetilde{\boldsymbol{\mathcal{K}}}(t, t') &= \mathbf{\Phi}(t, t_0)\boldsymbol{\mathcal{K}}_0\mathbf{\Phi}(t', t_0)^\top \\ &\quad + \int_{t_0}^{\min(t,t')} \mathbf{\Phi}(t, s)\mathbf{F}(s)\mathbf{Q}_C\mathbf{F}(s)^\top\mathbf{\Phi}(t', s)^\top\,\mathrm{d}s.\end{aligned} \tag{2.9}$$

$\boldsymbol{\mu}_0$ and $\boldsymbol{\mathcal{K}}_0$ are the initial mean and covariance of the start state respectively.

The desired prior of trajectories between a given start state $\boldsymbol{\theta}_0$ and goal state $\boldsymbol{\theta}_N$ for a finite set of support states, as described in Section 2.4.1, can be found by conditioning this GP with a fictitious observation on the goal state with mean $\boldsymbol{\mu}_N$ and covariance $\boldsymbol{\mathcal{K}}_N$. Specifically

$$\boldsymbol{\mu} = \widetilde{\boldsymbol{\mu}} + \widetilde{\boldsymbol{\mathcal{K}}}(t_N, \boldsymbol{t})^\top(\widetilde{\boldsymbol{\mathcal{K}}}(t_N, t_N) + \boldsymbol{\mathcal{K}}_N)^{-1}(\boldsymbol{\theta}_N - \boldsymbol{\mu}_N) \tag{2.10}$$

$$\boldsymbol{\mathcal{K}} = \widetilde{\boldsymbol{\mathcal{K}}} - \widetilde{\boldsymbol{\mathcal{K}}}(t_N, \boldsymbol{t})^\top(\widetilde{\boldsymbol{\mathcal{K}}}(t_N, t_N) + \boldsymbol{\mathcal{K}}_N)^{-1}\widetilde{\boldsymbol{\mathcal{K}}}(t_N, \boldsymbol{t}), \tag{2.11}$$

where $\widetilde{\boldsymbol{\mathcal{K}}}(t_N, \boldsymbol{t}) = [\widetilde{\boldsymbol{\mathcal{K}}}(t_N, t_0) \quad \dots \quad \widetilde{\boldsymbol{\mathcal{K}}}(t_N, t_N)]$ (please see Appendix A.1 for proof).

This particular construction of the prior leads to a Gauss-Markov model that generates a GP with an exactly sparse tridiagonal precision matrix (inverse kernel) that can be factored as:

$$\boldsymbol{\mathcal{K}}^{-1} = \mathbf{B}^\top\mathbf{Q}^{-1}\mathbf{B} \tag{2.12}$$

with,

$$\mathbf{B} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ -\mathbf{\Phi}(t_1, t_0) & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\mathbf{\Phi}(t_2, t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & -\mathbf{\Phi}(t_N, t_{N-1}) & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{2.13}$$

which has a band diagonal structure and $\mathbf{Q}^{-1}$ is block diagonal such that

$$\mathbf{Q}^{-1} = \operatorname{diag}(\boldsymbol{\mathcal{K}}_0^{-1}, \mathbf{Q}_{0,1}^{-1}, \dots, \mathbf{Q}_{N-1,N}^{-1}, \boldsymbol{\mathcal{K}}_N^{-1}), \tag{2.14}$$

$$\mathbf{Q}_{a,b} = \int_{t_a}^{t_b} \mathbf{\Phi}(b, s)\mathbf{F}(s)\mathbf{Q}_c\mathbf{F}(s)^\top\mathbf{\Phi}(b, s)^\top \, \mathrm{d}s \tag{2.15}$$

(see Appendix A for proof). This sparse structure is useful for fast GP interpolation (Section 2.4.3) and efficient optimization (Section 2.5 and 2.6).

An interesting observation here is that this choice of kernel can be viewed as a generalization of CHOMP [44]. For instance, if the identity and zero blocks in the precision matrix are scalars, the state transition matrix $\mathbf{\Phi}$ is a unit scalar, and $\mathbf{Q}^{-1}$ is an identity matrix, $\boldsymbol{\mathcal{K}}^{-1}$ reduces to the matrix A formed by finite differencing in CHOMP. In this context, it means that CHOMP considers a trajectory of positions in configuration space, that is generated by a deterministic differential equation (since $\mathbf{Q}^{-1}$ is identity).

The linear model in Eq. (2.5) is sufficient to model kinematics for the robot manipulators considered in the scope of this work, however our framework can be extended to consider non-linear models following [82].

2.4.3 Gaussian process interpolation

One of the benefits of Gaussian processes is that they can be parameterized by only a sparse set of *support states*, but the trajectory can be queried at *any* time of interest through Gaussian process interpolation. The reduced parameterization makes each iteration of trajectory

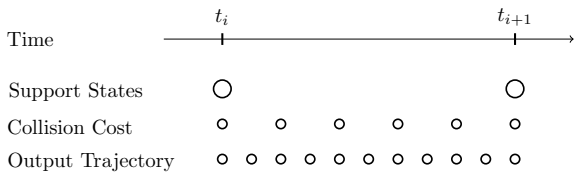


Figure 2.3: An example that shows the trajectory at different resolutions. Support states parameterize the trajectory, collision cost checking is performed at a higher resolution during optimization and the output trajectory can be up-sampled further for execution.

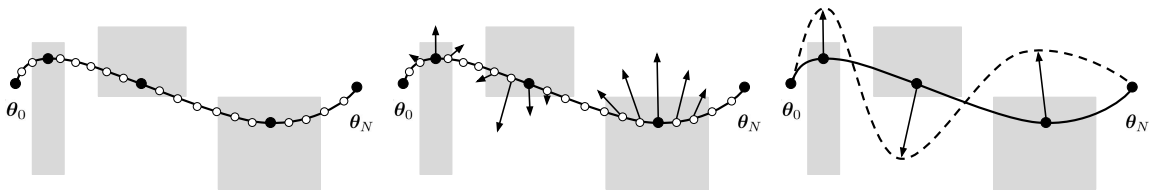
optimization efficient. Given the choice of the structured prior from the previous subsection, rich collision costs between the support states can be evaluated by performing dense GP interpolation between the support states quickly and efficiently. This cost can then be used to update the support states in a meaningful manner, reducing the computational effort. A much denser resolution of interpolation (Figure 2.3) can also be useful in practice to feed the trajectory to a controller on a real robot.

The process of updating a trajectory with GP interpolation is explained through an example illustrated in Figure 2.4. At each iteration of optimization, the trajectory with a sparse set of support states can be densely interpolated with a large number of states, and the collision cost can be evaluated on all the states (both support and interpolated). Next, collision costs at the interpolated states are propagated and accumulated to the nearby support states (the exact process to do this is explained in Section 2.5.3 and 2.7). Finally, the trajectory is updated by only updating the support states given the accumulated cost information.

Following [81, 71, 72], we show how to exploit the structured prior to perform fast GP interpolation. The posterior mean of the trajectory at any time τ can be found in terms of the current trajectory $\boldsymbol{\theta}$ at time points $\boldsymbol{t}$ [35] by conditioning on the support states that parameterize trajectory:

$$\boldsymbol{\theta}(\tau) = \widetilde{\boldsymbol{\mu}}(\tau) + \widetilde{\boldsymbol{\mathcal{K}}}(\tau, \boldsymbol{t})\widetilde{\boldsymbol{\mathcal{K}}}^{-1}(\boldsymbol{\theta} - \widetilde{\boldsymbol{\mu}}) \tag{2.16}$$

i.e. performing Gaussian process regression. Although the interpolation in Eq. (2.16)



(a) Interpolated trajectory (b) Gradients on all states (c) Gradients on support states

Figure 2.4: An example showing how GP interpolation is used during optimization. (a) shows the current iteration of the trajectory (black curve) parameterized by a sparse set of support states (black circles). GP regression is used to densely up-sample the trajectory with interpolated states (white circles). Then, in (b) cost is evaluated on all states and their gradients are illustrated by the arrows. Finally, in (c) the cost and gradient information is propagated to just the support states illustrated by the larger arrows such that only the support states are updated that parameterize the new trajectory (dotted black curve).

naïvely requires $O(N)$ operations, $\boldsymbol{\theta}(\tau)$ can be computed in $O(1)$ by leveraging the structure of the sparse GP prior generated by the Gauss-Markov model introduced in Section 2.4. This implies that $\boldsymbol{\theta}(\tau)$ at $\tau, t_i < \tau < t_{i+1}$ can be expressed as a linear combination of *only* the adjacent function values $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{i+1}$ and is efficiently computed as

$$\boldsymbol{\theta}(\tau) = \widetilde{\boldsymbol{\mu}}(\tau) + \boldsymbol{\Lambda}(\tau)(\boldsymbol{\theta}_i - \widetilde{\boldsymbol{\mu}}_i) + \boldsymbol{\Psi}(\tau)(\boldsymbol{\theta}_{i+1} - \widetilde{\boldsymbol{\mu}}_{i+1}) \tag{2.17}$$

where

$$\begin{aligned}\boldsymbol{\Lambda}(\tau) &= \boldsymbol{\Phi}(\tau, t_i) - \boldsymbol{\Psi}(\tau)\boldsymbol{\Phi}(t_{i+1}, t_i) \\ \boldsymbol{\Psi}(\tau) &= \mathbf{Q}_{i,\tau}\boldsymbol{\Phi}(t_{i+1}, \tau)^\top \mathbf{Q}_{i,i+1}^{-1}\end{aligned}$$

is derived by substituting

$$\widetilde{\boldsymbol{\mathcal{K}}}(\tau)\widetilde{\boldsymbol{\mathcal{K}}}^{-1} = [\, \mathbf{0} \;\ldots\; \mathbf{0}\; \boldsymbol{\Lambda}(\tau)\; \boldsymbol{\Psi}(\tau)\; \mathbf{0} \;\ldots\; \mathbf{0}\,]$$

in Eq. (2.16) with only the $(i)^{th}$ and $(i+1)^{th}$ block columns being non-zero.

This provides an elegant way to do fast GP interpolation on the trajectory that exploits the structure of the problem. In Section 2.5.3 and 2.7 we show how this is utilized to

perform efficient optimization.

2.5 Motion planning with Gaussian processes

We now describe the Gaussian Process Motion Planner (**GPMP**), which combines the Gaussian process representation with a gradient descent-based optimization algorithm for motion planning.

2.5.1 Cost functionals

Following the problem definition in Eq. (2.1) we design the objective functional as

$$\mathcal{F}[\boldsymbol{\theta}(t)] = \mathcal{F}_{obs}[\boldsymbol{\theta}(t)] + \lambda \mathcal{F}_{gp}[\boldsymbol{\theta}(t)] \tag{2.18}$$

where $\mathcal{F}_{gp}$ is the GP prior cost functional (the negative natural logarithm of prior distribution) from Eq. (2.4)

$$\mathcal{F}_{gp}[\boldsymbol{\theta}(t)] = \mathcal{F}_{gp}[\boldsymbol{\theta}] = \frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}\|^2_{\boldsymbol{\mathcal{K}}} \tag{2.19}$$

penalizing the deviation of the parameterized trajectory from the prior mean, $\mathcal{F}_{obs}$ is the obstacle cost functional that penalizes collision with obstacles and λ is the trade-off between the two functionals.

As discussed in Section 2.4.2 the GP smoothness prior can be considered a generalization to the one used in practical applications of CHOMP constructed through finite dynamics. In contrast to CHOMP, we also consider our trajectory to be augmented by velocities and acceleration. This allows us to keep the state Markovian in the prior model (Section 2.4.2), is useful in computation of the obstacle cost gradient (Section 2.5.2), and also allows us to stretch or squeeze the trajectory in space while keeping the states on the trajectory temporally equidistant [51].

The obstacle cost functional $\mathcal{F}_{obs}$ is also similar to the one used in CHOMP [44]. This functional computes the arc-length parameterized line integral of the workspace obstacle

cost of each body point as it passes through the workspace, and integrates over all body points:

$$\mathcal{F}_{obs}[\boldsymbol{\theta}(t)] = \int_{t_0}^{t_N} \int_{\mathcal{B}} c(x)\|\dot{x}\| \, \mathrm{d}u \, \mathrm{d}t \tag{2.20}$$

where $c(\cdot) : \mathbb{R}^3 \to \mathbb{R}$ is the workspace cost function that penalizes the set of points $\mathcal{B} \subset \mathbb{R}^3$ on the robot body when they are in or around an obstacle, and x is the forward kinematics that maps robot configuration to workspace (see [44] for details).

In practice, the cost functional can be approximately evaluated on the discrete support state parameterization of the trajectory i.e. $\mathcal{F}_{obs}[\boldsymbol{\theta}(t)] = \mathcal{F}_{obs}[\boldsymbol{\theta}]$, the obstacle cost is calculated using a precomputed signed distance field (see Section 2.9.1), and the inner integral is replaced with a summation over a finite number of body points that well approximate the robot's physical body.

2.5.2 Optimization

We adopt an iterative, gradient-based approach to minimize the non-convex objective functional in Eq. (2.18). In each iteration, we form an approximation to the cost functional via a Taylor series expansion around the current parameterized trajectory $\boldsymbol{\theta}$:

$$\mathcal{F}[\boldsymbol{\theta} + \delta\boldsymbol{\theta}] \approx \mathcal{F}[\boldsymbol{\theta}] + \bar{\nabla}\mathcal{F}[\boldsymbol{\theta}]\delta\boldsymbol{\theta} \tag{2.21}$$

We next minimize the approximate cost while constraining the trajectory to be close to the previous one. Then the optimal perturbation $\delta\boldsymbol{\theta}^*$ to the trajectory is:

$$\delta\boldsymbol{\theta}^* = \underset{\delta\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \mathcal{F}[\boldsymbol{\theta}] + \bar{\nabla}\mathcal{F}[\boldsymbol{\theta}]\delta\boldsymbol{\theta} + \frac{\eta}{2}\|\delta\boldsymbol{\theta}\|^2_{\boldsymbol{\mathcal{K}}} \right\} \tag{2.22}$$

where η is the regularization constant. Differentiating the right-hand side and setting the result to zero we obtain the update rule for each iteration:

$$\bar{\nabla}\mathcal{F}[\boldsymbol{\theta}] + \eta\boldsymbol{\mathcal{K}}^{-1}\delta\boldsymbol{\theta}^* = 0 \quad \Longrightarrow \quad \delta\boldsymbol{\theta}^* = -\frac{1}{\eta}\boldsymbol{\mathcal{K}}\bar{\nabla}\mathcal{F}[\boldsymbol{\theta}]$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}^* = \boldsymbol{\theta} - \frac{1}{\eta}\boldsymbol{\mathcal{K}}\bar{\nabla}\mathcal{F}[\boldsymbol{\theta}] \tag{2.23}$$

To compute the update rule we need to find the gradient of the cost functional at the current trajectory

$$\bar{\nabla}\mathcal{F}[\boldsymbol{\theta}] = \bar{\nabla}\mathcal{F}_{obs}[\boldsymbol{\theta}] + \lambda\bar{\nabla}\mathcal{F}_{gp}[\boldsymbol{\theta}], \tag{2.24}$$

which requires computing the gradients of the GP and obstacle cost functional. The gradient of the GP prior cost can be computed by taking the derivative of Eq. (2.19) with respect to the current trajectory

$$\mathcal{F}_{gp}[\boldsymbol{\theta}] = \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu})^\top\boldsymbol{\mathcal{K}}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu})$$
$$\bar{\nabla}\mathcal{F}_{gp}[\boldsymbol{\theta}] = \boldsymbol{\mathcal{K}}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}) \tag{2.25}$$

The gradient of the obstacle cost functional can be computed from the Euler-Lagrange equation [83] in which a functional of the form $\mathcal{F}[\boldsymbol{\theta}(t)] = \int v(\boldsymbol{\theta}(t))\,\mathrm{d}t$ yields a gradient

$$\bar{\nabla}\mathcal{F}[\boldsymbol{\theta}(t)] = \frac{\partial v}{\partial\boldsymbol{\theta}(t)} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial v}{\partial\dot{\boldsymbol{\theta}}(t)} \tag{2.26}$$

Applying Eq. (2.26) to find the gradient of Eq. (2.20) in the workspace and then mapping it back to the configuration space via the kinematic Jacobian J, and following the proof by [84], we compute the gradient with respect to configuration position, velocity, and acceler-

ation at any time point t_i as

$$\bar{\nabla}\mathcal{F}_{obs}[\boldsymbol{\theta}_i] = \begin{bmatrix} \int_{\mathcal{B}} J^\top ||\dot{x}|| \left[(I - \hat{\dot{x}}\hat{\dot{x}}^\top)\nabla c - c\kappa \right] \mathrm{d}u \\ \int_{\mathcal{B}} J^\top c\, \hat{\dot{x}}\, \mathrm{d}u \\ 0 \end{bmatrix} \tag{2.27}$$

where $\kappa = ||\dot{x}||^{-2}(I - \hat{\dot{x}}\hat{\dot{x}}^\top)\ddot{x}$ is the curvature vector along the workspace trajectory traced by a body point, $\dot{x}$, $\ddot{x}$ are the velocity and acceleration respectively, of that body point determined by forward kinematics and the Hessian, and $\hat{\dot{x}} = \dot{x}/||\dot{x}||$ is the normalized velocity vector. Due to the augmented state, the velocity and acceleration can be obtained through the Jacobian and Hessian directly from the state. This is in contrast to CHOMP, which approximates the velocity and acceleration through finite differencing. The gradients at each time point are stacked together into a single vector $\mathrm{g} = \bar{\nabla}\mathcal{F}_{obs}[\boldsymbol{\theta}]$. We plug the cost gradients back into the update rule in Eq. (2.23) to get the update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{1}{\eta}\boldsymbol{\mathcal{K}}\left(\lambda \boldsymbol{\mathcal{K}}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}) + \mathrm{g} \right) \tag{2.28}$$

This update rule can be interpreted as a generalization of the update rule for CHOMP with an augmented trajectory and a generalized prior.

2.5.3 Compact trajectory representations and faster updates

In this section, we show that the finite number of states used to parameterize smooth trajectories can be very sparse in practice. Through GP interpolation, we can up-sample the trajectory to any desired resolution, calculate costs and gradients at this resolution, and then project the gradients back to just the sparse set of support states. To interpolate n_{ip} states between two support states at t_i and t_{i+1}, we define two aggregated matrices using

Eq. (2.17),

$$\boldsymbol{\Lambda}_i = \begin{bmatrix} \boldsymbol{\Lambda}_{i,1}^\top & \dots & \boldsymbol{\Lambda}_{i,j}^\top & \dots & \boldsymbol{\Lambda}_{i,n_{ip}}^\top \end{bmatrix}^\top$$
$$\boldsymbol{\Psi}_i = \begin{bmatrix} \boldsymbol{\Psi}_{i,1}^\top & \dots & \boldsymbol{\Psi}_{i,j}^\top & \dots & \boldsymbol{\Psi}_{i,n_{ip}}^\top \end{bmatrix}^\top$$

If we want to up-sample a sparse trajectory $\boldsymbol{\theta}$ by interpolating n_{ip} states between every support state, we can quickly compute the new trajectory $\boldsymbol{\theta}_{up}$ as

$$\boldsymbol{\theta}_{up} = \mathbf{M}(\boldsymbol{\theta} - \boldsymbol{\mu}) + \boldsymbol{\mu}_{up} \tag{2.29}$$

where $\boldsymbol{\mu}_{up}$ corresponds to the prior mean with respect to the up sampled trajectory, and

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \dots & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Lambda}_0 & \boldsymbol{\Psi}_0 & \mathbf{0} & \dots & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \dots & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Lambda}_1 & \boldsymbol{\Psi}_1 & \dots & \dots & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & & & & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \boldsymbol{\Lambda}_i & \boldsymbol{\Psi}_i & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & & & & \ddots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \dots & \dots & \dots & \boldsymbol{\Lambda}_{N-1} & \boldsymbol{\Psi}_{N-1} \\ \mathbf{0} & \mathbf{0} & \dots & \dots & \dots & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{2.30}$$

is a tall matrix that up-samples a sparse trajectory $\boldsymbol{\theta}$ with only $N + 1$ support states to trajectory $\boldsymbol{\theta}_{up}$ with $(N+1)+N\times n_{ip}$ states. The fast, high-temporal-resolution interpolation is also useful in practice if we want to feed the planned trajectory into a controller.

The efficient update rule is defined analogous to Eq. (2.28) except on a sparse parametrization of the trajectory

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{1}{\eta}\boldsymbol{\mathcal{K}}\Big(\lambda\boldsymbol{\mathcal{K}}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}) + \mathbf{M}^\top \mathbf{g}_{up}\Big) \tag{2.31}$$

where the obstacle gradient over the sparse trajectory is found by chain rule using Eq. (2.29) and the obstacle gradient, $\mathbf{g}_{up}$ over the up-sampled trajectory. In other words, the above equation calculates the obstacle gradient for *all* states (interpolated and support) and then projects them back onto just the support states using $\mathbf{M}^\top$. Cost information between support states is still utilized to perform the optimization, however only a sparse parameterization is necessary making the remainder of the update more efficient.

GPMP demonstrates how a continuous-time representation of the trajectory using GPs can generalize CHOMP and improve performance through sparse parameterization. However, the gradient-based optimization scheme has two drawbacks: first, convergence is slow due to the large number of iterations required to get a feasible solution; and, second, the gradients can be costly to calculate (See Figure 2.12). We improve upon GPMP and address these concerns in the next section.

2.6 Motion planning as probabilistic inference

To fully evoke the power of GPs, we view motion planning as probabilistic inference. A similar view has been explored before by Toussaint et al. [15, 37]. Unlike this previous work, which uses message passing to perform inference, we exploit the duality between inference and optimization and borrow ideas from the SLAM community for a more efficient approach. In particular, we use tools from the Smoothing and Mapping (SAM) framework [39] that performs inference on factor graphs by solving a nonlinear least squares problem [38]. This approach exploits the sparsity of the underlying problem to obtain quadratic convergence.

The probabilistic inference view of motion planning provides several advantages:

1. The duality between inference and least squares optimization allows us to perform inference very efficiently, so motion planning is extremely *fast*.

2. Inference tools from other areas of robotics, like the incremental algorithms based

on the Bayes tree data structure [43], can be exploited and used in the context of planning. These tools can help speed up *replanning*.

3. Inference can provide a deeper understanding of the connections between different areas of robotics, such as planning and control [85], estimation and planning [86, 87], and learning from demonstration and planning [88, 89].

In this and the next section, we develop the **GPMP2** algorithm, which is more efficient compared to GPMP.

To formulate this problem as inference, we seek to find a trajectory parameterized by $\boldsymbol{\theta}$ given desired events $\mathbf{e}$. For example, binary events e_i at t_i might signify that the trajectory is collision-free if all $e_i = 0$ (i.e. $\mathbf{e} = \mathbf{0}$) and in collision if any $e_i = 1$. In general, the motion planning problem can be formulated with any set of desired events.

The posterior density of $\boldsymbol{\theta}$ given $\mathbf{e}$ can be computed by Bayes rule from a prior and likelihood

$$p(\boldsymbol{\theta}|\mathbf{e}) = p(\boldsymbol{\theta})p(\mathbf{e}|\boldsymbol{\theta})/p(\mathbf{e}) \tag{2.32}$$

$$\propto p(\boldsymbol{\theta})p(\mathbf{e}|\boldsymbol{\theta}), \tag{2.33}$$

where $p(\boldsymbol{\theta})$ is the prior on $\boldsymbol{\theta}$ that encourages smooth trajectories, and $p(\mathbf{e}|\boldsymbol{\theta})$ is the likelihood that the trajectory $\boldsymbol{\theta}$ is collision free. The optimal trajectory $\boldsymbol{\theta}$ is found by the *maximum a posteriori* (MAP) estimator, which chooses the trajectory that maximizes the posterior $p(\boldsymbol{\theta}|\mathbf{e})$

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}|\mathbf{e}) \tag{2.34}$$

$$= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta})l(\boldsymbol{\theta};\mathbf{e}), \tag{2.35}$$

where $l(\boldsymbol{\theta}; \mathbf{e})$ is the likelihood of states $\boldsymbol{\theta}$ given events $\mathbf{e}$ on the whole trajectory

$$l(\boldsymbol{\theta}; \mathbf{e}) \propto p(\mathbf{e}|\boldsymbol{\theta}). \tag{2.36}$$

We use the same GP prior as in Section 2.4

$$p(\boldsymbol{\theta}) \propto \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\boldsymbol{\mathcal{K}}} \right\}. \tag{2.37}$$

The likelihood is defined as a distribution in the exponential family

$$l(\boldsymbol{\theta}; \mathbf{e}) = \exp\left\{ -\frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e}) \parallel^2_{\boldsymbol{\Sigma}} \right\} \tag{2.38}$$

where $\boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e})$ is a vector-valued cost function for the trajectory, and $\boldsymbol{\Sigma}$ is a diagonal matrix and the hyperparameter of the distribution.

2.7 Structure with factor graphs

Given the Markovian structure of the trajectory and sparsity of inverse kernel matrix, the posterior distribution can be further factored such that MAP inference can be equivalently viewed as performing inference on a *factor graph* [38].

A factor graph $G = \{\boldsymbol{\Theta}, \mathcal{F}, \mathcal{E}\}$ is a bipartite graph, which represents a factored function, where $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_N\}$ are a set of variable nodes, $\mathcal{F} \doteq \{f_0, \ldots, f_M\}$ are a set of factor nodes, and $\mathcal{E}$ are edges connecting the two type of nodes.

In our problems, the factorization of the posterior distribution can be written as

$$p(\boldsymbol{\theta}|\mathbf{e}) \propto \prod_{m=1}^{M} f_m(\boldsymbol{\Theta}_m), \tag{2.39}$$

where f_m are factors on variable subsets $\boldsymbol{\Theta}_m$.

Given the tridiagonal inverse kernel matrix defined by Eq. (2.12)-(2.14), we factor the

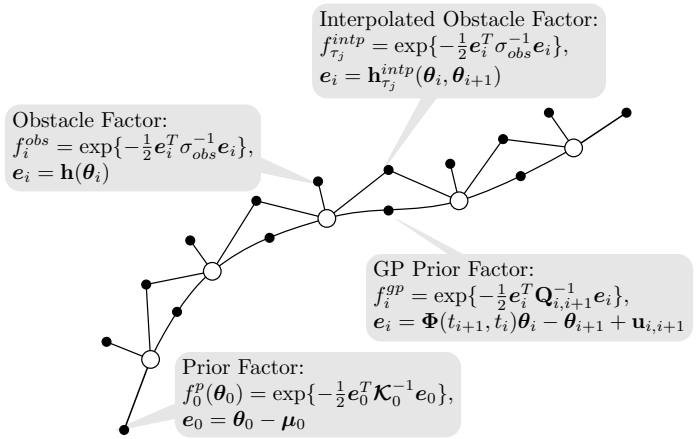


Figure 2.5: A factor graph of an example trajectory optimization problem showing support states (white circles) and four kinds of factors (black dots), namely prior factors on start and goal states, GP prior factors that connect consecutive support states, obstacle factors on each state, and interpolated obstacle factors between consecutive support states (only one shown here for clarity, any number of them may be present in practice).

prior

$$p(\boldsymbol{\theta}) \propto f^p_0(\boldsymbol{\theta}_0) f^p_N(\boldsymbol{\theta}_N) \prod_{i=0}^{N-1} f^{gp}_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}), \tag{2.40}$$

where $f^p_0(\boldsymbol{\theta}_0)$ and $f^p_N(\boldsymbol{\theta}_N)$ define the prior distributions on start and goal states respectively

$$f^p_i(\boldsymbol{\theta}_i) = \exp\left\{ -\frac{1}{2}\|\boldsymbol{\theta}_i - \boldsymbol{\mu}_i\|^2_{\boldsymbol{\mathcal{K}}_i} \right\}, i = 0 \text{ or } N \tag{2.41}$$

where $\boldsymbol{\mathcal{K}}_0$ and $\boldsymbol{\mathcal{K}}_N$ are covariance matrices on start and goal states respectively, and $\boldsymbol{\mu}_0$ and $\boldsymbol{\mu}_N$ are prior (known) start and goal states respectively. The GP prior factor is

$$f^{gp}_i(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) = \exp\left\{ -\frac{1}{2}\|\boldsymbol{\Phi}(t_{i+1}, t_i)\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1} + \mathbf{u}_{i,i+1}\|^2_{\mathbf{Q}_{i,i+1}} \right\} \tag{2.42}$$

where $\mathbf{u}_{a,b} = \int_{t_a}^{t_b} \boldsymbol{\Phi}(b, s)\mathbf{u}(s)\,\mathrm{d}s$, $\boldsymbol{\Phi}(t_{i+1}, t_i)$ is the state transition matrix, and $\mathbf{Q}_{i,i+1}$ is defined by Eq. (2.14) (see [72] for details).

To factor the likelihood $l(\boldsymbol{\theta}; \mathbf{e})$ (here we primarily consider obstacle avoidance), we define two types of obstacle cost factors: regular obstacle factors f^{obs}_i and interpolated

obstacle factors $f_{\tau_j}^{intp}$. The $l(\boldsymbol{\theta};\mathbf{e})$ is the product of all obstacle factors

$$l(\boldsymbol{\theta};\mathbf{e}) = \prod_{i=0}^{N} \left\{ f_i^{obs}(\boldsymbol{\theta}_i) \prod_{j=1}^{n_{ip}} f_{\tau_j}^{intp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) \right\}, \tag{2.43}$$

where n_{ip} is the number of *interpolated* states defined between each nearby support state pair $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{i+1}$, and τ_j is the time to perform interpolation which satisfies $t_i < \tau_j < t_{i+1}$.

The regular obstacle factor describes the obstacle cost on a single state variable and is a *unary* factor defined as

$$f_i^{obs}(\boldsymbol{\theta}_i) = \exp\left\{ -\frac{1}{2} \parallel \mathbf{h}(\boldsymbol{\theta}_i) \parallel^2_{\boldsymbol{\sigma}_{obs}} \right\}, \tag{2.44}$$

where $\mathbf{h}(\boldsymbol{\theta}_i)$ is a M-dimensional vector-valued obstacle cost function for a single state, and $\boldsymbol{\sigma}_{obs}$ is a $M \times M$ hyperparameter matrix.

The interpolated obstacle factor describes the obstacle cost at τ_j, which is not on any support state and needs be interpolated from the support states. Since the Gauss-Markov model we choose enables fast interpolation from adjacent states, we can interpolate a state at any τ_j from $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{i+1}$ by Eq. (2.17), which satisfies $t_i < \tau_j < t_{i+1}$. This allows us to derive a *binary* interpolated obstacle factor that relates the cost at an interpolated point to the adjacent two trajectory states

$$\begin{aligned} f_{\tau_j}^{intp}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) &= \exp\left\{ -\frac{1}{2} \parallel \mathbf{h}(\boldsymbol{\theta}(\tau_j)) \parallel^2_{\boldsymbol{\sigma}_{obs}} \right\} \\ &= \exp\left\{ -\frac{1}{2} \parallel \mathbf{h}^{intp}_{\tau_j}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) \parallel^2_{\boldsymbol{\sigma}_{obs}} \right\}. \end{aligned} \tag{2.45}$$

In other words, $\boldsymbol{\theta}(\tau_j)$ is a function of $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_{i+1}$ (see Eq. (2.17)). Just like in GPMP, here too the interpolated obstacle factor incorporates the obstacle information at all τ in the factor graph and is utilized to meaningfully update the sparse set of support states.

An example factor graph that combines all of the factors described above is illustrated in Figure 2.5. Note that if there are enough support states to densely cover the trajec-

tory, interpolated obstacle factors are not needed. But to fully utilize the power of the continuous-time trajectory representation and to maximize performance, the use of sparse support states along with interpolated obstacle factor is encouraged.

Given the factorized obstacle likelihood in Eq. (2.43)-(2.45), we can retrieve the vector-valued obstacle cost function of the trajectory defined in Eq. (2.38) by simply stacking all the vector-valued obstacle cost functions on all regular and interpolated states into a single vector

$$\begin{aligned}\boldsymbol{h}(\boldsymbol{\theta};\mathbf{e}) = \big[&\mathbf{h}(\boldsymbol{\theta}_0); \mathbf{h}^{intp}_{\tau_1}(\boldsymbol{\theta}_0,\boldsymbol{\theta}_1); \ldots; \mathbf{h}^{intp}_{\tau_{n_{ip}}}(\boldsymbol{\theta}_0,\boldsymbol{\theta}_1);\\ &\mathbf{h}(\boldsymbol{\theta}_1); \mathbf{h}^{intp}_{\tau_1}(\boldsymbol{\theta}_1,\boldsymbol{\theta}_2); \ldots; \mathbf{h}^{intp}_{\tau_{n_{ip}}}(\boldsymbol{\theta}_1,\boldsymbol{\theta}_2);\\ &\ldots\\ &\mathbf{h}(\boldsymbol{\theta}_{N-1}); \mathbf{h}^{intp}_{\tau_1}(\boldsymbol{\theta}_{N-1},\boldsymbol{\theta}_N); \ldots; \mathbf{h}^{intp}_{\tau_{n_{ip}}}(\boldsymbol{\theta}_{N-1},\boldsymbol{\theta}_N);\\ &\mathbf{h}(\boldsymbol{\theta}_N)],\end{aligned} \tag{2.46}$$

where all $\mathbf{h}$ are obstacle cost functions from regular obstacle factors defined in Eq. (2.44), and all $\mathbf{h}^{intp}$ are obstacle cost functions from interpolated obstacle factors defined in Eq. (2.45). Since there are a total of $N + 1$ regular obstacle factors on support states, and n_{ip} interpolated factors between each support state pair, the total dimensionality of $\boldsymbol{h}(\boldsymbol{\theta})$ is $M \times (N + 1 + N \times n_{ip})$. The hyperparameter matrix $\boldsymbol{\Sigma}$ in Eq. (2.38) is then defined by

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\sigma}_{obs} & & \\ & \ddots & \\ & & \boldsymbol{\sigma}_{obs} \end{bmatrix}, \tag{2.47}$$

which has size $M \times (N + 1 + N \times n_{ip})$ by $M \times (N + 1 + N \times n_{ip})$.

In our framework, the obstacle cost function $\mathbf{h}$ can be any nonlinear function, and the construction of $\mathbf{h}$, M, and $\boldsymbol{\sigma}_{obs}$ are flexible as long as $l(\boldsymbol{\theta};\mathbf{e})$ gives the likelihood. Effectively $\mathbf{h}(\boldsymbol{\theta}_i)$ should have a larger value when a robot collides with obstacles at $\boldsymbol{\theta}_i$, and

a smaller value when the robot is collision-free. Our implementation of $\mathbf{h}$, definition of M, and guideline for the hyperparameter $\boldsymbol{\sigma}_{obs}$ is discussed in Section 2.9.2.

To solve the MAP inference problem in Eq. (2.35), we first illustrate the duality between inference and optimization by performing minimization on the negative log of the posterior distribution

$$\begin{aligned}\boldsymbol{\theta}^* &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta}) l(\boldsymbol{\theta}; \mathbf{e}) \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ -\log\Big(p(\boldsymbol{\theta}) l(\boldsymbol{\theta}; \mathbf{e})\Big) \right\} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\boldsymbol{\mathcal{K}}} + \frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e}) \parallel^2_{\boldsymbol{\Sigma}} \right\} \end{aligned} \tag{2.48}$$

where Eq. (2.48) follows from Eq. (2.37) and Eq. (2.38). This duality connects the two different perspectives on motion planning problems such that the terms in Eq. (2.48) can be viewed as 'cost' to be minimized, or information to be maximized. The apparent construction of the posterior now becomes clear as we have a nonlinear least squares optimization problem, which has been well studied and for which many numerical tools are available. Iterative approaches, like Gauss-Newton or Levenberg-Marquardt repeatedly resolve a quadratic approximation of Eq. (2.48) until convergence.

Linearizing the nonlinear obstacle cost function around the current trajectory $\boldsymbol{\theta}$

$$\boldsymbol{h}(\boldsymbol{\theta} + \mathrm{d}\boldsymbol{\theta}; \mathbf{e}) \approx \boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e}) + \mathbf{H}\, \mathrm{d}\boldsymbol{\theta} \tag{2.49}$$

$$\mathbf{H} = \frac{\mathrm{d}\boldsymbol{h}}{\mathrm{d}\boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}} \tag{2.50}$$

where $\mathbf{H}$ is the Jacobian matrix of $\boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e})$, we convert Eq. (2.48) to a linear least squares problem

$$\delta\boldsymbol{\theta}^* = \underset{\delta\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\boldsymbol{\theta} + \delta\boldsymbol{\theta} - \boldsymbol{\mu} \parallel^2_{\boldsymbol{\mathcal{K}}} + \frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e}) + \mathbf{H}\delta\boldsymbol{\theta} \parallel^2_{\boldsymbol{\Sigma}} \right\}. \tag{2.51}$$

The optimal perturbation $\delta\boldsymbol{\theta}^*$ results from solving the following linear system

$$(\boldsymbol{\mathcal{K}}^{-1} + \mathbf{H}^\top \boldsymbol{\Sigma}^{-1} \mathbf{H}) \delta\boldsymbol{\theta}^* = -\boldsymbol{\mathcal{K}}^{-1}(\boldsymbol{\theta} - \boldsymbol{\mu}) - \mathbf{H}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e}) \tag{2.52}$$

Once the linear system is solved, the iteration

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta}^* \tag{2.53}$$

is applied until convergence criteria are met. Eq. (2.53) serves as the update rule for GPMP2.

If the linear system in Eq. (2.52) is sparse, then $\delta\boldsymbol{\theta}^*$ can be solved efficiently by exploiting the sparse Cholesky decomposition followed by forward-backward passes [90]. Fortunately, this is the case: we have selected a Gaussian process prior with a block tridiagonal precision matrix $\boldsymbol{\mathcal{K}}^{-1}$ (Section 2.4.2) and $\mathbf{H}^\top \boldsymbol{\Sigma}^{-1} \mathbf{H}$ is also block tridiagonal (see proof in Appendix B). The structure exploiting iteration combined with the quadratic convergence rate of nonlinear least squares optimization method we employ (Gauss-Newton or Levenberg-Marquardt) makes GPMP2 more efficient and faster compared to GPMP.

2.8 Incremental inference for fast replanning

We have described how to formulate motion planning problem as probabilistic inference on factor graphs, results in fast planning through least squares optimization. In this section, we show that this perspective also gives us the flexibility to use other inference and optimization tools on factor graphs. In particular, we describe how factor graphs can be used to perform *incremental* updates to solve *replanning* problems efficiently.

The replanning problem can be defined as: given a solved motion planning problem, resolve the problem with partially changed conditions. Replanning problems are commonly encountered in the real world, when, for example: (i) the goal position for the end-effector has changed during middle of the execution; (ii) the robot receives updated estimation about

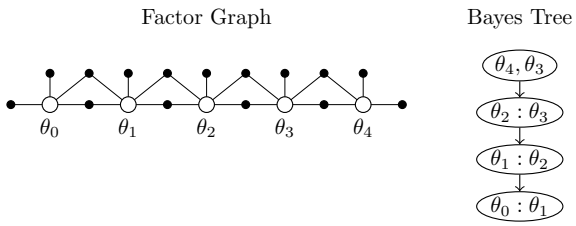


Figure 2.6: Example of a Bayes Tree with its corresponding factor graph.

its current state; or (iii) new information about the environment is available. Since replanning is performed online, possibly in dynamic environments, fast real-time replanning is critical to ensuring safety.

A naïve way to solve this problem is to literally replan by re-optimizing from scratch. However, this is potentially too slow for real-time settings. Furthermore, if the majority of the problem is left unchanged, resolving the entire problem duplicates work and should be avoided to improve efficiency. Here we adopt an incremental approach to updating the current solution given new or updated information. We use the *Bayes tree* [91, 43] data structure to perform incremental inference on factor graphs.[1]

Two replanning examples with Bayes tree are shown in Figure 2.7. The first example shows replanning when the goal configuration changes causing an update to the prior factor on the goal state. When the Bayes tree is updated with the new goal, only the root node of the tree is changed. The second example shows a replanning problem, given an observation of the current configuration (e.g. from perception during execution) that is added as a prior factor at $\boldsymbol{\theta}_2$ where the estimation was taken. When the Bayes Tree is updated, the parts of the tree that change correspond to the parts of the trajectory that get updated.

In our implementation, we use the iSAM2 incremental solver [43] within the GPMP2 framework to solve the replanning problem. We call this incremental variant of GPMP2, **iGPMP2**. A replanning scenario typically has the following steps. First, the original batch problem is solved with GPMP2. Then, we collect the additional information to form factors

[1]Given that the trajectories are represented by GPs, the incremental updates of the factor graphs can also be viewed as incremental GP regression [72].

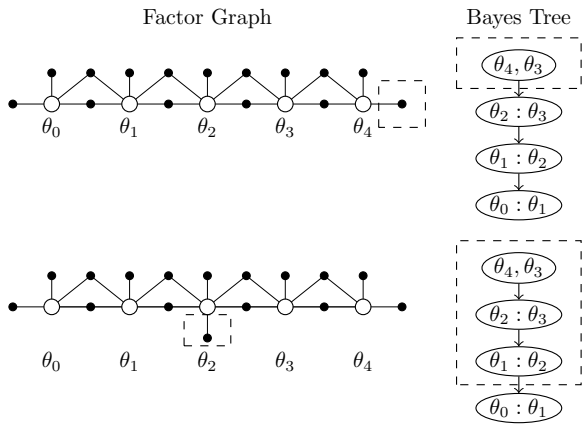


Figure 2.7: Replanning examples using Bayes Trees. Dashed boxes indicate parts of the factor graphs and Bayes Trees that are affected and changed while performing replanning.

that need to be added or replaced within the factor graph. Finally, we update the Bayes Tree inside iSAM2, to get a newly updated optimal solution.

2.9 Implementation details

GPMP is implemented on top of the CHOMP [44] code since it uses an identical framework, albeit with several augmentations. To implement GPMP2 and iGPMP2 algorithms, we used the GTSAM [92] library. Our implementation is available as a single open source C++ library, `gpmp2`.[2] In this section we describe the implementation details of our algorithms.

2.9.1 GPMP

GP prior:

GPMP employs a constant-acceleration (i.e. jerk-minimizing) prior to generate a trajectory with a Markovian state comprising of configuration position, velocity and acceleration, by

[2]Implementation along with a ROS interface available at `https://github.com/gtrll/gpmp2`

following the LTV-SDE in Eq. (2.5) with parameters

$$\mathbf{A}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \mathbf{u}(t) = \mathbf{0}, \mathbf{F}(t) = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{I} \end{bmatrix} \tag{2.54}$$

and given $\Delta t_i = t_{i+1} - t_i$,

$$\mathbf{\Phi}(t,s) = \begin{bmatrix} \mathbf{I} & (t-s)\mathbf{I} & \frac{1}{2}(t-s)^2\mathbf{I} \\ \mathbf{0} & \mathbf{I} & (t-s)\mathbf{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{2.55}$$

$$\mathbf{Q}_{i,i+1} = \begin{bmatrix} \frac{1}{2}\Delta t_i^5 \mathbf{Q}_C & \frac{1}{8}\Delta t_i^4 \mathbf{Q}_C & \frac{1}{6}\Delta t_i^3 \mathbf{Q}_C \\ \frac{1}{8}\Delta t_i^4 \mathbf{Q}_C & \frac{1}{3}\Delta t_i^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C \\ \frac{1}{6}\Delta t_i^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C & \Delta t_i \mathbf{Q}_C \end{bmatrix} \tag{2.56}$$

This prior is centered around a zero jerk trajectory and encourages smoothness by attempting to minimize jerk during optimization.

Obstacle avoidance and constraints:

To quickly calculate the collision cost for an arbitrary shape of the robot's physical body, GPMP represents the robot with a set of spheres, as in [44] (shown in Figure 2.8). This leads to a more tractable approximation to finding the signed distance from the robot surface to obstacles. GPMP uses the same obstacle cost function as CHOMP (see Eq. (2.20)) where the cost is summed over the sphere set on the robot body calculated using a precomputed *signed distance field* (SDF). Constraints are also handled in the same manner as CHOMP. Joint limits are enforced by smoothly projecting joint violations using the technique similar to projecting the obstacle gradient in Eq. (2.31). Along each point on the up-sampled trajectory the violations are calculated via L_1 projections to bring inside the limits (see [44] for details). Then they are collected into a violation trajectory, $\boldsymbol{\theta}_{up}^v$ to be

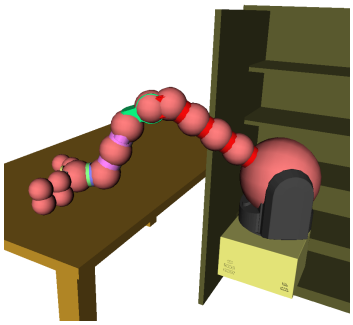


Figure 2.8: The WAM arm is represented by multiple spheres (pink), which are used during collision cost calculation.

projected:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathcal{K}\mathbf{M}^{\top}\boldsymbol{\theta}^{v}_{up}. \tag{2.57}$$

2.9.2 GPMP2 and iGPMP2

GPMP2 uses the Levenberg-Marquardt algorithm to solve the nonlinear least squares optimization problem, with the initial damping parameter set as 0.01. The optimization is stopped if a maximum of 100 iterations is reached, or if the relative decrease in error is smaller than 10^{-4}. iGPMP2 uses the iSAM2 [43] incremental optimizer with default settings.

GP prior:

We use a constant-velocity prior in GPMP2 with the Markovian state comprising of configuration position and velocity. Note that, unlike GPMP, we did not include acceleration since it was not needed for any gradients and an acceleration-minimizing prior for optimization was sufficient for the tasks we consider in this work. Ideally a jerk-minimizing trajectory would be beneficial to use on faster moving systems like quadrotors. GPMP2 scales only quadratically in computation with the size of the state. So even if the same prior as GPMP was used, GPMP2 would still be faster given its quadratic convergence rate.

The trajectory is similarly generated by following the LTV-SDE in Eq. (2.5) with

$$\mathbf{A}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \mathbf{u}(t) = \mathbf{0}, \mathbf{F}(t) = \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \tag{2.58}$$

and given $\Delta t_i = t_{i+1} - t_i$,

$$\mathbf{\Phi}(t,s) = \begin{bmatrix} \mathbf{I} & (t-s)\mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \mathbf{Q}_{i,i+1} = \begin{bmatrix} \frac{1}{3}\Delta t_i^3 \mathbf{Q}_C & \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C \\ \frac{1}{2}\Delta t_i^2 \mathbf{Q}_C & \Delta t_i \mathbf{Q}_C \end{bmatrix} \tag{2.59}$$

Analogously this prior is centered around a zero-acceleration trajectory.

Collision-free likelihood:

Similar to GPMP and CHOMP, the robot body is represented by a set of spheres as shown in Figure 2.8, and the obstacle cost function for any configuration $\boldsymbol{\theta}_i$ is then completed by computing the hinge loss for each sphere S_j ($j = 1, \ldots, M$) and collecting them into a single vector,

$$\mathbf{h}(\boldsymbol{\theta}_i) = [\mathbf{c}(\mathbf{d}(\mathbf{x}(\boldsymbol{\theta}_i, S_j)))]\Big|_{1 \le j \le M} \tag{2.60}$$

where $\mathbf{x}$ is the forward kinematics, $\mathbf{d}$ is the signed distance function, $\mathbf{c}$ is the hinge loss function, and M is the number of spheres that represent the robot model.

Forward kinematics $\mathbf{x}(\boldsymbol{\theta}_i, S_j)$ maps any configuration $\boldsymbol{\theta}_i$ to the 3D workspace, to find the center position of any sphere S_j. Given a sphere and its center position, we calculate $\mathbf{d}(x)$, the *signed distance* from the sphere at x to the closest obstacle surface in the workspace. The sphere shape makes the surface-to-surface distance easy to calculate, since it is equal to the distance from sphere center to closest obstacle surface minus the sphere radius. Using a precomputed signed distance field (SDF), stored in a voxel grid with a desired resolution, the signed distance of any position in 3D space is queried by trilinear

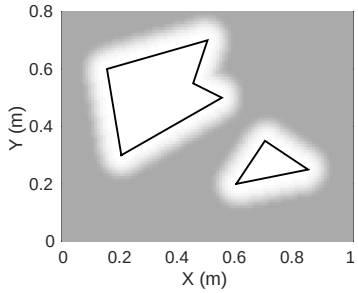


Figure 2.9: The likelihood function $\mathbf{h}$ in a 2D space with two obstacles and $\epsilon = 0.1$m. Obstacles are marked by black lines and darker area has higher likelihood for no-collision.

interpolation on the voxel grid. The hinge loss function[3] is defined as

$$\mathbf{c}(d) = \begin{cases} -d + \epsilon & \text{if } d \leqslant \epsilon \\ 0 & \text{if } d > \epsilon \end{cases} \tag{2.61}$$

where d is the signed distance, and ϵ is a 'safety distance' indicating the boundary of the 'danger area' near obstacle surfaces. By adding a non-zero obstacle cost, even if the robot is not in collision but rather too close to the obstacles, ϵ enables the robot to stay a minimum distance away from obstacles. The remaining parameter $\boldsymbol{\sigma}_{obs}$ needed to fully implement the likelihood in Eq. (2.44) and Eq. (2.45) is defined by an isotropic diagonal matrix

$$\boldsymbol{\Sigma} = \sigma_{obs}^2 \mathbf{I}, \tag{2.62}$$

where σ_{obs} is the 'obstacle cost weight' parameter.

Figure 2.9 visualizes a 2D example of the collision-free likelihood defined by the obstacle cost function in Eq. (2.60). The darker region shows a free configuration space where the likelihood of no-collision is high. The small area beyond the boundary of the obstacles is lighter, implying 'safety marginals' defined by ϵ.

Note that the obstacle cost function used here is different from the one used in GPMP

[3]The hinge loss function is not differentiable at $d = \epsilon$, so in our implementation we set $\mathrm{d}\mathbf{c}(d)/\,\mathrm{d}d = -0.5$ when $d = \epsilon$.

and CHOMP, where $\mathbf{c}$ is instead a smooth function (necessary for gradient calculation) and is multiplied with the norm of the workspace velocity (see Eq. (2.20)). This arc-length parameterization helps in making the trajectory avoid obstacles rather than speeding through them, while minimizing cost. The GP prior we use for GPMP2 helps us achieve the same purpose, by incorporating cost on large accelerations. The choice of cost function in Eq. (2.61) serves as a good approximation for the tasks we consider and is also less computationally expensive.

Motion constraints:

Motion constraints exist in real-world planning problems and should be considered during trajectory optimization. Examples include the constrained start and goal states as well as constraints on any other states along the trajectory. Since we are solving unconstrained least square problems, there is no way to enforce direct equality or inequality constraints during inference. In our framework, these constraints are instead handled in a 'soft' way, by treating them as prior knowledge on the trajectory states with very small uncertainties. Although the constraints are not exact, this has not been an issue in practice in any of our evaluations.

Additional *equality* motion constraints, such as end-effector rotation constraints (e.g. holding a cup filled with water upright) written as $\mathbf{f}(\boldsymbol{\theta}_c) = \mathbf{0}$, where $\boldsymbol{\theta}_c$ is the set of states involved, can be incorporated into a likelihood,

$$L_{constraint}(\boldsymbol{\theta}) \propto \exp\left\{ -\frac{1}{2} \parallel \mathbf{f}(\boldsymbol{\theta}_c) \parallel^2_{\boldsymbol{\Sigma}_c} \right\} \tag{2.63}$$

where, $\boldsymbol{\Sigma}_c = \sigma_c^2 \mathbf{I}$, σ_c is an arbitrary variance for this constraint, indicating how 'tight' the constraint is.

To prevent joint-limit (and velocity-limit) violations, we add *inequality* soft constraint factors to the factor graph. Similar to obstacle factors, the inequality motion constraint

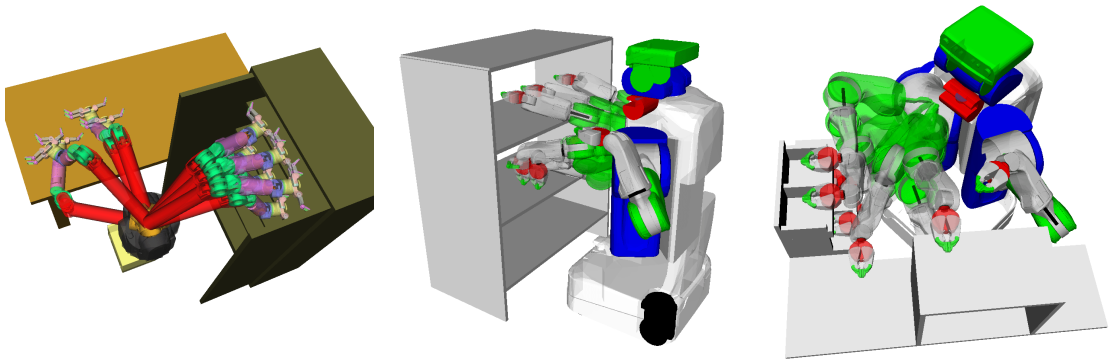


Figure 2.10: Environments used for evaluation with robot start and goal configurations showing the WAM dataset (left), and a subset of the PR2 dataset (*bookshelves* (center) and *industrial* (right)).

factor uses a hinge loss function to enforce soft constraints at both the maximum $\boldsymbol{\theta}^d_{\text{max}}$ and the minimum $\boldsymbol{\theta}^d_{\text{min}}$ values, with some given safety margin ϵ on each dimension $d = \{1, \ldots, D\}$

$$\mathbf{c}(\boldsymbol{\theta}^d_i) = \begin{cases} -\boldsymbol{\theta}^d_i + \boldsymbol{\theta}^d_{\text{min}} - \epsilon & \text{if } \boldsymbol{\theta}^d_i < \boldsymbol{\theta}^d_{\text{min}} + \epsilon \\ \mathbf{0} & \text{if } \boldsymbol{\theta}^d_{\text{min}} + \epsilon \leq \boldsymbol{\theta}^d_i \leq \boldsymbol{\theta}^d_{\text{max}} - \epsilon \\ \boldsymbol{\theta}^d_i - \boldsymbol{\theta}^d_{\text{max}} + \epsilon & \text{if } \boldsymbol{\theta}^d_i > \boldsymbol{\theta}^d_{\text{max}} - \epsilon \end{cases} \tag{2.64}$$

This factor has a vector valued cost function $\mathbf{f}(\boldsymbol{\theta}_i) = [\mathbf{c}(\boldsymbol{\theta}^d_i)]\Big|_{1 \leq d \leq D}$ and the same likelihood as the equality constraint factor in Eq. (2.63). At the final iteration we also detect limit violations and clamp to the maximum or minimum values.

2.10 Evaluation

We conducted our experiments[4] on two datasets with different start and goal configurations. We used: (1) the 7-DOF WAM arm dataset [36] consisting of 24 unique planning problems in the *lab* environment; and (2) the PR2's 7-DOF right arm dataset [45] consisting of a total of 198 unique planning problems in four different environments (Figure 2.10). Finally, we

[4] A video of experiments is available at `https://youtu.be/mVA8qhGf7So`.

validated successful trajectories on a real 7-DOF WAM arm in an environment identical to the simulation (Figure 2.1).

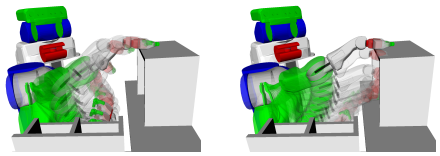
2.10.1 Batch planning benchmark

Setup:

We benchmarked our algorithms, GPMP and GPMP2, both with interpolation (GPMP2-intp) during optimization and without interpolation (GPMP2-no-intp) against trajectory optimizations algorithms - TrajOpt [45] and CHOMP [44], and against sampling based algorithms - RRT-Connect [5] and LBKPIECE [46] available within the OMPL implementation [47]. All benchmarks were run on a single thread of a 3.4GHz Intel Core i7 CPU.

For trajectory optimizers, GPMP2, TrajOpt and CHOMP were initialized by a constant-velocity straight line trajectory in configuration space and GPMP was initialized by an acceleration-smooth straight line. For the WAM dataset all initialized trajectories were parameterized by 101 temporally equidistant states. GPMP2-intp and GPMP use interpolation so we initialized them with 11 support states and $n_{ip} = 9$ (101 states effectively). Since trajectory tasks are shorter in the PR2 dataset, we used 61 temporally equidistant states to initialize the trajectories and for GPMP2-intp and GPMP we used 11 support states and $n_{ip} = 5$ (61 states effectively).

To keep comparisons fair we also compared against TrajOpt using only 11 states (TrajOpt-11) in both datasets since it uses continuous-time collision checking and can usually find a successful trajectory with fewer states. Although TrajOpt is faster when using fewer states, post-processing on the resulting trajectory is needed to make it executable and keep it smooth. It is interesting to note that since the continuous time-collision checking is performed only linearly, after the trajectory is post-processed it may not offer any collision-free guarantees. GPMP and GPMP2 avoid this problem when using fewer states by up-sampling the trajectory with GP interpolation and checking for collision at the interpolated points. This up-sampled trajectory remains smooth and can be used directly during execution. For



(a) $\sigma_{obs} = 0.005$ (b) $\sigma_{obs} = 0.05$

Figure 2.11: (a) shows a successful trajectory with a good selection of σ_{obs}; (b) shows failure, where the trajectory collides with the top part of the shelf, when σ_{obs} is too large.

sampling-based planners no post processing or smoothing step was applied and they were used with default settings.

All algorithms were allowed to run for a maximum of 10 seconds on any problem and marked successful if a feasible solution is found in that time. GPMP, CHOMP, RRT-Connect and LBKPIECE are stopped if a collision free trajectory is found before the max time (for GPMP and CHOMP collision checking is started after optimizing for at least 10 iterations). GPMP2 and TrajOpt are stopped when convergence is reached before the max time (we observed this was always the case) and feasibility is evaluated post-optimization.

Parameters:

For both GPMP and GPMP2, $\mathbf{Q}_C$ controls the uncertainty in the prior distribution. A higher value means the trajectories will have a lower cost on deviating from the mean and the distribution covers a wider area of the configuration space. Thus a higher value is preferable in problems with more difficult navigation constraints. However, a very high value might result in noisy trajectories since the weight on the smoothness cost becomes relatively low. A reverse effect will be seen with a smaller value. This parameter can be set based on the problem and the prior model used (for example, constant velocity or constant acceleration). In our benchmarks, for GPMP we set $\mathbf{Q}_C = 100$ for the WAM dataset and $\mathbf{Q}_C = 50$ for the PR2 dataset and for GPMP2 we set $\mathbf{Q}_C = 1$ for both datasets.

Another common parameter, 'safety distance,' ϵ is selected to be about double the minimum distance to any obstacle allowed in the scene and should be adjusted based on the robot, environment, and the obstacle cost function used. In our benchmarks we set

$\epsilon = 0.2$m for both GPMP and GPMP2 for the WAM dataset, and $\epsilon = 0.05$m for GPMP and $\epsilon = 0.08$m for GPMP2 for the PR2 dataset.

For GPMP2 the 'obstacle cost weight' σ_{obs} acts like a weight term that balances smoothness and collision-free requirements on the optimized trajectory and is set based on the application. Smaller σ_{obs} puts more weight on obstacle avoidance and vice versa. Figure 2.11 shows an example of an optimized trajectory for PR2 with different settings of σ_{obs}. In our experiments we found that the range $[0.001, 0.02]$ works well for σ_{obs} and larger robot arms should use larger σ_{obs}. In the benchmarks we set $\sigma_{obs} = 0.02$m for the WAM dataset and $\sigma_{obs} = 0.005$ for the PR2 dataset.

Analysis:

The benchmark results for the WAM dataset are summarized in Table 2.1[5] and for the PR2 dataset are summarized in Table 2.2[6]. Average time and maximum time include only successful runs.

Evaluating motion planning algorithms is a challenging task. The algorithms here use different techniques to formulate and solve the motion planning problem, and exhibit performance that depends on initial conditions as well as a range of parameter settings that can change based on the nature of the planning problem. Therefore, in our experiments we have tuned each algorithm to the settings close to default ones that worked best for each dataset. However, we still observe that TrajOpt-11 performs poorly on the WAM dataset (possibly due to using too few states on the trajectory) while GPMP performs poorly on the PR2 dataset (possibly due to the different initialization of the trajectory, and also the start and end configurations in the dataset being very close to the obstacles).

From the results in Table 2.1 and 2.2 we see that GPMP2 perform consistently well compared to other algorithms on these datasets. Using interpolation during optimization

[5]Parameters for benchmark on the WAM dataset: For GPMP and CHOMP, $\lambda = 0.005$, $\eta = 1$. For CHOMP, $\epsilon = 0.2$. For TrajOpt, coeffs $= 20$, dist_pen $= 0.05$.

[6]Parameters for benchmark on the PR2 dataset: For CHOMP, $\epsilon = 0.05$. All remaining parameters are the same from the WAM dataset.

Table 2.1: Results for 24 planning problems on the 7-DOF WAM arm.

	GPMP2-intp	**GPMP2-no-intp**	**TrajOpt-101**	**TrajOpt-11**	**GPMP**	**CHOMP**	**RRT-Connect**	**LBKPIECE**
Success (%)	91.7	**100.0**	91.7	20.8	95.8	75	91.7	62.5
Avg. Time (s)	0.121	0.384	0.313	**0.027**	0.3	0.695	1.87	6.89
Max Time (s)	0.367	0.587	0.443	**0.033**	0.554	2.868	5.18	9.97

Table 2.2: Results for 198 planning problems on PR2's 7-DOF right arm.

	GPMP2-intp	**GPMP2-no-intp**	**TrajOpt-61**	**TrajOpt-11**	**GPMP**	**CHOMP**	**RRT-Connect**	**LBKPIECE**
Success (%)	79.3	78.8	68.7	77.8	36.9	59.1	**82.3**	33.8
Avg. Time (s)	**0.11**	0.196	0.958	0.191	1.7	2.38	3.02	7.12
Max Time (s)	**0.476**	0.581	4.39	0.803	9.08	9.81	9.33	9.95

(GPMP2-intp) achieves $30 - 50\%$ speedup of average and maximum runtime when compared to not using interpolation (GPMP2-no-intp). On the WAM dataset TrajOpt-11 has the lowest runtime but is able to solve only 20% of the problems, while GPMP2-intp has the second lowest runtime with a much higher success rate. GPMP2-no-intp has the highest success rate. On the relatively harder PR2 dataset, GPMP2-intp has the lowest runtime and is twice as fast with a slightly higher success rate compared to TrajOpt-11. GPMP2-intp has the second highest success rate and is slightly behind RRT-Connect but is 30 times faster. The timing for RRT-Connect would further increase if a post processing or smoothing step was applied.

As seen from the max run times, GPMP2 always converges well before the maximum time limit and all the failure cases are due to infeasible local minima. Solutions like, random restarts (that are commonly employed) or GPMP-GRAPH [93], an extension to our approach that uses graph-based trajectories, can help contend with this issue.

To understand how the GP representation and the inference framework result in performance boost we compare timing breakdowns during any iteration for CHOMP, GPMP and GPMP2. Figure 2.12 shows the breakdown of average timing per task per iteration on the WAM dataset where the solution update portion (dark blue) incorporates the optimization costs. Table 2.3 shows average number of optimization iterations for successful runs in both the WAM and the PR2 datasets. We see that compared to CHOMP, GPMP is more expensive per iteration primarily from the computation of the Hessian, that is needed to find

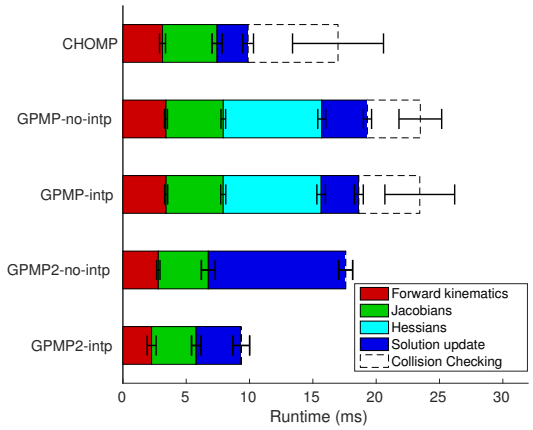


Figure 2.12: Breakdown of average timing per task **per iteration** on all problems in the WAM dataset is shown for CHOMP, GPMP-no-intp, GPMP-intp, GPMP2-no-intp and GPMP2-intp.

Table 2.3: Average number of optimization iterations on successful runs.

	CHOMP	**GPMP-no-intp**	**GPMP-intp**	**GPMP2-no-intp**	**GPMP2-intp**
WAM	26.4	11.5	12.0	23.6	13.0
PR2	46.6	32.2	19.1	26.4	24.4

the acceleration in workspace (CHOMP approximates the acceleration with finite differencing). However, due to the GP representation and gradients on the augmented trajectory, GPMP is able to take larger update steps and hence converge faster with fewer iterations. GPMP2 on the other hand takes advantage of quadratic convergence while also benefiting from the GP representation and the inference framework. GP interpolation further reduces the runtime per iteration, especially for GPMP2. The dashed bars in Figure 2.12 represent computational costs due to collision checking during optimization at a finer resolution, on top of the computational cost incurred to evaluate gradient information. This was necessary to determine convergence, since the CHOMP solution can jump in and out of feasibility between iterations [44]. GPMP also incurs this cost since it too exhibits this behavior due to its similar construction. Note that the total computational time in Table 2.1 reflects the total iteration time as shown in Figure 2.12 plus time before and after the iterations including setup and communication time.

2.10.2 Incremental planning benchmark

We evaluate our incremental motion planner iGPMP2 by benchmarking it against GPMP2 on replanning problems with the WAM and PR2 datasets.

For each problem in this benchmark, we have a planned trajectory from a start configuration to an originally assigned goal configuration. Then, at the middle time-step of the trajectory a new goal configuration is assigned. The replanning problem entails finding a trajectory to the newly assigned goal. This requires two changes to the factor graph: a new goal factor at the end of the trajectory to ensure that the trajectory reaches the new location in configuration, and a fixed state factor at the middle time step to enforce constraint of current state.

A total of 72 and 54 replanning problems are prepared for the WAM and the PR2 datasets, respectively. GP interpolation is used and all parameters are the same as the batch benchmarks. The benchmark results are shown in Table 2.4 and Table 2.5. We see from the results that iGPMP2 provides an order of magnitude speed-up, but suffers loss in success rate compared to GPMP2.

GPMP2 reinitializes the trajectory as a constant-velocity straight line from the middle state to the new goal and replans from scratch. However, iGPMP2 can use the solution to the old goal and the updated Bayes Tree as the initialization to incrementally update the trajectory, thus finding the solution much faster. There are three possible explanations why iGPMP2's success rate suffers as compared to the GPMP2's. First, iGPMP2 uses the original trajectory as initialization, which may be a poor choice if the goal has moved significantly. Second, in iSAM2 not every factor is relinearized and updated in Bayes tree for efficiency, which may lead to a poor linear approximation. Finally, GPMP2 uses Levenberg-Marquardt for optimization that provides appropriate step damping, helping to improve the results, but iGPMP2 does not use similar step damping in its current implementation. Since relinearizing factors or reinitializing variables will update the corresponding cliques of the Bayes tree and break its incremental nature, this results in runtime similar to

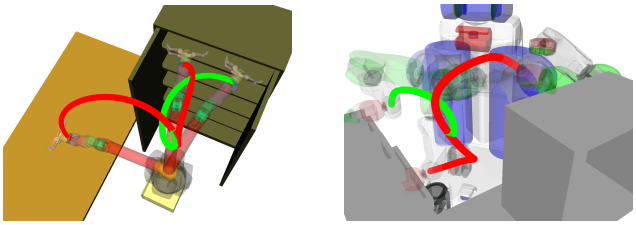


Figure 2.13: Example iGPMP2 results on the WAM and PR2 *industrial*. Red lines show originally planned end-effector trajectories, and green lines show replanned end-effector trajectories. Best viewed in color.

Table 2.4: Results for 72 replanning problems on WAM.

	iGPMP2	**GPMP2**
Success (%)	**100.0**	**100.0**
Avg. Time (ms)	**8.07**	65.68
Max Time (ms)	**12.65**	148.31

Table 2.5: Results for 54 replanning problems on PR2.

	iGPMP2	**GPMP2**
Success (%)	66.7	**88.9**
Avg. Time (ms)	**6.17**	27.30
Max Time (ms)	**7.37**	87.95

batch optimization, and should not be done frequently. A good heuristic is to only perform relinearization/reinitialization when a planning failure is detected. We leave the task of designing a better solution to overcome this issue as future work.

To maximize performance and overcome the deficiencies of iGPMP2, the rule of thumb when using iGPMP2 for replanning is to keep the difference between replanning problems and existing solutions to a minimum. This will lead to better initialization and reduced effect of linearization errors, and thus will improve iGPMP2's success rate. We verify this with the PR2 benchmark, where a smaller distance between original goal configuration and new goal configuration means a smaller difference between the replanning problem and the existing solution. We use $L2$ distance $\| \boldsymbol{\theta}_o - \boldsymbol{\theta}_r \|_2$ to quantify the distance between the original goal $\boldsymbol{\theta}_o$ and the new goal $\boldsymbol{\theta}_r$. From the PR2 benchmark, 27 problems have $\| \boldsymbol{\theta}_o - \boldsymbol{\theta}_r \|_2 < 2.0$, where iGPMP2 has 81.5% success rate. On the other hand, we see that for the remaining 27 problems where $\| \boldsymbol{\theta}_o - \boldsymbol{\theta}_r \|_2 \geq 2.0$, iGPMP2 only has 51.9% success

rate.

Examples of successfully replanned trajectories generated using iGPMP2 are shown in Figure 2.13. The use of the fixed state factor at the middle time step helps make a smooth transition between original trajectories and replanned trajectories, which is critical if the trajectory is being executed on a real robot.

2.11 Discussion

Comparisons with related work

GPMP can be viewed as a generalization on CHOMP where the trajectory is a sample from a GP and is augmented with velocities and accelerations. Both GPMP and GPMP2 use the GP representation for a continuous-time trajectory, GP interpolation, and signed distance fields for collision checking. However, with GPMP2 we fully embrace the probabilistic view of motion planning. In contrast to similar views on motion planning [15, 14] that use message passing, we instead solve the inference problem as nonlinear least squares. This allows us to use solvers with quadratic convergence rates that exploit the sparse structure of our problem, leading to a much faster algorithm compared to GPMP (and CHOMP) that only has linear convergence and is encumbered by the slow gradient computation. The update step in GPMP2 involves only linearization and the Cholesky decomposition to solve the linear system.

TrajOpt [11, 45] formulates the motion planning problem as constrained optimization, which allows the use of hard constraints on obstacles but also makes the optimization problem much more difficult and, as a consequence, slower to solve. Benchmark results in Section 2.10.1 show that our approach is faster than TrajOpt even when it uses a small number of states to represent the trajectory. TrajOpt performs continuous-time collision checking and can, therefore, solve problems with only a few states, in theory. However, the trajectory does not have a continuous-time representation and therefore must perform collision checking by approximating the convex-hull of obstacles and a straight line be-

tween states. This may not work in practice since a trajectory with few states would need to be post-processed to make it executable. Furthermore, depending on the post-processing method, collision-free guarantees may not exist for the final trajectory. Representing trajectories in continuous-time with GPs and using GP interpolation to up-sample them, allows our algorithms to circumvent this problem. GPMP2 has also been extended to support priors defined in general Lie groups [94]. This allows us to perform full body planning and incorporate state spaces for mobile manipulators like PR2.

Unlike sampling based methods, our algorithms do not guarantee probabilistic completeness. However, from the benchmarks we see that GPMP2 is efficient at finding locally optimal trajectories that are feasible from naïve straight line initialization that may be in collision. We note that trajectory optimization is prone to local minima and this strategy may not work on harder planning problems like mazes where sampling based methods excel. Recent work however, has begun to push the boundaries in trajectory optimization based planning. GPMP-GRAPH [93], an extension of our work, employs graph-based trajectories to explore exponential number of initializations simultaneously rather than trying them one at a time. Results show that it can quickly find feasible solutions even in mazes. Depending on the problem and time budget, multiple random initializations can also be a viable approach (since GPMP2 is fast), or GPMP2 can also be used on top of a path returned from a sampling based method to generate a time parameterized trajectory that is smooth.

Finally, our framework allows us to solve replanning problems very quickly, something that none of the above trajectory optimization approaches can provide. We are able to achieve this through incremental inference on a factor graph. On simpler replanning problems like changing goals, multi-query planners like PRM [4] can be useful but are time consuming since a large initial exploration of the space is necessary to build the first graph, a majority of which may not be needed. Solving these types of problems fast is very useful in real-time real-world applications.

Limitations

A drawback of iterative methods for solving nonlinear least square problems is that they offer no global optimality guarantees. However, given that our objective is to satisfy smoothness and to be collision-free, a globally optimal solution is not strictly necessary. Many of the prior approaches to motion planning face similar issues of getting stuck in local minima. Random restarts is a commonly used method to combat this, however our approach allows for a more principled way [93] in which this problem can be tackled.

The main drawback of our proposed approach is that it is limited in its ability to handle motion constraints like nonlinear inequality constraints. Sequential quadratic programming (SQP) can be used to solve problems with such constraints, and has been used before in motion planning [11, 45]. We believe that SQP can be integrated into our trajectory optimizer, although this remains future work. Such strategies would also be helpful in extending the planner to handle contact during manipulation. We have applied the inference technique to the trajectory estimation problem in this domain [95].

Part II

Learning on Factor Graphs

CHAPTER 3
COMBINING LFD AND MOTION PLANNING

3.1 Introduction

As robots assume collaborative roles alongside humans in dynamic environments, they must have the ability to learn and execute new behaviors to achieve desired tasks. To accomplish this, there are two established approaches for generating trajectories, namely, motion planning [6] and Learning from Demonstration (LfD) [96]. As discussed in Chapter 2, motion planning focuses on generating trajectories that are optimal with respect to pre-defined criteria (e.g. smooth accelerations) while maintaining feasibility (e.g. obstacle avoidance, reaching via points) [6]. LfD, on the other hand, aims to generate trajectories which satisfy the skill-based constraints learned from demonstrations [96, 97]. As a result, motion planning and LfD can be viewed as having complementary trade-offs. Motion planning generalizes well to new scenarios (comprising the desired/given robot states and the external environment) but requires precise optimality criteria that may be difficult to define for complicated skills, whereas trajectory-based LfD methods circumvent the need for hand coding optimality criteria, but typically do not generalize well.

By leveraging the framework and structure introduced in Chapter 2, here we develop an efficient approach to skill learning and generalizable skill reproduction that combines the strengths of motion planning and trajectory-based LfD while mitigating their weaknesses. We again view the problem of generating trajectories as equivalent to probabilistic inference, where a posterior distribution of successful trajectories is computed from a prior that encodes optimality, and a likelihood that characterizes feasibility in a given scenario. Earlier, the trajectory prior was simple and pre-defined: it encouraged trajectories that minimize acceleration. We now argue that the trajectory prior can instead be learned from a

set of demonstrations, and our key insight is that the resulting inference based planning paradigm is identical to skill reproduction. The resulting algorithm, Combined Learning from demonstration And Motion Planning (CLAMP) [88], performs probabilistic inference to compute a posterior distribution of trajectories encouraged to match demonstrations while remaining feasible for any given scenario. The structure induced by factor graphs within CLAMP allows us to retain the efficiency during inference and is critical in making learning generalizable, interpretable, and safe. We demonstrate these properties with the evaluations of our approach on three skills including *box-opening*, *drawer-opening*, and *picking*.

3.2 Related work

Probabilistic methods for trajectory-based LfD provide a viable way to learn a skill from multiple demonstrations. However, the generalization capabilities of these methods vary immensely. Purely probabilistic approaches, including Gaussian mixture models (GMM / GMR) [98] and LfD by Averaging Trajectories (LAT) [99], attract reproduced trajectories towards an average form of the demonstrated motions, without regard to the initial or goal state. Task-parameterized GMM/GMRs [100] generalize better by assigning reference frames to relevant objects and landmarks. Attempts at combining probabilistic approaches with dynamical systems [101, 102, 103] have also met some success at generalization. However, these methods generally require tedious parameter tuning to generate the desired skill models. Although Gaussian processes (GPs) provide a non-parametric alternative [104, 105], the computational complexity of conventional GP approaches scales cubically with the number of data points, limiting their effectiveness in trajectory-based LfD settings.

CLAMP assumes that the demonstrated trajectories are governed by a latent stochastic feedback control policy, which can be approximated as a linear stochastic dynamical system. This simple yet powerful assumption yields a GP over trajectories with an ex-

actly sparse inverse kernel matrix, enabling a significant boost in learning and inference efficiency. This GP produces a Gaussian prior distribution over trajectories. A similar approach, probabilistic movement primitives (ProMPs) [106] directly fits a Gaussian distribution over demonstrations. New skill constraints are incorporated in ProMPs via inference and feedback control policy is then found to follow the resulting trajectory distribution on a robot. In contrast, inference over the prior in CLAMP generates trajectories which naturally follow the demonstrated policy while satisfying all additional constraints.

We consider skill reproduction as performing inference over a prior trajectory distribution, which is of course related to our inference-based planning method GPMP2 [40] described earlier. Similarly, [107] show that trajectory adaptation to new start/goal states via dynamic movement primitives (DMPs) [108] is a result of pre-specified Hilbert norm minimization based on finite differences, thus drawing connections to CHOMP [44], a gradient-based trajectory optimizer. The norm minimization procedure in CLAMP, however, goes one step further by minimizing the Mahalanobis distance from a learned prior distribution.

Apart from generalizing skills over different start and goal states, skill reproduction should also generalize to environmental changes, e.g. avoiding unforeseen obstacles. Many conventional LfD approaches are not equipped to handle arbitrarily placed obstacles [98, 99]. Of those that do, obstacle avoidance is rather carried out reactively, without regard to optimality of the entire trajectory [108]. Since motion planning provides a principled way to handle obstacles, attempts at combining LfD and motion planning have been relatively more successful. [109] presented a hierarchical framework that adapts the output of a learned statistical model to avoid obstacles using a sampling-based motion planner as an ad-hoc post-processing step. However, since the aim of both LfD and motion planning is finding optimal and feasible trajectories, such a hierarchical approach induces redundancies by assuming the two constituent steps to be independent. Recent trajectory optimization based methods take a relatively more unified route. [110] carry out a functional gradient-

based optimization for reproduction similar to CHOMP. Not only does their optimization routine carry the same computational inefficiencies as CHOMP (see the discussion in Section 2.11), their demonstration based cost functional disregards the motion dynamics. On the other hand, [111], in a similar approach as ours, carry out probabilistic trajectory optimization. This method performs optimization as a (partially) redundant two-step process. An offline routine first learns a trajectory distribution in the presence of new obstacles and fits a ProMP to represent it, and then an online routine adapts the ProMP given new start/goal states or via-points. A major disadvantage of this approach is that the trajectory distribution has to be re-learned every time an obstacle is displaced or further new obstacles are introduced. In CLAMP, all skill generalization routines are carried out in an efficient one-shot posterior inference procedure, while the trajectory distribution (prior) only encodes human demonstrations.

3.3 The trajectory prior as a skill model

We argue that for generalizable skill reproduction, LfD should adhere to the same motivation as motion planning: finding trajectories that are optimal and feasible. In contrast to motion planning, where optimality is pre-specified (e.g. smooth accelerations), LfD would require the optimality criteria to be learned from demonstrations. The feasibility criteria represent the reproduction scenario, e.g. collision avoidance, a fixed start state, reaching a desired goal/via-point, or a combination thereof. We adopt the probabilistic inference perspective on motion planning introduced in Chapter 2 that naturally allows the incorporation of optimality metrics learned from demonstrations in the form of a prior distribution. Feasibility is encoded into a likelihood function specified in terms of a collection of binary events $\mathbf{e}$. Then the desired trajectory can be found by calculating the maximum a posteriori (MAP) trajectory from the prior and likelihood by Eq. (2.35). Figure 3.1 illustrates our framework.

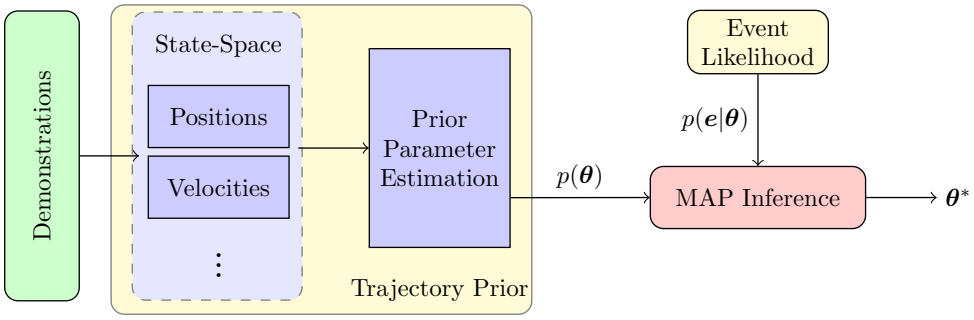


Figure 3.1: Block diagram showing various components of CLAMP.

3.3.1 Structured heteroscedastic GPs

We again use structured Gaussian processes (GPs) [82] to generate the trajectory prior, but this time we allow them to be heteroscedastic (time-varying covariance). The inherent sparsity in the precision matrix (i.e. inverse covariance matrix) associated with these GPs can be exploited in both learning and inference for efficient computation.

We view trajectories as solutions to a linear time-varying stochastic differential equation (LTV-SDE)

$$\dot{\boldsymbol{\theta}}(t) = \mathbf{A}(t)\boldsymbol{\theta}(t) + \mathbf{u}(t) + \mathbf{F}(t)\mathbf{w}(t), \quad \mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_C(t)\delta(t - t')), \tag{3.1}$$

where $\boldsymbol{\theta}(t)$ is the instantaneous robot state consisting of vectorized current positions and their higher-order time derivatives (for all degrees of freedom), $\mathbf{u}(t)$ is a bias term, $\mathbf{A}(t)$ and $\mathbf{F}(t)$ are time-varying system matrices and $\mathbf{w}(t)$ is a white noise process with covariance $\mathbf{Q}_C(t)$ and dirac-delta δ. However, the key difference here is that the covariance $\mathbf{Q}_C(t)$, is time-varying and hence generates a heteroscedastic GP, which is suitable for encoding the different ways of executing a skill.

Taking the first and second moments of the solution to the LTV-SDE yields the desired

GP with

$$\boldsymbol{\mu}(t) = \boldsymbol{\Phi}(t, t_0)\boldsymbol{\mu}_0 + \int_{t_0}^{t} \boldsymbol{\Phi}(t, s)\mathbf{u}(s)ds, \tag{3.2}$$

$$\boldsymbol{\mathcal{K}}(t, t') = \boldsymbol{\Phi}(t, t_0)\mathbf{Q}_0\boldsymbol{\Phi}(t', t_0)^T + \int_{t_0}^{\min(t,t')} \boldsymbol{\Phi}(t, s)\mathbf{F}(s)\mathbf{Q}_C(s)\mathbf{F}(s)^T\boldsymbol{\Phi}(t', s)^T ds \tag{3.3}$$

where $\boldsymbol{\Phi}(t, s)$ is the state transition matrix, and $\boldsymbol{\mu}_0$ and $\mathbf{Q}_0$ are the initial mean and covariance. Following [82], we can decompose the mean, covariance and precision (i.e the inverse covariance) of the GP parameterized by a finite number of support states $\boldsymbol{\theta} = [\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N]^T$ as

$$\boldsymbol{\mu} = \mathbf{A}\mathbf{u}, \quad \boldsymbol{\mathcal{K}} = \mathbf{A}\mathbf{Q}\mathbf{A}^T, \quad \boldsymbol{\mathcal{K}}^{-1} = \mathbf{A}^{-T}\mathbf{Q}^{-1}\mathbf{A}^{-1}, \tag{3.4}$$

where,

$$\boldsymbol{\mu} = \Big[\boldsymbol{\mu}(t_0), \boldsymbol{\mu}(t_1), \ldots \boldsymbol{\mu}(t_N)\Big]^T,$$

$$\mathbf{u} = \Big[\boldsymbol{\mu}_0, \mathbf{u}_{0,1}, \ldots \mathbf{u}_{N-1,N}\Big]^T, \quad \mathbf{u}_{i,i+1} = \int_{t_i}^{t_{i+1}} \boldsymbol{\Phi}(t_{i+1}, s)\mathbf{u}(s)ds,$$

$$\mathbf{Q} = \text{diag}(\mathbf{Q}_0, \mathbf{Q}_{0,1}, \ldots, \mathbf{Q}_{N-1,N}),$$

$$\mathbf{Q}_{i,i+1} = \int_{t_i}^{t_{i+1}} \boldsymbol{\Phi}(t_{i+1}, s)\mathbf{F}(s)\mathbf{Q}_C(s)\mathbf{F}(s)^T\boldsymbol{\Phi}(t_{i+1}, s)^T ds,$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Phi}(t_1, t_0) & \mathbf{1} & \ldots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Phi}(t_2, t_0) & \boldsymbol{\Phi}(t_2, t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Phi}(t_{N-1}, t_0) & \boldsymbol{\Phi}(t_{N-1}, t_1) & \ldots & \mathbf{1} & \mathbf{0} \\ \boldsymbol{\Phi}(t_N, t_0) & \boldsymbol{\Phi}(t_N, t_1) & \ldots & \boldsymbol{\Phi}(t_N, t_{N-1}) & \mathbf{1} \end{bmatrix}.$$

Due to the lower-triangular form of $\mathbf{A}$, and the block-diagonal form of $\mathbf{Q}$, the precision matrix $\boldsymbol{\mathcal{K}}^{-1}$ has a block-tridiagonal structure. In Section 3.4, we show how to perform fast

and efficient inference by exploiting the exactly sparse structure of this precision matrix. Note that, for the remainder of this chapter, $\boldsymbol{\theta}(t)$ will specifically refer to the robot's state in configuration space.

3.3.2 A combined prior

Usually, only the demonstrated workspace trajectories are relevant for skill execution. Therefore, we choose to learn a prior distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$ from demonstrations, generated by using the LTV-SDE described above, but defined over the robot's end effector state in workspace $\boldsymbol{x}(t)$, instead of that in configuration space $\boldsymbol{\theta}(t)$. We can directly use $p(\boldsymbol{x}|\boldsymbol{\theta})$ as the prior in Eq. (2.35) to generate a MAP trajectory in workspace. However, the problem of finding an associated configuration space trajectory is under-constrained for high-degree-of-freedom robots. To resolve this, we reintroduce the pre-specified smoothness prior in configuration space, $p(\boldsymbol{\theta}) \propto \exp\{-\frac{1}{2}||\boldsymbol{\theta} - \boldsymbol{\mu}^{\boldsymbol{\theta}}||^2_{\boldsymbol{\mathcal{K}}^{\boldsymbol{\theta}}}\}$, giving a combined configuration space prior

$$p_{\boldsymbol{x}}(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\boldsymbol{x}) \propto p(\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta}). \tag{3.5}$$

The combined prior eventually functions as our skill model instead of (2.4). The effect of the combined prior is to yield trajectories that are similar to the demonstrations given in workspace while at the same time maintaining smoothness in configuration space. Next, we detail the procedure for learning the workspace prior $p(\boldsymbol{x}|\boldsymbol{\theta})$. The configuration space smoothness prior given by $p(\boldsymbol{\theta})$, is analogous to the homoscedastic GP prior used for motion planning in Section 2.6.

In practice, based on the skills required, we may instead choose to directly learn the prior $p(\boldsymbol{\theta})$ in the configuration space, by considering the configuration space demonstrations. In fact, any combination of learned or hand-coded priors in configuration or workspace can be used, as the skill dictates.

3.3.3 Learned workspace prior

The workspace prior distribution in Eq. (3.5) is defined as

$$p(\boldsymbol{x}|\boldsymbol{\theta}) \propto \exp\{-\frac{1}{2}||\mathbf{C}(\boldsymbol{\theta}) - \boldsymbol{\mu}^{\boldsymbol{x}}||^2_{\boldsymbol{\mathcal{K}}^{\boldsymbol{x}}}\}, \tag{3.6}$$

where the function $\mathbf{C}$ maps a trajectory in configuration space to a workspace trajectory, and the hyper-parameters $\boldsymbol{\mu}^{\boldsymbol{x}}$ and $\boldsymbol{\mathcal{K}}^{\boldsymbol{x}}$ are the mean and the covariance of the distribution. We seek to estimate these hyper-parameters from provided workspace demonstrations.

Since demonstrations are recorded at discrete time instances, we only have access to the support states $\boldsymbol{x}_i$ to estimate the underlying workspace LTV-SDE. A discrete version of the LTV-SDE in Eq. (3.1) proved sufficient for the experiments we considered in this chapter, defined as

$$\boldsymbol{x}_{i+1} = \boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)\boldsymbol{x}_i + \mathbf{u}^{\boldsymbol{x}}_{i,i+1} + \mathbf{w}^{\boldsymbol{x}}_{i,i+1}, \quad \mathbf{w}^{\boldsymbol{x}}_{i,i+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}^{\boldsymbol{x}}_{i,i+1}), \tag{3.7}$$

where the unknown parameters $\boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)$, $\mathbf{u}^{\boldsymbol{x}}_{i,i+1}$ and $\mathbf{Q}^{\boldsymbol{x}}_{i,i+1}$ are defined as in Eq. (3.4), but in workspace. Given a set of M trajectory demonstrations $\mathbf{X} = \{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^M\}$, the regularized maximum likelihood estimate of the unknown parameters for the time interval $[t_i, t_{i+1}]$ is given by

$$\boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i),\ \mathbf{u}^{\boldsymbol{x}}_{i,i+1} = \underset{\mathbf{u}^{\boldsymbol{x}}_{i,i+1},\ \boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1},t_i)}{\operatorname{argmin}} \sum_{m=1}^{M} \|\mathbf{r}^m_{i,i+1}\|^2 + \lambda \|\boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)\|^2_F, \tag{3.8}$$

$$\mathbf{Q}^{\boldsymbol{x}}_{i,i+1} = \frac{1}{M} \sum_{m=1}^{M} \mathbf{r}^m_{i,i+1} \mathbf{r}^{m^T}_{i,i+1}, \tag{3.9}$$

where the residual $\mathbf{r}^m_{i,i+1} = \mathbf{u}^{\boldsymbol{x}}_{i,i+1} - \boldsymbol{x}^m_{i+1} + \boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)\boldsymbol{x}^m_i$ and λ is the regularization parameter. We use linear ridge regression [112] to solve Eq. (3.8). The hyper-parameters of the prior are calculated using the relationships in (3.4). Note that, if necessary, a continuous formulation could be learned through variational inference [113].

3.4 Efficient inference via factor graphs

In this section, we exploit the sparsity of the underlying system to efficiently carry out MAP inference using the learned prior, to reproduce the skill. The structure of the precision matrix of a distribution is captured by the structure of its factor graph, i.e. a sparser precision matrix leads to a more factorized distribution. Efficiency during inference is a direct result of this factorization (see Section 2.7).

3.4.1 Prior factors

Using the structured GP formulation (Section 3.3.1), the combined prior in Eq. (3.5) can be factored as

$$p_{\boldsymbol{x}}(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta}) \propto f^{\text{gp},\boldsymbol{\theta}} f^{\text{gp},\boldsymbol{x}} = \prod_{i=0}^{N-1} f_i^{\text{gp},\boldsymbol{\theta}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1})\ f_i^{\text{gp},\boldsymbol{x}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}), \tag{3.10}$$

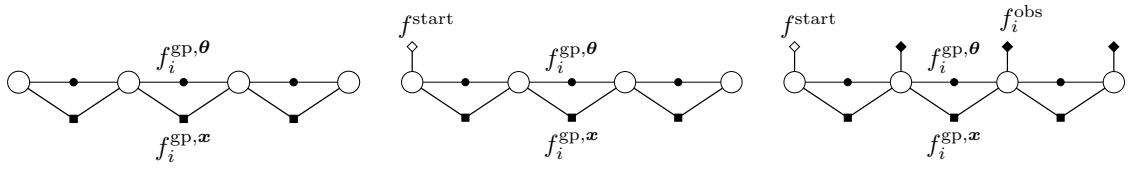
and is shown in Figure 3.2(a), where,

$$f_i^{\text{gp},\boldsymbol{x}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) = \exp\left\{-\frac{1}{2}\|\boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)\mathbf{C}(\boldsymbol{\theta}_i) - \mathbf{C}(\boldsymbol{\theta}_{i+1}) + \mathbf{u}^{\boldsymbol{x}}_{i,i+1}\|^2_{\mathbf{Q}^{\boldsymbol{x}}_{i,i+1}}\right\}, \tag{3.11}$$

are the workspace prior factors learned from demonstrations as described in Section 3.3.3, and

$$f_i^{\text{gp},\boldsymbol{\theta}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) = \exp\left\{-\frac{1}{2}\|\boldsymbol{\Phi}^{\boldsymbol{\theta}}(t_{i+1}, t_i)\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1} + \mathbf{u}^{\boldsymbol{\theta}}_{i,i+1}\|^2_{\mathbf{Q}^{\boldsymbol{\theta}}_{i,i+1}}\right\}, \tag{3.12}$$

are the pre-specified smoothness prior factors in configuration space (see Section 3.3.2) as described in Eq. (2.42).



(a) (b) (c)

Figure 3.2: Example factor graphs of (a) the prior distribution, and the joint distribution of the prior and the likelihood when the likelihood describes events associated with (b) different start conditions or (c) obstacle avoidance and different start conditions. States $\boldsymbol{\theta}_i$ are shown as white circles.

3.4.2 Likelihood factors

The factorization of the likelihood (see Section 2.7) is problem-specific and depends on the events being considered. Here, we only consider events involving different start conditions and/or collision avoidance. Figure 3.2(b) shows the joint distribution of the prior and *start-state* likelihood. The posterior inference involves conditioning the prior on a desired start state,

$$p(\mathbf{e}|\boldsymbol{\theta}) \propto f^{\text{start}} = \exp\left\{-\frac{1}{2}\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}_{start}\|^2_{\boldsymbol{\sigma}_{start}}\right\}, \tag{3.13}$$

where a very small covariance $\boldsymbol{\sigma}_{start}$ signifies the certainty of finding a solution that starts from a desired start state $\boldsymbol{\theta}_{start}$. Figure 3.2(c) shows the joint distribution with an additional *collision-free* likelihood. The posterior and associated likelihood are then defined as,

$$p(\mathbf{e}|\boldsymbol{\theta}) \propto f^{\text{start}} f^{\text{obs}} = f^{\text{start}}(\boldsymbol{\theta}_0) \prod_{i=1}^{N} f_i^{\text{obs}}(\boldsymbol{\theta}_i), \quad f_i^{\text{obs}}(\boldsymbol{\theta}_i) = \exp\left\{-\frac{1}{2} \parallel \mathbf{h}(\boldsymbol{\theta}_i) \parallel^2_{\boldsymbol{\sigma}_{obs}}\right\}, \tag{3.14}$$

where f_i^{obs} are unary obstacle factors. The collision for any state is evaluated with a pre-computed signed distance field, a cost function $\mathbf{h}$, and a hyperparameter $\boldsymbol{\sigma}_{obs}$ that balances the weight on collision avoidance versus staying close to the prior. This was also used in GPMP2 for collision avoidance during motion planning. It is worth noting that, due to this generic formulation, the learned skills can be reproduced in any new environment with never-before-seen obstacles as long as a signed distance field is calculated beforehand.

3.4.3 Skill reproduction

Finally, for efficient MAP inference, we take the negative log of the posterior distribution $p(\boldsymbol{\theta}|\mathbf{e}) \propto p_{\boldsymbol{x}}(\boldsymbol{\theta})p(\mathbf{e}|\boldsymbol{\theta})$ using the combined prior Eq. (3.5),

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu}^{\boldsymbol{\theta}} \parallel^2_{\boldsymbol{\mathcal{K}}^{\boldsymbol{\theta}}} + \frac{1}{2} \parallel \mathbf{C}(\boldsymbol{\theta}) - \boldsymbol{\mu}^{\boldsymbol{x}} \parallel^2_{\boldsymbol{\mathcal{K}}^{\boldsymbol{x}}} + \frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}; \boldsymbol{e}) \parallel^2_{\boldsymbol{\Sigma}}, \right\} \tag{3.15}$$

Thus, giving a nonlinear least squares optimization based formulation for the inference problem. The factor graph allows us to compactly organize the computation, with optimization performed using Gauss-Newton or Levenberg-Marquardt. Combining the structure exploiting inference and the quadratic convergence rates of the optimization, make this approach computationally efficient. The computational complexity is directly related to how well the distributions factorize, and since only unary or binary factors are present, the problem is extremely sparse and thus very efficient to solve.

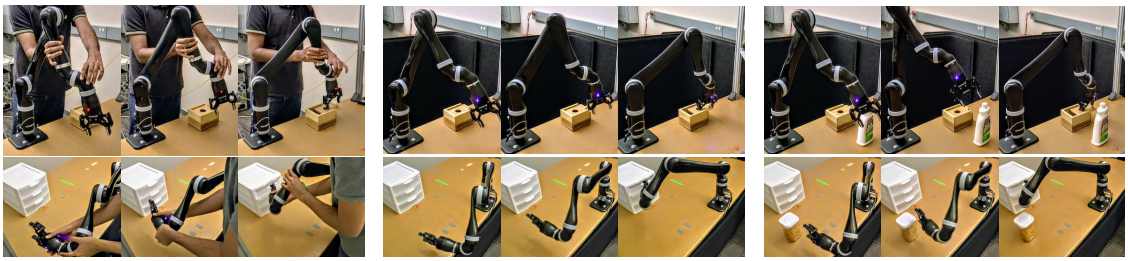
3.5 Evaluation

We implemented CLAMP[1] using the GPMP2 C++ library and tested it on manipulation problems. For skill learning, we considered workspace state $\boldsymbol{x}(t)$ as composed of end-effector 3D position and linear velocity, which proved sufficient for the experiments[2] considered here. We considered joint positions and velocities for the configuration space state $\boldsymbol{\theta}(t)$, and we employed the constant velocity prior for $f^{\text{gp},\boldsymbol{\theta}}$, encouraging smoothness in joint accelerations.

We validated our method on three skills including, *box-opening*, *drawer-opening* and *picking*. All skills were executed on a Kinova JACO[2] 6-DOF arm. For each skill, we provided multiple demonstrations with different initial end-effector states (varying initial position, zero initial velocity) through kinesthetic teaching [96]. The end-effector positions

[1] Implementation available at `https://github.com/GT-RAIL/clamp`

[2] A video of experiments is available at `https://youtu.be/DDs_ZxsN0Ek`



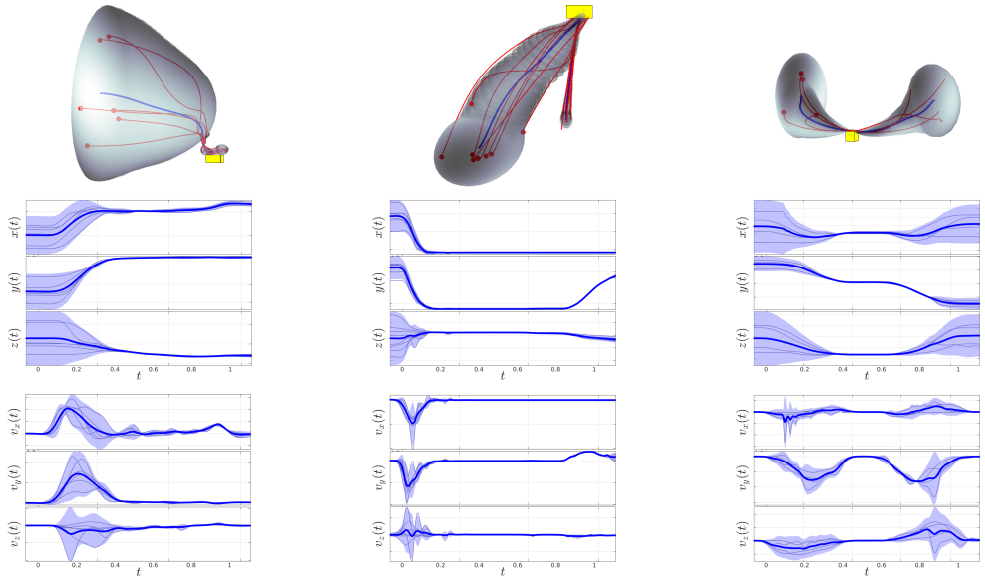
(a) Kinesthetic demonstration (b) Reproduction (c) Reproduction with obstacle

Figure 3.3: Demonstration and reproduction of *box-opening* (top) and *drawer-opening* (bottom).

over time were recorded and the trajectories were temporally aligned using dynamic time warping [114]. The corresponding end-effector linear velocities were estimated by fitting a cubic spline and differentiating with respect to time. Figure 3.4 shows the learned prior distributions i.e. the skill models.

For the *box-opening* skill, each demonstration is composed of two primitive actions, reaching and sliding the lid of the box. The sliding part of the skill is more constrained compared to the reaching part. As shown in Figure 3.4(a), the variance in the state variables (i.e. positions and velocities) become much smaller during the sliding portion of the trajectory. For the *drawer-opening* skill, each demonstration involves reaching the drawer handle and pulling it in the direction perpendicular to the drawer body. Like the *box-opening* skill, the second part of the demonstrations are highly restrictive in both positions and velocities to satisfy skill completion, as shown in Figure 3.4(b). Finally, the *picking* skill involves reaching an object from different initial end-effector positions and then placing it at different locations. As shown in Figure 3.4(c), since object location is fixed across all demonstrations, the variance in the position state variable is much smaller in the middle part of the skill. However, compared to the other two skills which deal with articulated object manipulation, the velocity profile is not as critical for the *picking* skill. For all the skills, the prior also encodes the coupling between the state variables. This is a consequence of the underlying LTV-SDE.

Provided the initial state of the robot, the likelihood in Eq. (3.13) was used during

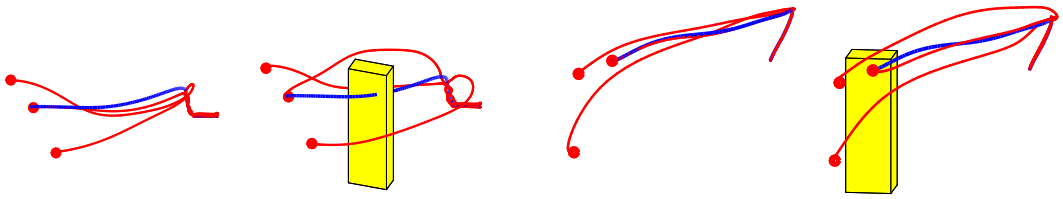


(a) *box-opening* skill (b) *drawer-opening* skill (c) *picking* skill

Figure 3.4: Top: Position workspace priors shown in 3D; Middle: Position workspace priors plotted against time; Bottom: Velcocity workspace priors plotted againt time. The mean is in blue with an envelope showing the 95% confidence. Demonstrations are overlayed.

inference to find MAP trajectories for skill reproduction. For obstacle avoidance, we further incorporated the likelihood in Eq. (3.14). $\boldsymbol{\sigma}_{obs}$ was set manually to enable the desired clearance of the robot from the obstacle. In general, $\boldsymbol{\sigma}_{obs}$ depends on the size of the robot, desired clearance and the environment itself. The MAP trajectories for all scenarios were found using factor graph optimization to solve Eq. (3.15).

Figure 3.5(a) shows the reproduced trajectories for the *box-opening* skill with three different initial robot states. In the left figure, our method was able to adapt the reaching motion as per the initial state. In the presence of a new obstacle (right figure), our method further adapted the reaching part of the skill around the obstacle. The sliding part of the skill is highly constrained, as encoded in the prior and hence does not allow as much adaptability as the reaching part. Figure 3.5(b) shows three MAP trajectories for the *drawer-opening* skill with different starting states, with and without a new obstacle. In this case, like the others, the reaching part of the skill adapts with varying initial states and obstacles, but the highly constrained pulling part remains more-or-less unchanged. Apart from the position



(a) *box-opening* skill (b) *drawer-opening* skill

Figure 3.5: Reproduced position trajectories in red from different initial states. The obstacle is in yellow and the prior position mean is in blue.

trajectories, the direction of motion is also highly constrained in the latter part of these skills, so having velocities in our prior played a crucial role. In all cases, the robot was successful at executing the desired skill. We note that placing the obstacle in front of the object being manipulated would cause failure due to the robot's inability to carry out the required pulling or sliding action. To detect such failure cases, we can use the workspace prior in Eq. (3.6) to provide a demonstration-based success likelihood of the MAP trajectory $\boldsymbol{\theta}^*$, as a pre-execution evaluation step.

3.6 Discussion

We have presented CLAMP, a novel approach which unifies probabilistic LfD and inference-based planning. Within this approach, we learned the skill in a non-parametric and efficient manner, modeling the underlying system as a stochastic dynamical system. In some skills we were able to analyze the skill model as a composition of primitive actions governed by their variability (i.e. variance in the prior). Therefore, we see that learning the skills though such an abstraction of factors on a graph allows for interpretability of the underlying skill. Such models could be closely analyzed and then updated or relearned in situations where the extracted behavior differs from the requirements or expectations of the user. Efficient inference and generalized skill reproduction is carried out by fast numerical optimization over factor graphs. Using this approach, we generate trajectories that are optimal with respect to the learned skill (i.e. the trajectory prior) and feasible with respect to the reproduction scenario composed of various events (i.e. the likelihood). We perform MAP

inference on the factor graphs with likelihoods constructed for satisfying the initial state of the robot, and obstacle avoidance. The structured construction of our framework naturally allows for easy generalization such that appropriate learned skill priors can be conditioned on scenario specific information encoded by the likelihood. We have also extended this framework to allow incremental learning of the skill, and to handle demonstrations obtained from cluttered environments, where they may have been influenced by the presence of objects irrelevant to the skill [89]. The structured compartmentalization makes constraints like safety easy to deal with even during learning (for example during incremental skill learning), as they are handled by the likelihood. Although in our current implementation, we consider robot trajectories to be comprised of positions and velocities and the events to be made up of robot's current initial state and obstacle clearance, our approach allows incorporation of further higher-order dynamics or event likelihoods.

CHAPTER 4

DIFFERENTIABLE INFERENCE-BASED PLANNING

4.1 Introduction and related work

In this chapter, we present another technique that incorporates learning in to the framework presented in Chapter 2 by leveraging the factor graph structure. In contrast to CLAMP in the previous chapter however, here we exploit more modern machine learning tools and also explore learning from algorithmic experts instead of human guided demonstrations.

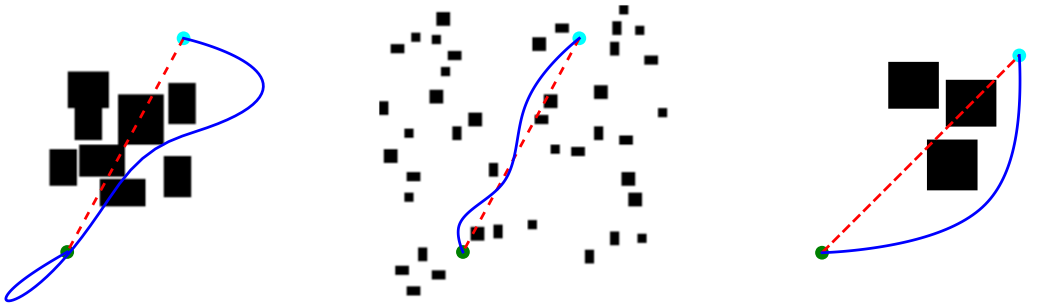
Popular state-of-the-art sampling [4, 5] and optimization [9, 45] based planning approaches have complementary practical trade-offs. Sampling based methods are more well suited for problems with tight navigation constraints like narrow hallways and mazes, but are computationally expensive for high dimensional systems and generally need post processing to get smooth solutions. On the other hand, trajectory optimization methods are fast and well suited to tackle reaching style problems on manipulators. However, these suffer from local minima problems that are usually the consequence of parameters in the objective function.

Learning can be employed in an effective manner to utilize past experience or computation in a manner that can accelerates or augments the planning process [115, 116] as evident from Section 3. With the advent of modern accessible machine learning techniques, there is a growing interest to use deep learning for planning. End-to-end networks have been trained to perform value iteration [117] and also learn rewards and transitions probabilities. But, it is hard to scale as the action space needs to be discretized. [118] learns a latent space embedding and a dynamics model in that space suitable for planning by gradient descent within a goal directed policy. Such approaches have shown that learning to plan is a promising research direction, but leave much to be desired. In general, we want to

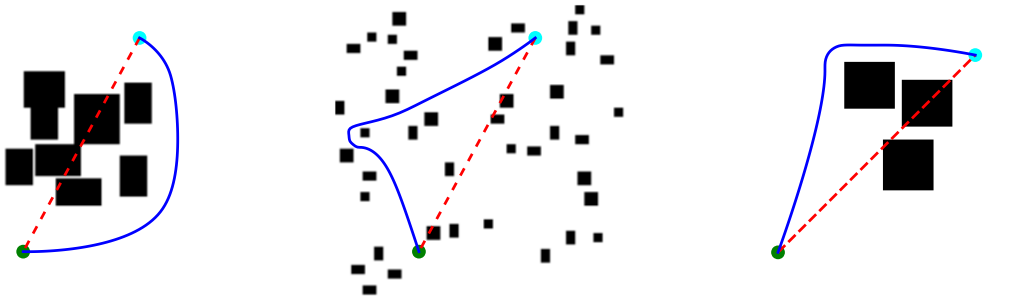
incorporate learning in a manner that is capable of handling planning considerations like specifying task objectives, incorporating domain knowledge and constraints, and handling uncertainty. Current literature falls short on many of these fronts.

Recent works in structured learning techniques offer avenues towards achieving this goal. Various methods have focused on incorporating optimization within the networks. For example, [119] implicitly learns to perform nonlinear least squares optimization by learning an RNN that predicts its update steps, [28] learns to perform gradient descent, and [30] utilizes a ODE solver within the network. Other methods like [29] learn a sequential quadratic program as a layer in the network while is extended to solve model predictive control [120]. [31] learns structured dynamics model for reactive visuomotor control. Taking inspiration from this body of work, in this chapter we present a differentiable inference-based motion planning approach.

Our approach rewrites GPMP2 (see Chapter 2), the inference-based planner, as a fully differentiable computational graph such that we can learn the parameters for its objective function from data. As a reminder, GPMP2 perform planning by constructing the planing problem as inference on a factor graph and finds solutions by solving a nonlinear least squares optimization function, where the inverse covariances in the factors show up as weights in the objective. We will discuss GPMP2's sensitiveness to objective parameters (i.e. factor covariances) and see how such a learning strategy will help in improving GPMP2's general performance. This differentiable version can be trained from experts (too slow to use in practice) to predict covariances that are time and space varying, in contrast to fixed covariances as used in the vanilla approach, and also mitigates the need for hand tuning. This form of structured learning offers interpretability and allows us to incorporate other planning constraints. We perform several experiments in simulated 2D environments comparing against vanilla GPMP2 to showcase the benefits of our approach, differentiable GPMP2 (dGPMP2) [121].



(a) $\sigma_{obs} = 0.1, \mathbf{Q}_C = 0.5 \times I$ (c) $\sigma_{obs} = 0.1, \mathbf{Q}_C = 0.5 \times I$ (e) $\sigma_{obs} = 0.03, \mathbf{Q}_C = I$



(b) $\sigma_{obs} = 0.01, \mathbf{Q}_C = 0.5 \times I$ (d) $\sigma_{obs} = 0.01, \mathbf{Q}_C = 0.5 \times I$ (f) $\sigma_{obs} = 0.01, \mathbf{Q}_C = I$

Figure 4.1: (a)-(b) `tarpit` dataset (robot radius = 0.4m, safety distance = 0.4m). For the same $\mathbf{Q}_C$, a smaller σ_{obs} is required to encourage the planner to navigate around obstacles. (c)-(d) `forest` dataset (robot radius = 0.2m, safety distance = 0.2m). For the same $\mathbf{Q}_C$, a larger σ_{obs} is required to focus on finding solutions near the straight line trajectory. (e)-(f) `multi_obs` dataset (robot radius = 0.4m, safety distance = 0.4m) A small change in obstacle covariance can lead to significant changes in the trajectory. In all figures, the red dashed trajectories are the initializations and the blue trajectories are the optimized solutions.

4.2 Sensitivity to objective function parameters

The performance of GPMP2 is dependent on the values of $\mathbf{Q}_C$ (the parameter that governs the covariance of the GP prior) and $\mathbf{\Sigma}$ (the covariance of the likelihood) as per its objective function (see Section 2.9.2). For example, for collision avoidance, the distribution of obstacles in the environment affects what relative settings of $\mathbf{Q}_C$ and obstacle covariance σ_{obs} (such that $\mathbf{\Sigma} = \sigma_{obs}^2 \times \mathbf{I}$) will be effective in solving the planning problem.

Different datasets require different relative settings of parameters. Due to the nonlinear interactions between these parameters it might not be possible to find a fixed setting that will always work, and in practice it can be a tedious task to find a setting that works for

many different environments. For example, in environments like the one in Figure 4.1a-4.1b, where the planner needs to find a trajectory that goes around the cluster of obstacles, a small obstacle covariance is required to make the planner navigate around the "tarpit." But, at the same time, if a large dynamics covariance is used, it might try to squeeze in between obstacles where the cost can have a local minima. So a smaller dynamics covariance is needed as well. Another example is shown in Figure 4.1c-4.1d with dispersed obstacles near the start and goal. Here an entirely different setting of covariances is effective. Since obstacles are small and diffused, solutions can generally be found close to the straight line initialization. A smaller dynamics covariance helps with that. Also, the start and goal can be very near obstacles which means that a small obstacle covariance might lead to solutions that violate the start and goal constraints. Having a smaller obstacle covariance can also lead to trajectories that are very long and convoluted as they try to stay far away from obstacles.

Small changes in parameters can lead to trajectories lying in different homotopy classes. For example, Figure 4.1e-4.1f illustrates how even minor changes in the obstacle covariance can lead to significant changes in the resulting trajectories. This makes tuning covariances harder, as the effects are further aggravated over large datasets with diverse environments leading to inconsistent results.

With sufficient domain expertise, the parameters can be hand-tuned. However, this process can be very inefficient and becomes increasingly hard for problems in higher dimensions or when complex constraints are involved. An ideal setup would be to have an algorithm that can predict appropriate parameters automatically for each problem. Therefore, in this work, we rebuild the GPMP2 algorithm as a fully differentiable computational graph, such that these parameters can be specified by deep neural networks which can be trained end-to-end from data. When deployed, our differentiable GPMP2 approach (dGPMP2) can then automatically select its own parameters given a particular motion planning problem.

4.3 A computational graph for planning

In this section, we first explain how GPMP2 can be interpreted as a differentiable computation graph. Then, we explain how learning can be incorporated in the framework and finally, we show how the entire system can be trained end-to-end from data.

Our architecture consists of two main components: a planning module $\mathbf{P}$ that is differentiable but has no learnable parameters and a trainable module $\mathbf{W}$ that can be implemented using a differentiable function approximator such as a neural network as shown in Figure 4.2. As discussed in Section 4.2, GPMP2 performs trajectory optimization via MAP inference on a factor graph by solving an iterative nonlinear optimization, where at any iteration the factor graph is linearized at the current estimate of the trajectory to produce the linear system in Eq. (2.52) and an update step is computed by solving that linear system. At a high level, our planning module $\mathbf{P}$ implements this update step as a computational graph. The trainable module $\mathbf{W}$ is then set up to parameterize some desired planning parameters and outputs these as $\boldsymbol{\phi}_L$ at every iteration. These parameters correspond to factor covariances used by $\mathbf{P}$ to construct the linearized factor graph. Additionally, $\mathbf{P}$ takes as input a set of fixed planning parameters $\boldsymbol{\phi}_F$ to allow parameters that can be user-specified and are not being learned, for example, obstacle safety distance and covariances of constraint factors like start, goal, and velocity. The key insight is that since solving Eq. (2.52) involves only matrix operations, we can easily differentiate through it using standard autograd tools [122] and thus train $\mathbf{W}$ in an end-to-end fashion from data.

Similar to GPMP2, during the forward pass, dGPMP2 iteratively optimizes the trajectory where at the i^{th} iteration, the planning module $\mathbf{P}$ takes the current estimate of the trajectory $\boldsymbol{\theta}^i$ and planning parameters $\boldsymbol{\phi}_L$ and $\boldsymbol{\phi}_F$ as inputs (where $\boldsymbol{\phi}_L$ is the output of the trainable module $\mathbf{W}$ and ϕ_F are user-defined and fixed) and produces the next estimate $\boldsymbol{\theta}^{i+1}$ as shown in Figure 4.2. The new estimate then becomes the input for the next iteration. This process continues until $\boldsymbol{\theta}^{i+1}$ passes a specified convergence check or a maximum

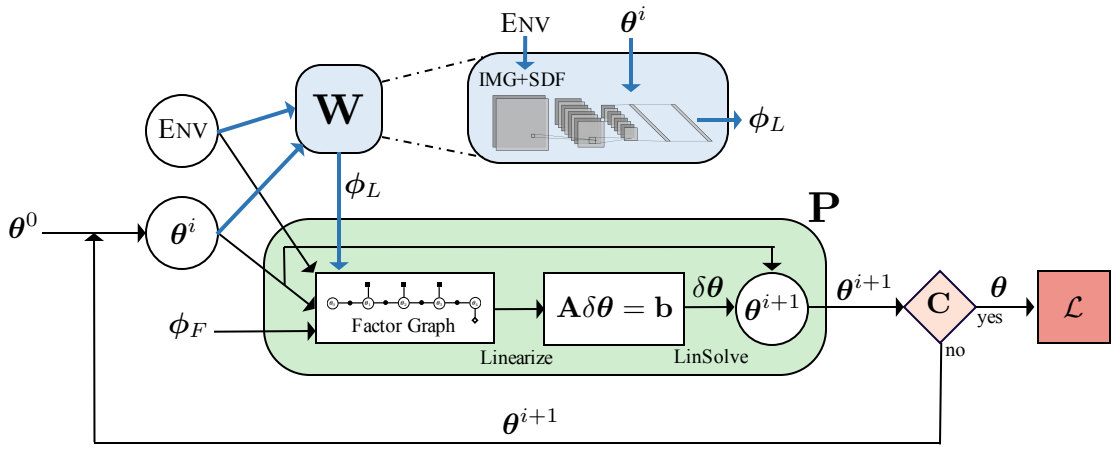


Figure 4.2: The computational graph of dGPMP2 where $\boldsymbol{\phi}_F$ represents some user defined planning parameters that are fixed and $\boldsymbol{\phi}_L$ represents the learned planning parameters. See text for details.

of T iterations and the optimization terminates. At the end of the optimization, we roll out a complete differentiable computation graph for the motion planner.

Notation: $\boldsymbol{\theta}^i$ refers to the trajectory estimate at the i^{th} iteration of the optimization that goes from $1, \ldots, T$ and $\boldsymbol{\theta}_i$ is the i^{th} state along the trajectory that goes from $1, \ldots, N$.

The planning module: $\boldsymbol{\theta}^i$ is fed into the planning module along with a signed distance field of the environment and additional planning parameters ($\boldsymbol{\phi}_F$ and $\boldsymbol{\phi}_L$) such as factor covariances, safety distance, robot kinematics, start-goal constraints, and other task related constraints. These inputs are used to construct the linear system in Eq. (2.52) corresponding to the linearized factor graph of the planning problem. Similar to standard GPMP2, constraints are implemented as factors with fixed small covariances and the likelihood function for obstacle avoidance is the hinge loss function (see Section 4.4) with covariance $\boldsymbol{\Sigma}$. The trajectory update $\delta\boldsymbol{\theta}^i$ is then computed by solving this linear system, using Cholesky decomposition of the normal equations [41, 39], and the new trajectory $\boldsymbol{\theta}^{i+1}$ is computed using a Gauss-Newton step. Since the above procedure involves only matrix operations it is fully differentiable and allows computing gradients in the backwards pass with respect to $\boldsymbol{\theta}^i$, GP covariance $\boldsymbol{\mathcal{K}}$ and likelihood function covariance $\boldsymbol{\Sigma}$.

The trainable module: The trainable module $\mathbf{W}$ outputs planning parameters $\boldsymbol{\phi}_L$. These correspond to covariances of factors in Eq. (2.52) that we wish to learn from data. In practice, we can choose to learn the GP covariance $\boldsymbol{\mathcal{K}}$, the likelihood covariance $\boldsymbol{\Sigma}$, or both. Additionally, this approach allows us to learn individual covariances for different states along the trajectory $[\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N]$ and different iterations of the optimization thus offering much more expressiveness than a single hand-tuned covariance. We implement $\mathbf{W}$ as a feed-forward convolutional neural network that takes as input the bitmap image of the environment and signed distance field and outputs a parameter vector $\boldsymbol{\phi}_L^i$ at every iteration i. Note that, given our architecture, $\mathbf{W}$ can be customized as per individual needs based on problem requirements or parameters chosen to be learned.

After a forward pass, we roll out a fully differentiable computation graph that outputs a sequence of trajectories $\{\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^T\}$. Then we evaluate a loss function on this sequence and backpropagate that loss to update the parameters of $\mathbf{W}$ such that it produces parameters ϕ_L that allow us to optimize for better quality trajectories on the dataset as measured by the loss. We explain our loss function and the training procedure in detail below.

Imitation loss: Consider the availability of expert demonstrations for a planning problem. These may be provided by an asymptotically optimal (but slow) motion planner [123] or by human demonstration [88]. dGPMP2 can be trained to produce similar trajectories by minimizing an error metric between the demonstrations and learner's output with

$$\mathcal{L}_{imitation} = ||\boldsymbol{\theta}^e - \boldsymbol{\theta}||_2^2 \tag{4.1}$$

where $\boldsymbol{\theta}^e$ is the expert's demonstrated trajectory and the metric is the L2 norm.

Task loss: Naively trying to match the expert can be problematic for a motion planner. For example, when equally good paths lie in different homotopy classes, the learner may land in a different one than the expert. In this case, penalizing for not matching the expert may be excessively conservative. If using human demonstrations as an expert, a

realizability gap can arise when the planner has different constraints as compared with the human. Thus, we use an external task loss as a regularizer that encourages smoothness and obstacle avoidance, while respecting start and goal constraints, as is often used in motion planning [44]:

$$\mathcal{L}_{plan} = \mathcal{F}_{smooth} + \lambda \times \mathcal{F}_{obs}, \tag{4.2}$$

where $\mathcal{F}_{smooth}$ corresponds to the GP prior error and $\mathcal{F}_{obs}$ is the obstacle cost and λ is a user specified parameter.

The overall loss for a single trajectory is, $\mathcal{L} = \mathcal{L}_{imitation} + \mathcal{L}_{plan}$.

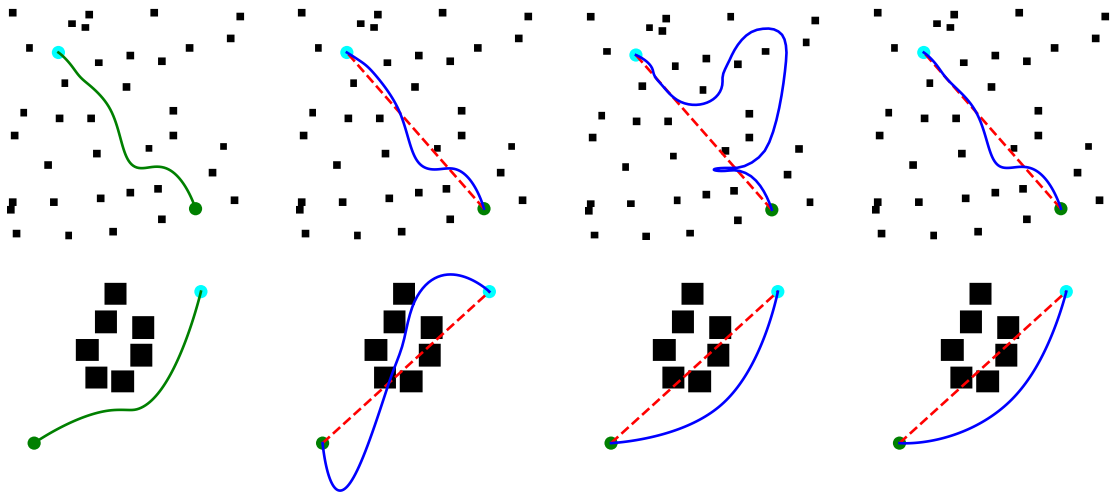
Training: During training we roll out our learner for a fixed number of iterations T and use Backpropagation Through Time (BPTT) [124] on the sum of losses of the intermediate trajectories in order to update the parameters of the trainable module $\mathbf{W}$. Then, the total loss minimized for our learner over a batch of size K is

$$\mathcal{L}_{total} = \frac{1}{K}\frac{1}{T}\sum_{k=1}^{K}\sum_{i=1}^{T}\mathcal{L}^{k,i}. \tag{4.3}$$

4.4 Evaluation

4.4.1 Implementation details

All our experiments and training are performed on a desktop with 8 Intel Core i7-7700K @ 4.20GHz CPUs, 32GB RAM and a 12GB NVIDIA Titan Xp. We consider a 2D point robot in a cluttered environment and planning is done in a state space $\boldsymbol{\theta}_i = [x, y, \dot{x}, \dot{y}]^T$. The robot is represented as a circle with radius r centered on its center of mass and the environment is a binary occupancy grid. A Euclidean signed distance field is computed from the occupancy grid to evaluate distance to obstacles and check collisions. We utilize the same collision likelihood factor as GPMP2 (see Section 2.9.2). In our current experiments, we consider $\boldsymbol{\phi}_L = \sigma_{obs}$ as the learned parameter and $\boldsymbol{\phi}_F = [\mathbf{Q}_C, \epsilon_{safe}, \boldsymbol{\mathcal{K}}_s, \boldsymbol{\mathcal{K}}_g]$ to be fixed i.e we only learn the obstacle covariance and keep the GP covariance fixed. Although, performance of



(a) Expert (b) GPMP2 $\sigma_{obs} = 0.15$ (c) GPMP2 $\sigma_{obs} = 0.01$ (d) dGPMP2

Figure 4.3: Example comparison of (d) dGPMP2 against (b)-(c) GPMP2 (fixed hand tuned covariances) and (a) Expert on `forest` (top row) and `tarpit` (bottom row) datasets. Hand tuned covariances that work well on one distribution of obstacles fail on the other and vice versa. By imitating the expert, dGPMP2 is able to perform consistently across different environment distributions. Green circle is start, cyan is goal, dashed red line is initialization, and $\mathbf{Q}_c = 0.5 \times I$, $r = 0.4m$ for all. Trajectory is in collision if at any state the signed distance between robot center of mass and nearest obstacle is less than or equal to r.

the planner depends on both $\mathbf{Q}_C$ and $\mathbf{\Sigma}$, for our task they trade off against each other and thus we can achieve a similar behavior by varying one relative to the other. Since in our setup the environment changes learning the likelihood covariance $\mathbf{\Sigma}$ is more relevant. For GPMP2, $\mathbf{\Sigma} = \sigma_{obs}^2 \times \mathbf{I}$, while for dGPMP2, $\mathbf{\Sigma} = \text{diag}(\sigma_{obs_1}^2, \ldots, \sigma_{obs_N}^2)$, where any σ_{obs_i} is a function of the current trajectory and the environment.

Loss function: Sampling based asymptotically optimal planning methods such as RRT* [123] are effective in finding good homotopy classes to serve as an initialization for local trajectory optimizers, but can be slow to converge and produce non-smooth solution paths. We use a combination of RRT* and GPMP2 as our expert. Expert trajectories are generated by first running RRT* and are then optimized with GPMP2 to yield smooth solutions. This allows dGPMP2 to learn by utilizing the best combination of local and global planning. We use the loss function defined in Section 4.3 with this expert.

Network architecture: For $\mathbf{W}$ we use a standard feed-forward neural network model consisting of convolutional and fully connected layers. The network consists of 5 convolutional layers with [16, 16, 16, 32, 32] filters respectively, all 3x3 in size. This is followed by two fully connected with [1000, 640] hidden units. We use ReLU activation with batch normalization in all layers and a dropout probability of 0.5 in the fully connected layers. The input to the neural network is a 128x128 bitmap of the environment stacked on top of the euclidean signed distance field of the same dimensions. Training is performed for fixed number of iterations, $T = 10$.

Comparing planners: The convergence for the optimization is based on the following criterion: a tolerance on the relative change in error across iterations $\texttt{tol}(\delta_{error})$, magnitude of update $\texttt{tol}(\delta\boldsymbol{\theta})$, and max iterations T_{max}. On convergence the final trajectory is returned. We report the following metrics on a test set of environments: (i) `success`, percent of problems solved i.e. when a collision free trajectory is found, (ii) average `gp_mse`, mean-squared GP error measuring smoothness and (iii) `collision_intensity`, the average portion of trajectory spent in collision when a collision occurs.

We test our framework on two different planning tasks to demonstrate (i) how learning covariances improves performance and (ii) how the planner's structure allows us to incorporate constraints. We compare against a baseline GPMP2 with hand-tuned parameters.

4.4.2 Learning on complex distributions

In this experiment, we show that if the planner's parameters are fixed, performance can be highly sensitive to distribution of obstacles in the environment. However, if a function can be learned to set the parameters based on the current planning problem, this can help the planner achieve uniformly good performance across different obstacle distributions. We construct a hybrid dataset which is a mixture of two distinct distributions of obstacles as shown in Figure 4.3. The first distribution called `forest` consists of small obstacles scattered around the workspace and the second called `tarpit` contains small number of

larger obstacles clumped together near the center of the workspace. We use a test set of 150 randomly sampled environments from this mixed dataset and further subdivide it into two sets for each of the constituent distributions (roughly equal in proportion). We then hand-tuned parameters for GPMP2 to find the best covariances for the individual distributions and compared them against dGPMP2 on three different test sets: two for the individual distributions and one for a mixed (roughly equal of the two distributions). The results in Table 4.1 show that for GPMP2 the best parameters on one distribution perform poorly on the other distribution in terms of success, although their performances on the mixed dataset are similar. Conversely, dGPMP2 has uniform and consistent performance across both distributions even though it is only trained on the mixed dataset. This demonstrates that dGPMP2 does not require manual tuning or domain knowledge for every distribution of planning environments, but can automatically predict the covariances to use based on the current trajectory and environment as can be seen in Figure 4.3. Additionally, dGPMP2 has the lowest `gp_mse` on the mixed dataset meaning the trajectories produced are still smooth. dGPMP2 also converges in fewer number of iterations than the GPMP2 due to the covariance being more expressive and varying over iterations.

Limitations: Since BPTT is known to have issues with exploding and vanishing gradients for long sequences, we use a small number of iterations ($T = 10$) during training which prevents the learner from sufficiently exploring during training. The network architecture is a simple feed-forward network and does not have any memory and hence the learner does not learn to escape local minima very well. We believe that these issues can be addressed in the future using learning techniques such as Truncated Backpropagation Through Time (TBPTT) [125], recurrent networks such as LSTMs [128], and policy gradient methods [126, 127].

Table 4.1: Comparison of dGPMP2 versus GPMP2 with fixed hand tune covariances. dGPMP2 learns the obstacle covariance σ_{obs} using training set of 5000 environments. $\mathbf{Q}_C = 0.5 \times I$ for all.

		GPMP2		dGPMP2
		$\sigma_{obs} = 0.15$	$\sigma_{obs} = 0.01$	
forest only	success	71.02	52.18	66.67
tarpit only	success	55.56	74.08	68.00
mixed	success	62.67	64.00	67.33
gp_mse		0.002	0.0484	0.0015
num_iters		55.69	86.74	50.00
coll_intensity		0.0464	0.0414	0.0374

Table 4.2: Performance of dGPMP2 with velocity constraints on different combinations of training and testing. Mild constraints are $v_{xmax} = 1.5m/s$, $v_{ymax} = 1.5m/s$, and $time = 15s$, tight constraints are $v_{xmax} = 1.0m/s$, $v_{ymax} = 1.0m/s$, and $time = 10s$ for the same start and goal.

Training condition	Mild	Mild	Tight
Testing condition	Mild	Tight	Tight
success	96	96	98.12
constraint_violation	0.0022	0.104	0.097

4.4.3 Planning with velocity constraints

We show that our learning method can explicitly incorporate planning constraints by including velocity limit factors into the optimization. We use a hinge loss similar to obstacle cost to bound the robot velocity v_x and v_y and set the covariance to a low value, $\boldsymbol{\mathcal{K}}_v = 10^{-4}$, analogous to joint limit factors in [41]. We evaluate the average constraint_violation on a dataset with multiple randomly placed obstacles and study the effect of incorporating constraints during training. Table 4.2 shows a comparison between dGPMP2 trained with mild constraints and tested on problems with mild and tight constraints versus dGPMP2 trained using tight constraints and tested on problems with tight constraints (details in the Table 4.2 caption). We see that, by incorporating tight constraints during training, dGPMP2 can learn to handle tight constraints while avoiding obstacles. This illustrates that dGPMP2 can successfully incorporate constraints within its structure, and that the method can learn to plan while respecting user-defined planning constraints.

4.5 Discussion

We formulated an inference-based motion planner as a differentiable computational graph. Our method learned to predict objective function parameters as part of the differentiable planner and showed competing performance against planning with fixed hand tuned parameters. With the help of the embedded planning structure we are also able to handle constraints. These preliminary results show that this approach is viable and is a promising direction to further investigate the benefits of structured learning as a way to bridge the gap between traditional planning methods and modern machine learning techniques. Our implementation is currently limited to only point robots in 2D environments. However, since the formulation was built on the GPMP2 planner, it can be extended to handle articulated robots in 3D workspaces in a fashion similar to the planner. The factor graph structure embedded within our approach that allows us to handle constraints, can also allow for handling uncertainty via noisy state measurements. Exploration of these ideas are a few directions for future work.

Part III

Reactive Policy Synthesis

CHAPTER 5
RIEMANNIAN POLICIES FOR REACTIVE MOTION

5.1 Introduction

We now switch gears to another common approach to motion generation with reactive policies. They operate over single steps and are locally governed, but can be fast with respect to computation as well adapting to dynamic changes in the environment. Broadly speaking, this is complementary in its strengths and weakness to trajectory optimization style of motion planning that was the focus of the first half of this thesis.

In this chapter, we develop a new reactive motion generation framework that enables globally stable controller design within *intrinsically* non-Euclidean spaces.[1] Non-Euclidean geometries are not often modeled explicitly in robotics, but are nonetheless common in the natural world. One important example is the apparent non-Euclidean behavior of obstacle avoidance. Obstacles become holes in this setting. As a result, straight lines are no longer a reasonable definition of shortest distance—geodesics must, therefore, naturally flow around them. This behavior implies a form of non-Euclidean geometry: the space is naturally curved by the presence of obstacles.

The planning literature has made substantial progress in modeling non-Euclidean task-space behaviors, but at the expense of efficiency and reactivity. Starting with early differential geometric models of obstacle avoidance [129] and building toward modern planning algorithms and optimization techniques [130, 131, 132, 15, 6, 123, 49, 41], these techniques can calculate highly nonlinear trajectories. However, they are often computationally intensive, sensitive to noise, and unresponsive to perturbation. In addition, the internal nonlinearities of robots due to kinematic constraints are sometimes simplified in

[1] Spaces defined by non-constant Riemannian metrics with non-trivial curvature.

the optimization.

At the same time, a separate thread of literature, emphasizing fast reactive control over computationally expensive planning, developed efficient closed-loop control techniques such as Operational Space Control (OSC) [133]. But while these techniques account for internal geometries from the robot's kinematic structure, they assume simple Euclidean geometry in task spaces [134, 135], failing to provide a complete treatment of the external geometries. As a result, obstacle avoidance, e.g., has to rely on *extrinsic* potential functions, leading to undesirable deceleration behavior when the robot is close to the obstacle. If the non-Euclidean geometry can be *intrinsically* considered, then fast obstacle avoidance motion would naturally arise as traveling along the induced geodesic. The need for a holistic solution to motion generation and control has motivated a number of recent system architectures tightly integrating planning and control [136, 85].

We develop a new approach to synthesizing motion policies that can accommodate and leverage the modeling capacity of intrinsically non-Euclidean robotics tasks. Taking inspiration from Geometric Control Theory [137],[2] we design a novel recursive algorithm, RMPflow [138], based on a recently proposed mathematical object for representing nonlinear policies known as the Riemannian Motion Policy (RMP) [139]. This algorithm enables the geometrically consistent fusion of many component policies defined across non-Euclidean task spaces that are related through a tree structure. Such a task-map tree structure embedded within our formulation will be critical to incorporating learning in the following chapters. We show that RMPflow, which generates behavior by calculating how the robot should accelerate, mimics the Recursive Newton-Euler algorithm [140] in structure, but generalizes it beyond rigid-body systems to a broader class of highly-nonlinear transformations and spaces.

In contrast to existing frameworks, our framework naturally models non-Euclidean task spaces with Riemannian metrics that are not only configuration dependent, but also *velocity*

[2] See Appendix B.1 for a discussion of why geometric mechanics and geometric control theory constitute a good starting point.

dependent. This allows RMPflow to consider, e.g., the *direction* a robot travels to define the importance weights in combing policies. For example, an obstacle, despite being close to the robot, can usually be ignored if robot is heading away from it. This new class of policies leads to an extension of Geometric Control Theory, building on a new class of non-physical mechanical systems we call Geometric Dynamical Systems (GDS).

We also show that RMPflow is Lyapunov-stable and coordinate-free. In particular, when using RMPflow, robots can be viewed each as different parameterizations of the same task space, defining a precise notion of behavioral consistency between robots. Additionally, under this framework, the implicit curvature arising from non-constant Riemannian metrics (which may be roughly viewed as position-velocity dependent inertia matrices in OSC) produces nontrivial and intuitive policy contributions that are critical to guaranteeing stability and generalization across embodiments. Our experimental results illustrate how these curvature terms can be impactful in practice, generating nonlinear geodesics that result in curving or orbiting around obstacles. We also demonstrate the utility of our framework with a fully reactive real-world system on multiple dual-arm manipulation problems.

5.2 Motion generation and control

Motion generation and control can be formulated as the problem of transforming curves from the configuration space $\mathcal{C}$ to the task space $\mathcal{T}$. Specifically, let $\mathcal{C}$ be a d-dimensional smooth manifold. A robot's motion can be described as a curve $q : [0, \infty) \to \mathcal{C}$ such that the robot's configuration at time t is a point $q(t) \in \mathcal{C}$. Without loss of generality, suppose $\mathcal{C}$ has a global coordinate $\mathbf{q} : \mathcal{C} \to \mathbb{R}^d$, called the *generalized coordinate*; for short, we would identify the curve q with its coordinate and write $\mathbf{q}(q(t))$ as $\mathbf{q}(t) \in \mathbb{R}^d$. A typical example of the generalized coordinate is the joint angles of a d-DOF (degrees-of-freedom) robot: we denote $\mathbf{q}(t)$ as the joint angles at time t and $\dot{\mathbf{q}}(t)$, $\ddot{\mathbf{q}}(t)$ as the joint velocities and accelerations. To describe the tasks, we consider another manifold $\mathcal{T}$, the task space, which is related to the configuration space $\mathcal{C}$ through a smooth *task map* $\psi : \mathcal{C} \to \mathcal{T}$. The task

space $\mathcal{T}$ can be the end-effector position/orientation [133, 141], or more generally can be a space that describes whole-body robot motion, e.g., in simultaneous tracking and collision avoidance [142, 143]. Thus, the goal of motion generation and control is to design the curve q so that the transformed curve $\psi \circ q$ exhibits desired behaviors on the task space $\mathcal{T}$.

Notation For clarity, we use boldface to distinguish the coordinate-dependent representations from abstract objects; e.g. we write $q(t) \in \mathcal{C}$ and $\mathbf{q}(t) \in \mathbb{R}^d$. In addition, we will often omit the time- and input-dependency of objects unless necessary; e.g. we may write $q \in \mathcal{C}$ and $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. For derivatives, we use both symbols ∇ and ∂, with a transpose relationship: for $\mathbf{x} \in \mathbb{R}^m$ and a differential map $\mathbf{y} : \mathbb{R}^m \to \mathbb{R}^n$, we write $\nabla_{\mathbf{x}}\mathbf{y}(\mathbf{x}) = \partial_{\mathbf{x}}\mathbf{y}(\mathbf{x})^\top \in \mathbb{R}^{m\times n}$. For a matrix $\mathbf{M} \in \mathbb{R}^{m\times m}$, we denote $\mathbf{m}_i = (\mathbf{M})_i$ as its ith column and $M_{ij} = (\mathbf{M})_{ij}$ as its (i, j) element. To compose a matrix, we use $(\cdot)^{\cdot}_{\cdot}$ for vertical (or matrix) concatenation and $[\cdot]^{\cdot}_{\cdot}$ for horizontal concatenation. For example, we write $\mathbf{M} = [\mathbf{m}_i]_{i=1}^m = (M_{ij})_{i,j=1}^m$ and $\mathbf{M}^\top = (\mathbf{m}_i^\top)_{i=1}^m = (M_{ji})_{i,j=1}^m$. We use $\mathbb{R}_+^{m\times m}$ and $\mathbb{R}_{++}^{m\times m}$ to denote the symmetric, positive semi-definite/definite matrices, respectively.

We model motion as a second-order differential equation of $\ddot{\mathbf{q}} = \pi(\mathbf{q}, \dot{\mathbf{q}})$, where we call π a *motion policy* and $(\mathbf{q}, \dot{\mathbf{q}})$ the *state*. We assume the system has been feedback linearized [144], which is a common technique used to control fully actuated nonlinear systems (serial manipulators). The idea is to build a controller that cancels the nonlinearities in the dynamics so that the acceleration becomes the new control and the system becomes a double integrator (i.e. linear dynamics). In practice, once the policy generation provides an acceleration based control we can execute that policy on the physical system in two particular ways: (i) apply inverse dynamics to calculate the true control (joint torques) to be executed, or (ii) integrate the acceleration to get a position and velocity trajectory to be tracked by a PD controller. The performance of the former method relies on access to an accurate dynamics model, while that of the latter depends on well tuned gains of the PD controller. Since in our experiments we deal with fully actuated manipulators performing non-high-speed tasks, the second method proved sufficient to control our systems.

5.3 Related work

In contrast to an open-loop trajectory, which forms the basis of many motion planners, a motion policy expresses the entire continuous collection of its integral trajectories[3] and therefore is robust to perturbations. Motion policies can model many adaptive behaviors, such as reactive obstacle avoidance [145, 136] or responses driven by planned Q-functions [146], and their second-order formulation enables rich behavior that cannot be realized by the velocity-based approach [147].

The geometry of motion has been considered by many planning and control algorithms. Geometrical modeling of task spaces is used in topological motion planning [131], and motion optimization has leveraged Hessian to exploit the natural geometry of costs [9, 15, 36, 40]. Ratliff et al. [130], e.g., use the workspace geometry inside a Gauss-Newton optimizer and generate natural obstacle-avoiding reaching motion through traveling along geodesics of curved spaces. Geometry-aware motion policies were also developed in parallel in controls. OSC is the best example [133]. Unlike the planning approaches, OSC focuses on the internal geometry of the robot and considers only simple task-space geometry. It reshapes the workspace dynamics into a simple spring-mass-damper system with a constant inertia matrix, enforcing a form of Euclidean geometry in the task space. Variants of OSC have been proposed to consider different metrics [148, 134, 143], task hierarchies [142, 149], and non-stationary inputs [150].

While these algorithms have led to many advances, we argue that their isolated focus on either the internal or the external geometry limits the performance. The planning approach fails to consider reactive dynamic behavior; the control approach cannot model the effects of velocity dependent metrics, which are critical to generating sensible obstacle avoidance motions, as discussed in the introduction. While the benefits of velocity dependent metrics was recently explored using RMPs [139], a systematic understanding is still an open question.

[3] An integral curve is the trajectory starting from a particular state.

5.4 From operational space control to geometric control

We set the stage for our development of RMPflow and geometric dynamical systems (GDSs) in Sections 5.5-5.7 by first giving some background on the key tools central to this work. Specifically, we give a tutorial on the controller design technique known as energy shaping and the geometric formulation of classical mechanics, both of which are commonly less familiar to robotics researchers. Then we will show that geometric control [137], which to a great extent developed independently of operational space control within a distinct community, nicely summarizes these two ideas and leads to techniques of leveraging energy shaping within the context of geometric mechanics.

This section targets at readers more familiar with operational space control and introduces many of the relevant ideas in a way that we hope is more accessible than the traditional exposition which assumes a background in differential geometry. We begin with a review of classical operational space control wherein tasks are represented as hard constraints on the mechanical system, and then show how energy shaping and the geometric mechanics formalism enable us to easily develop provably stable operational space controllers that simultaneously trade off many tasks. The material presented in this section primarily rehashes existing techniques from a perhaps unfamiliar community, restating them in a way that should be more natural to researchers familiar with operational space control. We end with a discussion of the limitations of these geometric control techniques that we will address with RMPflow and GDSs.

Energy shaping and classical operational space control

Energy shaping is a controller design technique, wherein the designer first configures a virtual mechanical system by shaping its kinetic and potential energies to exhibit a certain behavior, and then drive the robot's dynamics to mimic that virtual system. This scheme

overall generates a control law with a well-defined Lyapunov function, given as the virtual system's total energy, and therefore has provable stability.

For instance, the earliest form of operational space control [133] formulates a virtual system that places all mass at the end-effector. Behavior is then shaped by applying potential energy functions (regulated by a damper) to that virtual mass (e.g. by connecting the end-effector to a target using a virtual (damped) spring). Controlling the system to behave like that virtual system then generates a control law whose stability is governed by the total energy of that virtual point-mass system. In this context, the choice of virtual mechanical system (the point end-effector mass) is a form of *kinetic energy shaping*, and the subsequent choice of potential energy applied to that point end-effector mass is known as *potential energy shaping*. This particular pattern of task-centric kinetic and potential energy shaping, is common throughout the operational space control literature.

A similar theme can be found in [134]. Here the virtual mechanical systems are designed by constraining an existing mechanical system (e.g. the robot's original dynamics) to satisfy task constraints. This is achieved by designing controllers around a generalized form of Gauss's principle of least constraint [151], so that virtual mechanical systems would behave in a sense as similarly as possible to the true robotic mechanical system while realizing the required task accelerations. In other words, the energies of the original mechanical system are reshaped to that given by the task constraints.

In essence, these early examples above are based on the idea that faithful execution of the task enables a simplified stability analysis as long as the task space behavior is itself well-understood and stable. This style of simplified analysis and the controller design has been successful in practice. Nonetheless, it imposes a limitation that the controllers cannot have more tasks than the number of DOF in the system. This becomes particularly problematic when one wishes to introduce more complex auxiliary behaviors, such as collision avoidance where the number of tasks might scale with the number of obstacles and the number of control points on the robot's body.

The rest of this section is dedicated to unify and then generalize these ideas through the lens of geometric mechanics, so that we can use operational space control to handle these more complex settings of many competing tasks, by using nuanced weighted priorities that might change as a function of the robot's configuration. However, we will eventually see that even this is still not quite sufficient for representing many common behaviors. The insights into sources of these limitation learned in this section are the motivation of our more in-depth subsequent development of RMPflow and geometric dynamical systems (GDSs).

A simple first step towards weighted priorities

This section leverages Gauss's principle of least constraint (different from the techniques mentioned briefly above [134]) to illustrate the concept of energy shaping, which will be used more abstractly below to derive a simple technique for combining multiple task-space policies.

Gauss's Principle: Gauss's principle of least constraint states that a nonlinearly constrained collection of particles evolves in a way that is most similar to its unconstrained evolution, as long as this notion of similarity is measured using the inertia-weighted squared error [135]. For example, let us consider N particles: $\mathbf{x}_i \in \mathbb{R}^3$ with respective (positive) inertia $m_i \in \mathbb{R}_+$, for $i = 1, \ldots, N$. Then the acceleration $\ddot{\mathbf{x}}_i$ of the i^{th} particle under Gauss's principle can be written as

$$\ddot{\mathbf{x}} = \operatorname{argmin}_{\ddot{\mathbf{x}}' \in \mathcal{A}} \tfrac{1}{2} \|\ddot{\mathbf{x}}^d - \ddot{\mathbf{x}}'\|^2_{\mathbf{M}} \tag{5.1}$$

where $\mathcal{A}$ denotes the set of admissible constrained accelerations. To simplify the notation, we've stacked[4] the particle accelerations into a vector $\ddot{\mathbf{x}} = (\ddot{\mathbf{x}}_1; \ldots; \ddot{\mathbf{x}}_N)$ and construct a diagonal matrix $\mathbf{M} = \operatorname{diag}(m_1\mathbf{I}, \ldots, m_N\mathbf{I})$, where $\mathbf{I} \in \mathbb{R}^{3\times 3}$ is the identity matrix.

[4]We use the notation $\mathbf{v} = (\mathbf{v}_1; \mathbf{v}_2; , \ldots, \mathbf{v}_N)$ to denote stacking of vectors $\mathbf{v}_i \in \mathbb{R}^3$ into a single vector $\mathbf{v} \in \mathbb{R}^{3N}$.

Kinematic Control-Point Design: Let us use the above idea to design a robot controller. If we define many kinematic control points $\mathbf{x}_i \in \mathbb{R}^3, i = 1, \ldots, N$ distributed across the robot's body and calculate a desired acceleration at those points $\ddot{\mathbf{x}}_i^d$, a sensible way to trade off these different accelerations is through the following quadratic program (QP):

$$\min_{\ddot{\mathbf{x}}_i} \sum_{i=1}^{N} \frac{m_i}{2} \|\ddot{\mathbf{x}}_i^d - \ddot{\mathbf{x}}_i\|^2 \quad \text{s.t. } \ddot{\mathbf{x}}_i = \mathbf{J}_i \ddot{\mathbf{q}} + \dot{\mathbf{J}}_i \dot{\mathbf{q}}, \tag{5.2}$$

where $m_i > 0$ is the importance weight, $\mathbf{x}_i = \psi_i(\mathbf{q})$ is the forward kinematics map to the i^{th} control point and $\mathbf{J}_i = \partial_{\mathbf{q}_i} \psi_i$ is its Jacobian. This QP states that the system should follow the desired accelerations as well as possible, with (constant) tradeoff priorities m_i in the event the tasks cannot be achieved exactly, subject to the physical kinematic constraints on how each control point can accelerate.

Comparing this QP to that given by Gauss's principle in Eq. (5.1), one can immediately see that its solution gives the constrained dynamics of a mechanical system defined by N point particles of mass m_i with unconstrained accelerations $\ddot{\mathbf{x}}_i^d$ and acceleration constraints $\ddot{\mathbf{x}}_i = \mathbf{J}_i \ddot{\mathbf{q}} + \dot{\mathbf{J}}_i \dot{\mathbf{q}}$. In particular, if $\ddot{\mathbf{x}}_i^d = -m_i^{-1} \nabla \phi_i - \beta_i \dot{\mathbf{x}}_i$ for some non-negative potential function ϕ_i and constant β_i, we arrive at a mechanical system with total energy $\sum_{i=1}^{N} \left(\frac{m_i}{2} \|\dot{\mathbf{x}}_i\|^2 + \phi_i(\mathbf{x}_i) \right)$. Controlling the robot system according to desired accelerations $\ddot{\mathbf{q}}^*$ given by solving Eq. (5.2) ensures that this total energy dissipates at a rate defined by the collective non-negative dissipation terms $\sum_i m_i \beta_i \|\dot{\mathbf{x}}_i\|^2$. This total energy, therefore, acts as a Lyapunov function.

This kinematic control-point design technique utilizes now more explicitly the methodology of energy shaping. In this case, we use Gauss's principle to design a virtual mechanical system that strategically distributes point masses throughout the robot's body at key control points (kinetic energy shaping). We then apply (damped) virtual potential functions to those masses to generate behavior (potential energy shaping). In combination, we see that the resulting system can be viewed as a QP which tries to achieve all tasks simul-

taneously as well as it can. Since exact replication of all tasks is impossible, the QP uses the mass values as relative priorities to define how the system should trade off task errors when necessary.

Abstract task spaces: simplified geometric mechanics

The controller we just described demonstrates the core concept around energy shaping, but is limited by requiring that tasks be designed specifically on kinematic control-points distributed *physically* across the robot's body. Usually task spaces are often more abstract than that, and most generally we consider any task space that can be described as a nonlinear map from the configuration space.

This abstraction is common in trajectory optimization. For instance, [15] describes some abstract topological spaces for behavior creation which enable behaviors such as wrapping an arm around a pole and unwrapping it, and abstract models of workspace geometry are represented in [130, 152] by designing high-dimensional task spaces consisting of stacked (proximity weighted) local coordinate representations of surrounding obstacles conveying how obstacles shape the space around them. Likewise, similar abstract spaces are highly relevant for describing common objectives in operational space control problems. Specifically, spaces of interest include one-dimensional spaces encoding distances to barrier constraints such as joint limits and obstacles, distances to targets, spaces of quaternions, and the joint space itself; all of these are more abstract than specific kinematic control-points. In order to generalize these ideas to abstract task spaces we need better tools. Below we show the insights from geometric mechanics and geometric control theory [137] provide the generalization that we need.

Quick review of Lagrangian mechanics: Lagrangian mechanics is a reformulation of classical mechanics that derives the equations of motion by applying the Euler-Lagrange equation the Lagrangian of the mechanical system [153]. Specifically, given a generalized inertia matrix $\mathbf{M}(\mathbf{q})$ and a potential function $\Phi(\mathbf{q})$, the Lagrangian is the difference between

kinetic and potential energies:

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = \tfrac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - \Phi(\mathbf{q}). \tag{5.3}$$

The Euler-Lagrange equation is given by

$$\tfrac{d}{dt}\partial_{\dot{\mathbf{q}}}\mathcal{L} - \partial_{\mathbf{q}}\mathcal{L} = \tau_{\text{ext}} \tag{5.4}$$

where τ_{ext} is the external force applied on the system. Applying Eq. (5.4) to the Lagrangian Eq. (5.3) gives the equations of motion,

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \nabla\Phi(\mathbf{q}) = \tau_{\text{ext}}, \tag{5.5}$$

where $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \dot{\mathbf{M}}(\mathbf{q})\dot{\mathbf{q}} - \frac{1}{2}\dot{\mathbf{q}}^\top \partial_{\mathbf{q}}\mathbf{M}(\mathbf{q})\dot{\mathbf{q}}$. For convenience, we will define this term as

$$\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{M}}(\mathbf{q})\dot{\mathbf{q}} - \tfrac{1}{2}\dot{\mathbf{q}}^\top \partial_{\mathbf{q}}\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} \tag{5.6}$$

which will play an important role when we discuss about the geometry of implicit task spaces. (This definition is consistent with the curvature term in GDSs that we later generalize.)

Ambient Geometric Mechanics: Geometric mechanics [137] is the application of differential geometry to describe mechanical systems. According to it, geometry is an intrinsic part of mechanics as the space of all possible configurations of a physical mechanical system has a natural geometric structure where constraints are intrinsically satisfied by that geometry. To see how geometry can arise in mechanics let's for instance use a two link manipulator. The system can be described by the points on the elbow and the end-effector in 3D Euclidean space. Thus, we need two 3D particle coordinates as well as two constraints each, one for the plane the link lies in and the other for the sphere on which the particle lies.

Alternatively, is it much easier to describe every configuration of this system by the angles of its two joints (i.e configuration space) such that the space of all possible states is then described by the surface of a torus (doughnut). This manifold is the configuration manifold as it encodes the constraints as well. In general, by choosing an appropriate geometric representation it is possible to intrinsically enforce the constraints of the system.

We are interested in evolution of the mechanical system and not just isolated states on the configuration manifold. Given a set of consecutive configurations, that form a smooth curve on the manifold, at consecutive times separated by some time intervals, if some interval approaches an infinitesimal value the distance on the manifold between the configurations around that interval also tends to zero. This curve describes the evolution of the system between the time from the first configuration to the time at the last configuration on that curve. Typically we only have access to the current state of the system comprising of the configuration position and velocity and can be used to find the system's evolution. The velocity vector at some time is by construction a tangent to the curve, and since the curve lies on the manifold the tangent then lies on the tangent space of the manifold at that point along that curve. To obtain the time evolution we can now apply Hamilton's principle of least action that states that physical paths on the configuration space are least resistant paths with respect to an action functional. This functional is the Lagrangian of the system that is the difference between the kinetic and potential energies as we have see above. Thus we can go from the Hamilton's principle to the Euler-Lagrange equations that provide the evolution of the system. The metric that is associated with the manifold and is in the kinetic energy later appears as the inertia of the system after solving the Euler-Lagrange equations and thus conservation of energy leads to the evolution of the system as a geodesic on the manifold. We will discuss this in more detail below.

Thus geometric mechanics can be viewed as a reformulation of classical mechanics that builds on the observation that classical mechanical systems evolve as geodesics across a (configuration) manifold whose geometry is defined by the system's inertia matrix.

To make the connection more clear we will use a simple example to derive an operational space QP similar in form to Eq. (5.2). Let $\mathbf{x} = \psi(\mathbf{q})$ be an arbitrary differentiable task map $\psi : \mathbb{R}^d \to \mathbb{R}^n$ where $n \geq d$. In practice, the full task map often consists of many smaller task maps stacked on top of one another, like a kinematic tree. We can define a positive definite matrix that changes as a function of configuration using $\mathbf{M}(\mathbf{q}) = \mathbf{J}(\mathbf{q})^\top \mathbf{J}(\mathbf{q})$ where $\mathbf{J}(\mathbf{q}) = \partial_{\mathbf{q}} \phi(\mathbf{q})$ is the Jacobian of the task map. Geometric mechanics states that we can think of $\mathbf{M}(\mathbf{q})$ as both the generalized inertia matrix of a mechanical system defining a dynamic behavior $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \tau_{\text{ext}}$ (see also Eq. (5.5)), and equivalently as a Riemannian metric defining an inner product $\langle \dot{\mathbf{q}}_1, \dot{\mathbf{q}}_2 \rangle_{\mathbf{M}} = \dot{\mathbf{q}}_1^\top \mathbf{M} \dot{\mathbf{q}}_2$ on the tangent space (for our purposes, the space of velocities $\dot{\mathbf{q}}$ at a given $\mathbf{q}$) of the configuration space $\mathcal{C}$ (the manifold where $\mathbf{q}$ lives). Note that since $\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$, the columns of $\mathbf{J}(\mathbf{q})$ span this tangent space, which we see from the dependency on $\mathbf{q}$ in $\mathbf{J}$ can change direction at different $\mathbf{q}$.

Because we can suppose $n \geq d$ in general[5], the set $\mathcal{X} = \{\mathbf{x} : \mathbf{x} = \psi(\mathbf{q}) \text{ for some } \mathbf{q} \in \mathcal{C}\}$ sweeps out a d-dimensional sub-manifold of the n-dimensional ambient Euclidean task space. In light of this picture, we can also view the tangent space as a first-order Taylor approximation to the surface at a point $\mathbf{x}_0 = \phi(\mathbf{q}_0)$ for some $\mathbf{q}_0$ in the sense $\mathbf{x} \approx \mathbf{x}_0 + \mathbf{J}(\mathbf{q} - \mathbf{q}_0)$.

Indeed, one connection between the mechanical system and this geometry is clear: the kinetic energy of the mechanical system is given by the norm of $\dot{\mathbf{q}}$ with respect to the inner product defined by the metric $\mathbf{M}(\mathbf{q})$:

$$\mathcal{K}(\mathbf{q}, \dot{\mathbf{q}}) = \tfrac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} = \|\dot{\mathbf{q}}\|_{\mathbf{M}}^2 = \langle \dot{\mathbf{q}}, \dot{\mathbf{q}} \rangle_{\mathbf{M}}. \tag{5.7}$$

Likewise, in this particular case, since $\mathbf{M} = \mathbf{J}^\top \mathbf{J}$ we see that specifically the kinetic energy

[5]This holds when ψ is full rank. Reduced rank ψ result in a similar geometry, but we would need to slightly modify the linear algebra used in the following discussion.

is given by the Euclidean velocity through the task space

$$\mathcal{K}(\mathbf{q}, \dot{\mathbf{q}}) = \tfrac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} = \tfrac{1}{2}\dot{\mathbf{q}}^\top \big(\mathbf{J}^\top \mathbf{J}\big)\dot{\mathbf{q}} = \tfrac{1}{2}\|\dot{\mathbf{x}}\|^2. \tag{5.8}$$

More generally, for the same reason, Euclidean inner products between velocities in the task space induce these generalized inner products in the configuration space in the sense $\dot{\mathbf{x}}_1^\top \dot{\mathbf{x}}_2 = \dot{\mathbf{q}}_1^\top \mathbf{M} \dot{\mathbf{q}}_2$. This connection between task space and configuration space inner products, exemplified by the equivalence between task space velocity and the system's kinetic energy offers a concrete connection between mechanics and geometry, and we can exploit to link the system's equations of motion to geodesics across $\mathcal{X}$.

For systems without potential functions and external forces, we can get some insight into the connection between dynamics and geodesics from the view point of Lagrangian mechanics as well. The Lagrangian Eq. (5.3) in this case simplifies to $\mathcal{L} = \tfrac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}\dot{\mathbf{q}} - \Phi(\mathbf{q}) = \tfrac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}\dot{\mathbf{q}}$. The Euler-Lagrange equation in Eq. (5.4) is the first-order optimality condition of an *action functional* which measures the time integral of the Lagrangian across a trajectory. In this case, it takes on a nice minimization form

$$\min_\xi \int_a^b \tfrac{1}{2}\dot{\mathbf{q}}^\top \mathbf{M}\dot{\mathbf{q}} dt \;\; \Leftrightarrow \;\; \min_\xi \int_a^b \tfrac{1}{2}\|\dot{\mathbf{x}}\|^2 dt, \tag{5.9}$$

where ξ is a trajectory through the configuration space $\mathcal{C}$. One can show that these trajectories are length-minimizing trajectories (i.e. solutions extremize the length functional $\int_a^b \tfrac{1}{2}\|\dot{\mathbf{x}}\| dt$), but with the additional property that the trajectories are of constant velocity.

This means the dynamical system will curve across the manifold $\mathcal{X}$ along a trajectory that is as straight as possible without speeding up or slowing down. Another way to characterize that statement, is to say the system never accelerates tangentially to the sub-manifold $\mathcal{X}$, i.e. it has no component of acceleration parallel to the tangent space. The curve certainly must accelerate to avoid diverging from the sub-manifold $\mathcal{X}$, but that acceleration is always purely orthogonal to the tangent space. Since we know that $\mathbf{J}$ spans the tangent

space, we can capture that sentiment fully in the following simple equation:

$$\mathbf{J}^\top \ddot{\mathbf{x}} = \mathbf{0}. \tag{5.10}$$

Plugging in $\ddot{\mathbf{x}} = \mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}}$ we get

$$\begin{aligned} \mathbf{J}^\top \ddot{\mathbf{x}} &= \mathbf{J}^\top \left(\mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}} \right) = \mathbf{0} \\ \Rightarrow \left(\mathbf{J}^\top \mathbf{J} \right) \ddot{\mathbf{q}} &+ \mathbf{J}^\top \dot{\mathbf{J}}\dot{\mathbf{q}} = \mathbf{0}. \end{aligned} \tag{5.11}$$

Comparing to Eq. (5.5) (with zero potential and external forces), since we already know $\mathbf{J}^\top \mathbf{J} = \mathrm{M}$, we can formally prove the connection between geodesics and classical mechanical dynamics if we can show that $\mathbf{J}^\top \dot{\mathbf{J}}\dot{\mathbf{q}} = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$. The required calculation is fairly involved, so we omit it here but note for those inclined that it's easiest to perform using tensor notation and the Einstein summation convention as is common in differential geometry. This equivalence also appears as by-product fo our RMPflow and GDS analysis, as we will revisit in chapter B as lemma 1.

Forced mechanical systems and geometric control: So far we have derived only the unforced behavior of this system as natural geodesic flow across the sub-manifold. To understand how desired accelerations contribute to the least squares properties of the system we express Eq. (5.11) in $\mathbf{x}$ by pushing them through the identity $\ddot{\mathbf{x}} = \mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}}$ and examine how arbitrary motion across the sub-manifold $\mathcal{X}$ decomposes. First, the equations of motion in $\ddot{\mathbf{x}}$ with Eq. (5.11), we get

$$\begin{aligned} \ddot{\mathbf{x}} = \mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}} &= -\mathbf{J}\left(\mathbf{J}^\top \mathbf{J}\right)^{-1} \mathbf{J}^\top \dot{\mathbf{J}}\dot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}} \\ &= \left(\mathbf{I} - \mathbf{J}\left(\mathbf{J}^\top \mathbf{J}\right)^{-1} \mathbf{J}^\top \right) \dot{\mathbf{J}}\dot{\mathbf{q}} \\ &= \mathbf{P}_\perp \dot{\mathbf{J}}\dot{\mathbf{q}}, \end{aligned} \tag{5.12}$$

where in the final expression, the matrix $\mathbf{P}_\perp = \mathbf{I} - \mathbf{P}_{/\!/}$ with $\mathbf{P}_{/\!/} = \mathbf{J}\left(\mathbf{J}^\top \mathbf{J}\right)^{-1} \mathbf{J}^\top$ is the

nullspace projection operator projecting onto the space orthogonal to the tangent space (spanned by the Jacobian $\mathbf{J}$). Note again these projections $\mathbf{P}_\perp$ and $\mathbf{P}_{/\!/}$ are functions of the configuration $\mathbf{q}$.

By construction, these geodesics accelerate only orthogonally to the tangent space. This implies that any trajectory, traveling on the sub-manifold $\mathcal{X}$ but deviating from geodesics, would necessarily maintain an acceleration component parallel to the tangent space, which we might write as $\ddot{\mathbf{x}}_{/\!/}$. Importantly, any such trajectory must still accelerate exactly as Eq. (5.12) in the orthogonal direction in order to stay moving along the sub-manifold $\mathcal{X}$. Therefore, we see that the overall acceleration of a trajectory on $\mathcal{X}$ can decomposed nicely into the geodesic acceleration and the tangential acceleration:

$$\ddot{\mathbf{x}} = \mathbf{P}_\perp \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{P}_{/\!/}\ddot{\mathbf{x}}_d, \tag{5.13}$$

where $\ddot{\mathbf{x}}^d$ is any vector of "desired" acceleration whose tangential component matches the tangential acceleration of the given trajectory, i.e. $\mathbf{P}_{/\!/}\ddot{\mathbf{x}}^d = \ddot{\mathbf{x}}_{/\!/}$.

Now we show how the decomposition Eq. (5.13) is related to and generalizes Eq. (5.2). This is based on the observation that Eq. (5.13) is the same as the solution to the least-squared problem below

$$\min_{\ddot{\mathbf{q}}} \tfrac{1}{2}\|\ddot{\mathbf{x}}^d - \ddot{\mathbf{x}}\|^2 \quad \text{s.t. } \ddot{\mathbf{x}} = \mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}}. \tag{5.14}$$

This equivalence can be easily seen by resolving the constraint, setting the gradient of the resulting quadratic to zero, i.e.,

$$\mathbf{J}^\top \mathbf{J}\ddot{\mathbf{q}} + \mathbf{J}^\top \dot{\mathbf{J}}\dot{\mathbf{q}} = \mathbf{J}^\top \ddot{\mathbf{x}}^d \tag{5.15}$$

and re-expressing the optimal solution in $\ddot{\mathbf{x}}$. This relationship demonstrates that a QP very similar in structure to Eq. (5.2).

The curvature terms: As a side note, the above discussion offers insight into the term $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{J}^\top \dot{\mathbf{J}}\dot{\mathbf{q}}$. The term $\dot{\mathbf{J}}\dot{\mathbf{q}}$ captures components describing both curvature of the manifold through the ambient task space and components describing how the specific coordinate $\mathbf{q}$ (tangentially) curves across the sub-manifold $\mathcal{X}$. Explicitly, $\ddot{\mathbf{x}}^c = \dot{\mathbf{J}}\dot{\mathbf{q}}$ has units of acceleration in the ambient space and captures how the tangent space (given by the columns of $\mathbf{J}$) changes in the direction of motion. The acceleration $\ddot{\mathbf{x}}^c$ decomposes as $\ddot{\mathbf{x}}^c = \ddot{\mathbf{x}}^c_\perp + \ddot{\mathbf{x}}^c_{/\!/}$, into two orthogonal components consisting of a component perpendicular to the tangent space $\ddot{\mathbf{x}}^c_\perp = \mathbf{P}_\perp \ddot{\mathbf{x}}^c$ and a component parallel to the tangent space $\ddot{\mathbf{x}}^c_{/\!/} = \mathbf{P}_{/\!/}\ddot{\mathbf{x}}^c$. The term $\mathbf{P}_\perp \dot{\mathbf{J}}\dot{\mathbf{q}} = \mathbf{P}_\perp \ddot{\mathbf{x}}^c$ given in Eq. (5.12) extracts specifically the perpendicular component $\ddot{\mathbf{x}}^c_\perp$. The other component $\ddot{\mathbf{x}}^c_{/\!/}$ is, therefore, in a sense irrelevant to fundamental geometric behavior of the underlying system, and is only required when expressing the behavior in the specific coordinates $\mathbf{q}$. Indeed, when expressing the equations of motion in $\mathbf{q}$, the related term manifests as $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{J}^\top \dot{\mathbf{J}}\dot{\mathbf{q}} = \mathbf{J}^\top \ddot{\mathbf{x}}^c_{/\!/}$ since $\mathbf{J}^\top \ddot{\mathbf{x}}^c_\perp = \mathbf{0}$, and depends only on the parallel component $\ddot{\mathbf{x}}^c_{/\!/}$. This observation emphasizes why we designate the term *fictitious forces*. Here and below we will consider these terms to be *curvature* terms as they compensate for curvature in the system coordinates.

Non-constant weights and implicit task spaces

The above derived the geometric perspective of equations of motion, but only for mechanical systems whose inertia matrix (equiv. Riemannian metric) can be expressed as $\mathbf{M}(\mathbf{q}) = \mathbf{J}(\mathbf{q})^\top \mathbf{J}(\mathbf{q})$ globally for some map $\mathbf{x} = \psi(\mathbf{q})$ with $\mathbf{J}(\mathbf{q}) = \partial_{\mathbf{q}}\phi(\mathbf{q})$. Fortunately, due to a deep and fundamental theorem proved by John Nash in 1956, called the Nash embedding theorem [154], *all* Riemannian manifolds, and hence *all* mechanical systems can be expressed this way, so the arguments of abstract task spaces above hold without loss of generality for all mechanical systems. It is called an embedding theorem because the map $\mathbf{x} = \psi(\mathbf{q})$ acts to embed the manifold $\mathcal{C}$ into a higher-dimensional ambient Euclidean space where we can replace implicit geometry represented by the metric $\mathbf{M}(\mathbf{q})$ with an explicit

sub-manifold in the ambient space.

This ambient representation is a convenient for understanding and visualizing the non-linear geometry of a mechanical system, but it is unfortunately often difficult, or even impossible, to find a closed form expression for a task map $\mathbf{x} = \psi(\mathbf{q})$ from a given metric $\mathbf{M}(\mathbf{q})$. We, therefore, cannot rely our ability to operate directly in the ambient space using the QP given in Eq. (5.14).

This subsection addresses that problem by deriving a QP expression analogous to Eq. (5.14), but for which task space weights are general non-constant positive definite matrices (which we will see are the same as Riemannian metrics). We will arrive at this expression by considering again the ambient setting, but assuming that the unknown embedding can be decomposed in the composition of a known task space $\mathbf{x} = \psi(\mathbf{q})$ and then a known map from the task space to the ambient Euclidean space $\mathbf{z} = \zeta(\mathbf{x})$ described in the Nash's theorem. We suppose the priority weight is given as the induced Riemannian metric $\mathbf{G}(\mathbf{x}) = \mathbf{J}_\zeta(\mathbf{x})^\top \mathbf{J}_\zeta(\mathbf{x})$ on $\mathbf{x}$ defined by the second map ζ. We note that the final result will be expressed entirely in terms of $\mathbf{G}(\mathbf{x})$, so it can be used without explicit knowledge of ζ.

Suppose we have a Riemannian metric (inertia matrix) $\mathbf{M}$ which decomposes as $\mathbf{M} = \mathbf{J}^\top \mathbf{J}$, where $\mathbf{J} = \mathbf{J}_\zeta \mathbf{J}_\psi$ is the Jacobian of the composite map $\mathbf{z} = \zeta \circ \psi(\mathbf{q})$ which itself consists of two parts $\mathbf{x} = \psi(\mathbf{q})$ and $\mathbf{z} = \zeta(\mathbf{x})$. Because the intermediate task space metric is $\mathbf{G} = \mathbf{J}_\zeta^\top \mathbf{J}_\zeta$, denote the task space force as $\mathbf{f}_\mathbf{x} = \mathbf{J}_\zeta^\top \mathbf{f}_\mathbf{z}$ and by Eq. (5.15) we have

$$
\begin{aligned}
&\mathbf{J}^\top \mathbf{J}\ddot{\mathbf{q}} + \mathbf{J}^\top \dot{\mathbf{J}}\dot{\mathbf{q}} = \mathbf{J}^\top \mathbf{f}_\mathbf{z} \\
\Rightarrow\ & \left(\mathbf{J}_\psi^\top \left(\mathbf{J}_\zeta^\top \mathbf{J}_\zeta\right) \mathbf{J}_\psi\right) \ddot{\mathbf{q}} + \mathbf{J}_\psi^\top \mathbf{J}_\zeta^\top \frac{d}{dt} \left(\mathbf{J}_\zeta \mathbf{J}_\psi\right) \dot{\mathbf{q}} = \mathbf{J}_\psi^\top \mathbf{J}_\zeta^\top \mathbf{f}_\mathbf{z} \\
\Rightarrow\ & \mathbf{J}_\psi^\top \mathbf{G} \mathbf{J}_\psi \ddot{\mathbf{q}} + \mathbf{J}_\psi^\top \mathbf{J}_\zeta^\top \left(\dot{\mathbf{J}}_\zeta \mathbf{J}_\psi + \mathbf{J}_\zeta \dot{\mathbf{J}}_\psi\right) \dot{\mathbf{q}} = \mathbf{J}_\psi^\top \mathbf{f}_\mathbf{x} \\
\Rightarrow\ & \mathbf{J}_\psi^\top \mathbf{G} \mathbf{J}_\psi \ddot{\mathbf{q}} + \mathbf{J}_\psi^\top \left(\mathbf{J}_\zeta^\top \dot{\mathbf{J}}_\zeta \dot{\mathbf{x}}\right) + \mathbf{J}_\psi^\top \left(\mathbf{J}_\zeta^\top \mathbf{J}_\zeta\right) \dot{\mathbf{J}}_\psi \dot{\mathbf{q}} = \mathbf{J}_\psi^\top \mathbf{f}_\mathbf{x} \\
\Rightarrow\ & \mathbf{J}_\psi^\top \mathbf{G} \mathbf{J}_\psi \ddot{\mathbf{q}} + \mathbf{J}_\psi^\top \mathbf{G} \dot{\mathbf{J}}_\psi \dot{\mathbf{q}} = \mathbf{J}_\psi^\top \left(\mathbf{f}_x - \boldsymbol{\xi}_\mathbf{G}\right).
\end{aligned}
$$

where we recall $\boldsymbol{\xi}_\mathbf{G}$ in given in Eq. (5.6). Rearranging that final expression and denoting

$\underline{\ddot{\mathbf{x}}}^d = \ddot{\mathbf{x}}^d - \mathbf{G}^{-1}\boldsymbol{\xi}_{\mathbf{G}}$ with $\ddot{\mathbf{x}}^d = \mathbf{G}^{-1}\mathbf{f}_{\mathbf{x}}$, we can write the above equation as

$$\mathbf{J}_{\psi}^{\top}\mathbf{G}\left(\underline{\ddot{\mathbf{x}}}^d - \left(\mathbf{J}_{\psi}\ddot{\mathbf{q}} + \dot{\mathbf{J}}_{\psi}\dot{\mathbf{q}}\right)\right) = \mathbf{0}, \tag{5.16}$$

which is the first-order optimality condition of the QP

$$\min_{\ddot{\mathbf{q}}} \tfrac{1}{2}\|\underline{\ddot{\mathbf{x}}}^d - \ddot{\mathbf{x}}\|^2_{\mathbf{G}} \qquad \text{s.t. } \ddot{\mathbf{x}} = \mathbf{J}_{\psi}\ddot{\mathbf{q}} + \mathbf{J}_{\psi}\dot{\mathbf{q}}. \tag{5.17}$$

This QP is expressed in terms of the task map $\mathbf{x} = \psi(\mathbf{q})$, the task space metric $\mathbf{G}(\mathbf{x})$, the task space desired accelerations $\ddot{\mathbf{x}}^d$, and the curvature term $\boldsymbol{\xi}_{\mathbf{G}}$ derived from the task space metric $\mathbf{G}(\mathbf{x})$. The QP follows a very similar pattern to the QPs described above, but this time the priority weight matrix $\mathbf{G}$ is a non-constant function of $\mathbf{x}$. The one modification required to reach this matching form is to augment the desired acceleration $\ddot{\mathbf{x}}^d$ with the curvature term $\boldsymbol{\xi}_{\mathbf{G}}$ calculated from $\mathbf{G}$ using Eq. (5.6) to get the target $\underline{\ddot{\mathbf{x}}}^d = \ddot{\mathbf{x}}^d - \mathbf{G}^{-1}\boldsymbol{\xi}_{\mathbf{G}}$. Importantly, while we start by assuming the map $\mathbf{z} = \zeta(\mathbf{x})$, at the end we show that we actually only need to know $\mathbf{G}$.

Limitations of geometric control

Even with the tools of geometric mechanics, the final QP given in Eq. (5.17) can still only express task priority weights as positive definite matrices that vary as a function of configuration (i.e. position). Frequently, more nuanced control over those priorities is crucial. For instance, collision avoidance tasks should activate when the control-point is close to an obstacle and heading toward it, but they should deactivate either when the control-point is far from the obstacle *or* when it's moving away from the obstacle, regardless of its proximity. Importantly, reducing the desired acceleration to zero in these cases is not enough—when these tasks deactivate, they should drop entirely from the equation rather than voting with high weight for zero acceleration. Enabling priorities vary as a function of the full robot state (configuration *and* velocity) is therefore paramount.

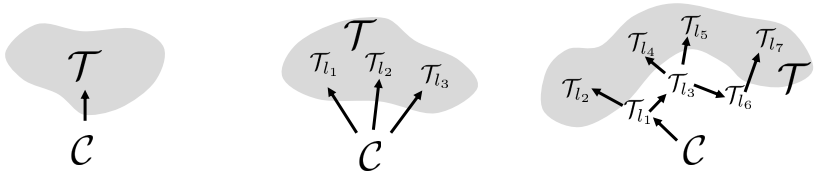
The theory of RMPflow and GDSs developed below generalizes geometric mechanics to enable expressing these more nuanced priority matrices while maintaining stability. Additionally, since geometric control theory itself is quite abstract, we build on results reducing the calculations to recursive least squares to derive a concrete tree data structure to aid in the design of controllers within this energy shaping framework.

5.5 Structure with task-map trees

RMPflow is an efficient manifold-oriented computational graph for automatic generation of motion policies. It is aimed for problems with a task space $\mathcal{T} = \{\mathcal{T}_{l_i}\}$ that is related to the configuration space $\mathcal{C}$ through a tree-structured task map ψ, where $\mathcal{T}_{l_i}$ is the ith subtask. Given user-specified motion policies $\{\pi_{l_i}\}$ on $\{\mathcal{T}_{l_i}\}$ as RMPs, RMPflow is designed to *consistently* combine these subtask policies into a global policy π on $\mathcal{C}$. To this end, RMPflow introduces 1) a data structure, called the *RMP-tree*, to describe the tree-structured task map ψ and the policies, and 2) a set of operators, called the *RMP-algebra*, to propagate information across the RMP-tree. To compute $\pi(\mathbf{q}(t), \dot{\mathbf{q}}(t))$ at time t, RMPflow operates in two steps: it first performs a *forward pass* to propagate the state from the root node (i.e. $\mathcal{C}$) to the leaf nodes (i.e. $\{\mathcal{T}_{l_i}\}$); then it performs a *backward pass* to propagate the RMPs from the leaf nodes to the root node while tracking their geometric information to achieve consistency. These two steps are realized by recursive use of RMP-algebra, exploiting shared computation paths arising from the tree structure to maximize efficiency.

5.5.1 Task-maps

In most cases, the task-space manifold $\mathcal{T}$ is structured. In this paper, we consider the case where the task map ψ can be expressed through a tree-structured composition of transformations $\{\psi_{e_i}\}$, where ψ_{e_i} is the ith transformation. Figure 5.1 illustrates some common examples. Each node denotes a manifold and each edge denotes a transformation. This family trivially includes the unstructured task space $\mathcal{T}$ (Figure 5.1a) and the product man-



(a) (b) (c)

Figure 5.1: Tree-structured task maps

ifold $\mathcal{T} = \mathcal{T}_{l_1} \times \cdots \times \mathcal{T}_{l_K}$ (Figure 5.1b), where K is the number of subtasks. A more interesting example is the kinematic tree (Figure 5.1c), where, e.g., the subtask spaces on the leaf nodes can describe the tracking and obstacle avoidance tasks along a multi-DOF robot.

The main motivation of explicitly handling the structure in the task map ψ is two-fold. First, it allows RMPflow to exploit computation shared across different subtask maps. Second, it allows the user to focus on designing motion policies for each subtask individually, which is easier than directly designing a global policy for the entire task space $\mathcal{T}$. For example, $\mathcal{T}$ may describe the problem of humanoid walking, which includes staying balanced, scheduling contacts, and avoiding collisions. Directly parameterizing a policy to satisfy all these objectives can be daunting, whereas designing a policy for each subtask is more feasible.

5.5.2 Riemannian motion policies

Knowing the structure of the task map is not sufficient for consistently combining subtask policies: we require some geometric information about the motion policies' behaviors [139]. Toward this end, we adopt an abstract description of motion policies, called RMPs [139], for the nodes of the RMP-tree. Specifically, let $\mathcal{M}$ be an m-dimensional manifold with coordinate $\mathbf{x} \in \mathbb{R}^m$. The *canonical form* of an RMP on $\mathcal{M}$ is a pair $(\mathbf{a}, \mathbf{M})^{\mathcal{M}}$, where $\mathbf{a} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^m$ is a continuous motion policy and $\mathbf{M} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^{m \times m}_+$ is a differentiable map. Borrowing terminology from mechanics, we call $\mathbf{a}(\mathbf{x}, \dot{\mathbf{x}})$ the *de-*

sired acceleration and $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})$ the *inertia matrix* at $(\mathbf{x}, \dot{\mathbf{x}})$, respectively.[6] $\mathbf{M}$ defines the directional importance of $\mathbf{a}$ when it is combined with other motion policies. Later in Section 5.7, we will show that $\mathbf{M}$ is closely related to Riemannian metric, which describes how the space is stretched along the curve generated by $\mathbf{a}$; when $\mathbf{M}$ depends on the state, the space becomes *non-Euclidean*. We additionally introduce a new RMP form, called the *natural form*. Given an RMP in its canonical form $(\mathbf{a}, \mathbf{M})^{\mathcal{M}}$, the natural form is a pair $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$, where $\mathbf{f} = \mathbf{M}\mathbf{a}$ is the *desired force* map. While the transformation between these two forms may look trivial, their distinction will be useful later when we introduce the RMP-algebra.

5.5.3 RMP-tree

The RMP-tree is the core data structure used by RMPflow. An RMP-tree is a directed tree, in which each node represents an RMP and its state, and each edge corresponds to a transformation between manifolds. The root node of the RMP-tree describes the global policy π on $\mathcal{C}$, and the leaf nodes describe the local policies $\{\pi_{l_i}\}$ on $\{\mathcal{T}_{l_i}\}$. To illustrate, let us consider a node u and its K child nodes $\{v_i\}_{i=1}^K$. Suppose u describes an RMP $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$ and v_i describes an RMP $[\mathbf{f}_i, \mathbf{M}_i]^{\mathcal{N}_i}$, where $\mathcal{N}_i = \psi_{e_i}(\mathcal{M})$ for some ψ_{e_i}. Then we write $u = ((\mathbf{x}, \dot{\mathbf{x}}), [\mathbf{f}, \mathbf{M}]^{\mathcal{M}})$ and $v_i = ((\mathbf{y}_i, \dot{\mathbf{y}}_i), [\mathbf{f}_i, \mathbf{M}_i]^{\mathcal{N}_i})$; the edge connecting u and v_i points from u to v_i along ψ_{e_i}. We will continue to use this example to illustrate how RMP-algebra propagates the information across the RMP-tree.

5.6 Automatic motion policy generation

5.6.1 RMP-algebra

The RMP-algebra consists of three operators (`pushforward`, `pullback`, and `resolve`) to propagate information.[7] They form the basis of the forward and backward passes for

[6] Here we adopt a slightly different terminology from [139]. We note that $\mathbf{M}$ and $\mathbf{f}$ do not necessarily correspond to the inertia and force of a physical mechanical system.

[7] Precisely it propagates the numerical values of RMPs and states at a particular time.

automatic policy generation.

1. `pushforward` is the operator to forward propagate the *state* from a parent node to its child nodes. Using the previous example, given $(\mathbf{x}, \dot{\mathbf{x}})$ from u, it computes $(\mathbf{y}_i, \dot{\mathbf{y}}_i) = (\psi_{e_i}(\mathbf{x}), \mathbf{J}_i(\mathbf{x})\dot{\mathbf{x}})$ for each child node v_i, where $\mathbf{J}_i = \partial_{\mathbf{x}}\psi_{e_i}$ is a Jacobian matrix. The name "pushforward" comes from the linear transformation of tangent vector $\dot{\mathbf{x}}$ to the image tangent vector $\dot{\mathbf{y}}_i$.

2. `pullback` is the operator to backward propagate the natural-formed RMPs from the child nodes to the parent node. It is done by setting $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$ with

$$\mathbf{f} = \sum_{i=1}^{K} \mathbf{J}_i^{\top}(\mathbf{f}_i - \mathbf{M}_i\dot{\mathbf{J}}_i\dot{\mathbf{x}}) \qquad \text{and} \qquad \mathbf{M} = \sum_{i=1}^{K} \mathbf{J}_i^{\top}\mathbf{M}_i\mathbf{J}_i \tag{5.18}$$

   The name "pullback" comes from the linear transformations of the cotangent vector (1-form) $\mathbf{f}_i - \mathbf{M}_i\dot{\mathbf{J}}_i\dot{\mathbf{x}}$ and the inertia matrix (2-form) $\mathbf{M}_i$. In summary, velocities can be pushfowarded along the direction of ψ_i, and forces and inertial matrices can be pullbacked in the opposite direction.

   To gain more intuition of `pullback`, we write `pullback` in the canonical form of RMPs. It can be shown that the canonical form $(\mathbf{a}, \mathbf{M})^{\mathcal{M}}$ of the natural form $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$ above is the solution to a least-squared problem:

$$\mathbf{a} = \operatorname{argmin}_{\mathbf{a}'} \tfrac{1}{2} \sum_{i=1}^{K} \|\mathbf{J}_i\mathbf{a}' + \dot{\mathbf{J}}_i\dot{\mathbf{x}} - \mathbf{a}_i\|^2_{\mathbf{M}_i} \tag{5.19}$$

   where $\mathbf{a}_i = \mathbf{M}_i^{\dagger}\mathbf{f}_i$ and $\|\cdot\|^2_{\mathbf{M}_i} = \langle \cdot, \mathbf{M}_i \cdot \rangle$. Because $\ddot{\mathbf{y}}_i = \mathbf{J}_i\ddot{\mathbf{x}} + \dot{\mathbf{J}}_i\dot{\mathbf{x}}$, `pullback` attempts to find an $\mathbf{a}$ that can realize the desired accelerations $\{\mathbf{a}_i\}$ while trading off approximation errors with an importance weight defined by the inertia matrix $\mathbf{M}_i(\mathbf{y}_i, \dot{\mathbf{y}}_i)$. The use of state dependent importance weights is a distinctive feature of RMPflow. It allows RMPflow to activate different RMPs according to *both* configuration and velocity (see Section 5.6.3 for examples). Finally, we note that the

`pullback` operator defined in this paper is slightly different from the original definition given in [139], which ignores the term $\dot{\mathbf{J}}_i\dot{\mathbf{x}}$ in Eq. (5.19). While ignoring $\dot{\mathbf{J}}_i\dot{\mathbf{x}}$ does not necessary destabilize the system [143], its inclusion is critical to implement consistent policy behaviors.

3. `resolve` is the last operator of RMP-algebra. It maps an RMP from its natural form to its canonical form. Given $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$, it outputs $(\mathbf{a}, \mathbf{M})^{\mathcal{M}}$ with $\mathbf{a} = \mathbf{M}^{\dagger}\mathbf{f}$, where $\dagger$ denotes Moore-Penrose inverse. The use of pseudo-inverse is because in general the inertia matrix is only positive semi-definite. Therefore, we also call the natural form of $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$ the *unresolved form*, as potentially it can be realized by multiple RMPs in the canonical form.

5.6.2 Algorithm

Now we show how RMPflow uses the RMP-tree and RMP-algebra to generate a global policy π on $\mathcal{C}$ from the user-specified subtask policies $\{\pi_{l_i}\}$ on $\{\mathcal{T}_{l_i}\}$. Suppose each subtask policy is provided as an RMP. First, we construct an RMP-tree with the same structure as ψ, where we assign subtask RMPs as the leaf nodes and the global RMP $[\mathbf{f}_r, \mathbf{M}_r]^{\mathcal{C}}$ as the root node. With the RMP-tree specified, RMPflow can perform automatic policy generation. At every time instance, it first performs a forward pass: it recursively calls `pushforward` from the root node to the leaf nodes to update the state information in each node in the RMP-tree. Second, it performs a backward pass: it recursively calls `pullback` from the leaf nodes to the root node to back propagate the values of the RMPs in the natural form, and finally calls `resolve` at the root node to transform the global RMP $[\mathbf{f}_r, \mathbf{M}_r]^{\mathcal{C}}$ into its canonical form $(\mathbf{a}_r, \mathbf{M}_r)^{\mathcal{C}}$ for policy execution (i.e. setting $\pi(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{a}_r$).

The process of policy generation of RMPflow uses the tree structure for computational efficiency. For K subtasks, it has time complexity $O(K)$ in the worst case as opposed to $O(K \log K)$ of a naive implementation which does not exploit the tree structure. Furthermore, all computations of RMPflow are carried out using matrix-multiplications, except for

the final `resolve` call, because the RMPs are expressed in the natural form in `pullback` instead of the canonical form suggested originally in [139]. This design makes RMPflow numerically stable, as only one matrix inversion $\mathbf{M}_r^\dagger \mathbf{f}_r$ is performed at the root node with both $\mathbf{f}_r$ and $\mathbf{M}_r$ in the span of the same Jacobian matrix due to `pullback`.

5.6.3 Example RMPs

We give a quick overview of some RMPs useful in practice (for a complete discussion of these RMPs please see [138]). We recall from Eq. (5.19) that $\mathbf{M}$ dictates the directional importance of an RMP.

Collision/joint limit avoidance

Barrier-type RMPs are examples that use velocity dependent inertia matrices, which can express importance as a function of robot heading (a property that traditional mechanical principles fail to capture). Here we demonstrate a collision avoidance policy in the 1D distance space $x = d(\mathbf{q})$ to an obstacle. Let $g(x, \dot{x}) = w(x)u(\dot{x}) > 0$ for some functions w and u. We consider a motion policy such that $m(x, \dot{x})\ddot{x} + \frac{1}{2}\dot{x}^2 \partial_x g(x, \dot{x}) = -\partial_x \Phi(x) - b\dot{x}$ and define its inertia matrix $m(x, \dot{x}) = g(x, \dot{x}) + \frac{1}{2}\dot{x}\partial_{\dot{x}} g(x, \dot{x})$, where Φ is a potential and $b > 0$ is a damper. We choose $w(x)$ to increase as x decreases (close to the obstacle), $u(\dot{x})$ to increase when $\dot{x} < 0$ (moving toward the obstacle), and $u(\dot{x})$ to be constant when $\dot{x} \geq 0$. This motion policy is a GDS and g is its metric (cf. Section 5.7); the terms $\frac{1}{2}\dot{x}\partial_{\dot{x}} g(x, \dot{x})$ and $\frac{1}{2}\dot{x}^2 \partial_x g(x, \dot{x})$ are due to non-Euclidean geometry and produce natural repulsive behaviors.

Target attractors

Designing an attractor policy is relatively straightforward. For a task space with coordinate $\mathbf{x}$, we can consider an inertia matrix $\mathbf{M}(\mathbf{x}) \succ 0$ and a motion policy such that $\ddot{\mathbf{x}} = -\nabla\widetilde{\Phi} - \beta(\mathbf{x})\dot{\mathbf{x}} - \mathbf{M}^{-1}\boldsymbol{\xi}_{\mathbf{M}}$, where $\widetilde{\Phi}(\mathbf{x}) \approx \|\mathbf{x}\|$ is a smooth attractor potential, $\beta(\mathbf{x}) \geq 0$ is a damper, and $\boldsymbol{\xi}_{\mathbf{M}}$ is a curvature term.

Orientations

As RMPflow directly works with manifold objects, orientation controllers become straightforward to design, independent of the choice of coordinate. For example, we can define RMPs on a robotic link's surface in any preferred coordinate (e.g. in one or two axes attached to an arbitrary point) with the above described attractor to control the orientation. This follows a similar idea outlined in the Appendix of [139].

Q-functions

Perhaps surprising, RMPs can be constructed using Q-functions as metrics (we invite readers to read [139] for details on how motion optimizers can be reduced to Q-functions and the corresponding RMPs). While these RMPs may not satisfy the conditions of a GDS that we later analyze, they represent a broader class of RMPs that leads to substantial benefits (e.g. escaping local minima) in practice.

5.7 Theoretical analysis of RMPflow

We investigate the properties of RMPflow when the child-node motion policies belong to a class of differential equations, which we call *structured geometric dynamical systems* (structured GDSs). We show that the `pullback` operator retains a closure of structured GDSs. When the child-node motion policies are structured GDSs, the parent-node dynamics also belong to the same class. Using this closure property, we provide sufficient conditions for the feedback policy of RMPflow to be stable. In particular, we cover a class of dynamics with *velocity-dependent* metrics that are new to the literature. For the analysis on the invariance property please refer to [138].

Below we consider the manifolds in the nodes of the RMP-tree to be finite-dimensional and smooth. Without loss of generality, for now we assume that each manifold can be described in a single chart, so that we can write down the equations concretely using finite-

dimensional variables. We also assume that all the maps are sufficiently smooth so the required derivatives are well defined. The proofs of this section are provided in Appendix C.

Geometric dynamical systems

We define a new family of dynamics useful to specify RMPs on manifolds. Let manifold $\mathcal{M}$ be m-dimensional with chart $(\mathcal{M}, \mathbf{x})$. Let $\mathbf{G} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m \times m}$, $\mathbf{B} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m \times m}$, and $\Phi : \mathbb{R}^m \to \mathbb{R}$. The tuple $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)$ is called a *GDS* if and only if

$$(\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}))\, \ddot{\mathbf{x}} + \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) = -\nabla_{\mathbf{x}} \Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}, \tag{5.20}$$

where let $\mathbf{g}_i(\mathbf{x}, \dot{\mathbf{x}})$ be the ith column of $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ and we define

$$\boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) \coloneqq \tfrac{1}{2} \textstyle\sum_{i=1}^{m} \dot{x}_i \partial_{\dot{\mathbf{x}}} \mathbf{g}_i(\mathbf{x}, \dot{\mathbf{x}}) \tag{5.21}$$

$$\boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) \coloneqq \overset{\mathbf{x}}{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} - \tfrac{1}{2} \nabla_{\mathbf{x}} (\dot{\mathbf{x}}^\top \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}) \tag{5.22}$$

where $\overset{\mathbf{x}}{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) \coloneqq [\partial_{\mathbf{x}} \mathbf{g}_i(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}]_{i=1}^m$. We refer to $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ as the *metric* matrix, $\mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})$ as the *damping* matrix, and $\Phi(\mathbf{x})$ as the *potential* function which is lower-bounded. In addition, we define $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) \coloneqq \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})$ as the *inertia* matrix, which can be asymmetric. We say a GDS is *non-degenerate* if $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})$ is nonsingular. We will assume Eq. (5.20) is non-degenerate so that it uniquely defines a differential equation and discuss the general case in Appendix B. $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ induces a metric of $\dot{\mathbf{x}}$, measuring its length as $\frac{1}{2}\dot{\mathbf{x}}^\top \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}$. When $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ depends on $\mathbf{x}$ and $\dot{\mathbf{x}}$, it also induces the *curvature* terms $\boldsymbol{\Xi}(\mathbf{x}, \dot{\mathbf{x}})$ and $\boldsymbol{\xi}(\mathbf{x}, \dot{\mathbf{x}})$. In a particular case when $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{G}(\mathbf{x})$, the GDSs reduce to the widely studied *simple mechanical systems* (SMSs) [137], $\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} + \nabla_{\mathbf{x}} \Phi(\mathbf{x}) = -\mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}$; in this case $\mathbf{M}(\mathbf{x}) = \mathbf{G}(\mathbf{x})$ and the Coriolis force $\mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}$ is equal to $\boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})$. The extension to velocity-dependent $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ is important and non-trivial. As discussed in Section 5.6.3, it generalizes the dynamics of classical rigid-body systems, allowing the space to morph according to the velocity direction.

As its name suggests, GDSs possess geometric properties. Particularly, when $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ is invertible, the left-hand side of Eq. (5.20) is related to a quantity $\mathbf{a}_{\mathbf{G}} = \ddot{\mathbf{x}} + \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})^{-1} (\boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\ddot{\mathbf{x}} + \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}))$, known as the *geometric acceleration*. In short, we can think of Eq. (5.20) as setting $\mathbf{a}_{\mathbf{G}}$ along the negative natural gradient $-\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})^{-1}\nabla_{\mathbf{x}}\Phi(\mathbf{x})$ while imposing damping $-\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})^{-1}\mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}$.

Closure

Earlier, we mentioned that by tracking the geometry in `pullback` in Eq. (5.18), the task properties can be preserved. Here, we formalize the consistency of RMPflow as a closure of differential equations, named structured GDSs. Structured GDSs augment GDSs with information on how the metric matrix factorizes. Suppose $\mathbf{G}$ has a structure $\mathcal{S}$ that factorizes $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{J}(\mathbf{x})^\top \mathbf{H}(\mathbf{y}, \dot{\mathbf{y}})\mathbf{J}(\mathbf{x})$, where $\mathbf{y} : \mathbf{x} \mapsto \mathbf{y}(\mathbf{x}) \in \mathbb{R}^n$ and $\mathbf{H} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{n\times n}_+$, and $\mathbf{J}(\mathbf{x}) = \partial_{\mathbf{x}}\mathbf{y}$. We say the tuple $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$ is a *structured GDS* if and only if

$$(\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}))\,\ddot{\mathbf{x}} + \boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{x}, \dot{\mathbf{x}}) = -\nabla_{\mathbf{x}}\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \tag{5.23}$$

where $\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{x}, \dot{\mathbf{x}}) \coloneqq \mathbf{J}(\mathbf{x})^\top(\boldsymbol{\xi}_{\mathbf{H}}(\mathbf{y}, \dot{\mathbf{y}}) + (\mathbf{H}(\mathbf{y}, \dot{\mathbf{y}}) + \boldsymbol{\Xi}_{\mathbf{H}}(\mathbf{y}, \dot{\mathbf{y}}))\dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}})$. Note the metric and factorization *in combination* defines $\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}$. As a special case, GDSs are structured GDSs with a *trivial* structure (i.e. $\mathbf{y} = \mathbf{x}$). Also, structured GDSs reduce to GDSs (i.e. the structure offers no extra information) if $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{G}(\mathbf{x})$, or if $n, m = 1$ (cf. Appendix C.1). Given two structures, we say $\mathcal{S}_a$ *preserves* $\mathcal{S}_b$ if $\mathcal{S}_a$ has the factorization (of $\mathbf{H}$) made by $\mathcal{S}_b$.

Below we show the closure property: when the children of a parent node are structured GDSs, the parent node defined by `pullback` is also a structured GDS with respect to the pullbacked structured metric matrix, damping matrix, and potentials. We note that $\mathbf{G}_i$ and $\mathbf{B}_i$ can be functions of both $\mathbf{y}_i$ and $\dot{\mathbf{y}}_i$.

Theorem 1. *Let the ith child node follow $(\mathcal{N}_i, \mathbf{G}_i, \mathbf{B}_i, \Phi_i)_{\mathcal{S}_i}$ and have coordinate $\mathbf{y}_i$. Let*

$\mathbf{f}_i = -\boldsymbol{\eta}_{\mathbf{G}_i;\mathcal{S}_i} - \nabla_{\mathbf{y}_i}\Phi_i - \mathbf{B}_i\dot{\mathbf{y}}_i$ *and* $\mathbf{M}_i = \mathbf{G}_i + \boldsymbol{\Xi}_{\mathbf{G}_i}$. *If* $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$ *of the parent node is given by* `pullback` *with* $\{[\mathbf{f}_i, \mathbf{M}_i]^{\mathcal{N}_i}\}_{i=1}^K$ *and* $\mathbf{M}$ *is non-singular, the parent node follows* $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$, *where* $\mathbf{G} = \sum_{i=1}^K \mathbf{J}_i^\top \mathbf{G}_i \mathbf{J}_i$, $\mathbf{B} = \sum_{i=1}^K \mathbf{J}_i^\top \mathbf{B}_i \mathbf{J}_i$, $\Phi = \sum_{i=1}^K \Phi_i \circ \mathbf{y}_i$, $\mathcal{S}$ *preserves* $\mathcal{S}_i$, *and* $\mathbf{J}_i = \partial_{\mathbf{x}}\mathbf{y}_i$. *Particularly, if* $\mathbf{G}_i$ *is velocity-free and the child nodes are GDSs, the parent node follows* $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)$.

Theorem 1 shows structured GDSs are closed under `pullback`. It means that the differential equation of a structured GDS with a tree-structured task map can be computed by recursively applying `pullback` from the leaves to the root.

Corollary 1. *If all leaf nodes follow GDSs and* $\mathbf{M}_r$ *at the root node is nonsingular, then the root node follows* $(\mathcal{C}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$ *as recursively defined by Theorem 1.*

Stability

By the closure property above, we analyze the stability of RMPflow when the leaf nodes are (structured) GDSs. For compactness, we will abuse the notation to write $\mathbf{M} = \mathbf{M}_r$. Suppose $\mathbf{M}$ is nonsingular and let $(\mathcal{C}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$ be the resultant structured GDS at the root node. We consider a Lyapunov candidate $V(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^\top \mathbf{G}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \Phi(\mathbf{q})$ and derive its rate using properties of structured GDSs.

Proposition 1. *For* $(\mathcal{C}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$, $\dot{\mathbf{V}}(\mathbf{q}, \dot{\mathbf{q}}) = -\dot{\mathbf{q}}^\top \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$.

Proposition 1 directly implies the stability of structured GDSs by invoking LaSalle's invariance principle [144]. Here we summarize the result without proof.

Corollary 2. *For* $(\mathcal{C}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$, *if* $\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}), \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}}) \succ 0$, *the system converges to a forward invariant set* $\mathcal{C}_\infty \coloneqq \{(\mathbf{q}, \dot{\mathbf{q}}) : \nabla_{\mathbf{q}}\Phi(\mathbf{q}) = 0, \dot{\mathbf{q}} = 0\}$.

To show the stability of RMPflow, we need to further check when the assumptions in Corollary 2 hold. The condition $\mathbf{B}(\mathbf{q}, \dot{\mathbf{q}}) \succ 0$ is easy to satisfy: by Theorem 1, $\mathbf{B}(\mathbf{q}, \dot{\mathbf{q}}) \succeq 0$; to strictly ensure definiteness, we can copy $\mathcal{C}$ into an additional child node with a (small)

positive-definite damping matrix. The condition on $\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}) \succ 0$ can be satisfied similarly. In addition, we need to verify the assumption that $\mathbf{M}$ is nonsingular. Here we provide a sufficient condition. When satisfied, it implies the global stability of RMPflow.

Theorem 2. *Suppose every leaf node is a GDS with a metric matrix in the form* $\mathbf{R}(\mathbf{x}) + \mathbf{L}(\mathbf{x})^\top \mathbf{D}(\mathbf{x}, \dot{\mathbf{x}})\mathbf{L}(\mathbf{x})$ *for differentiable functions* $\mathbf{R}$*,* $\mathbf{L}$*, and* $\mathbf{D}$ *satisfying*

$$\mathbf{R}(\mathbf{x}) \succeq 0, \qquad \mathbf{D}(\mathbf{x}, \dot{\mathbf{x}}) = \operatorname{diag}((d_i(\mathbf{x}, \dot{y}_i))_{i=1}^n) \succeq 0, \qquad \dot{y}_i \partial_{\dot{y}_i} d_i(\mathbf{x}, \dot{y}_i) \geq 0$$

where $\mathbf{x}$ *is the coordinate of the leaf-node manifold and* $\dot{\mathbf{y}} = \mathbf{L}\dot{\mathbf{x}} \in \mathbb{R}^n$*. It holds* $\mathbf{\Xi}_{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}}) \succeq 0$*. If further* $\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}), \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}}) \succ 0$*, then* $\mathbf{M} \in \mathbb{R}^{d \times d}_{++}$*, and the global RMP generated by RMPflow converges to the forward invariant set* $\mathcal{C}_\infty$ *in Corollary 2.*

A particular condition in Theorem 2 is when all the leaf nodes with velocity dependent metric are 1D. Suppose $x \in \mathbb{R}$ is its coordinate and $g(x, \dot{x})$ is its metric matrix. The sufficient condition essentially boils down to $g(x, \dot{x}) \geq 0$ and $\dot{x}\partial_{\dot{x}} g(x, \dot{x}) \geq 0$. This means that, given any $x \in \mathbb{R}$, $g(x, 0) = 0$, $g(x, \dot{x})$ is non-decreasing when $\dot{x} > 0$, and non-increasing when $\dot{x} < 0$. This condition is satisfied by the collision avoidance policy in Section 5.6.3.

5.8 Applying RMPflow on a simple example problem

In this section, we use a simple 2-DOF planar manipulator as shown in Figure 5.2a tasked with avoiding an obstacle and making its end-effector reach a goal, to illustrate how to go about constructing an RMP-tree and relevant RMPs given a problem.

The overall task is decomposed into several subtasks, like making the end-effector reach the goal and avoiding the obstacle at various locations on the robot (including the end-effector) such that these locations well approximate the robot's body. We begin by

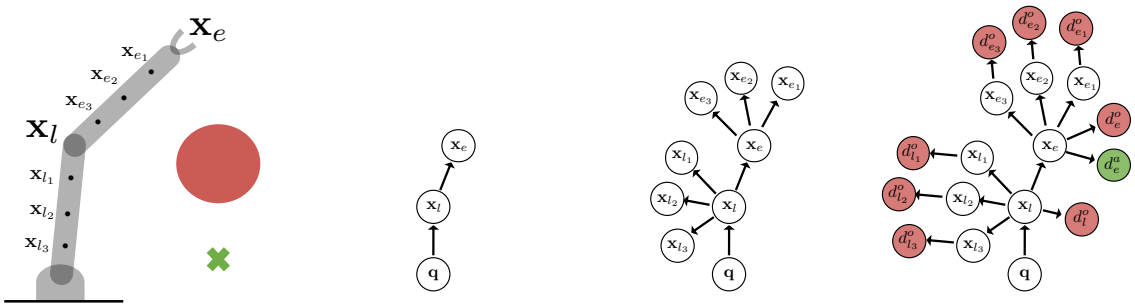


Figure 5.2: (a) A 2-DOF planar manipulator tasked with avoiding an obstacle (red) and reaching a goal (green). Task-map tree for the problem in (a) being built starting from (b) the main kinematic chain to (c) all kinematic body point locations to (d) full task-map tree consisting of all abstract task spaces for obstacle avoidance (red) and target attraction (green).

building a task-map tree consisting of the main kinematic chain of the robot as shown in Figure 5.2b. The root of the tree is the configuration space of the robot $\mathbf{q}$, where both joint angles ($\mathbf{q}_1$ and $\mathbf{q}_2$) are available and where the desired overall joint acceleration policy needs to be evaluated to achieve the task. The main kinematic chain goes from the configuration space to the position space of the elbow $\mathbf{x}_l$ that is calculated with a forward map $(\mathbf{T}_l, \mathbf{q}_2) = \psi_l(\mathbf{T}_b, \mathbf{q}_1, \mathbf{q}_2)$, and then from the elbow to the end-effector position space $\mathbf{x}_e$ that is calculated with a forward map $\mathbf{T}_e = \psi_e(\mathbf{T}_l, \mathbf{q}_2)$. Here, $\mathbf{T}$ are homogeneous transformation matrices and $\mathbf{T}_b$ represents the base frame of the robot.

From the main chain we branch out the tree to task spaces that specify body points of interest (that well approximate the body for collision avoidance) as shown in Figure 5.2c. In this example, we use three body points along each link where their respective forward maps are just linear offsets from those link frames, $\mathbf{x}_{l_i}$ from $\mathbf{x}_l$ and $\mathbf{x}_{e_i}$ from $\mathbf{x}_e$, where $i = 1, 2, 3$. Next, we grow the tree from robot kinematics to abstract task-spaces where we will define appropriate sub-task RMPs to achieve the desired task. In our example, these abstract task spaces are distance spaces to the obstacle or the goal as shown in Figure 5.2d. On every node associated with a body point on the robot we attach a task space d_i^o where the associated forward map $d_i^o = \|\mathbf{x}_i - \mathbf{x}_o\|_2 - r$, defines the distance of the body point

i to the obstacle (at $\mathbf{x}_o$ with radius r). Note that in the presence of multiple obstacle we attach multiple such task spaces that define the distance of that body point to each of those obstacles. Finally, we attach a task space d_e^a with forward map $d_e^a = \|\mathbf{x}_e - \mathbf{x}_a\|_2$ that defines the distance of the end-effector to the target attractor at $\mathbf{x}_a$.

To solve our example problem we place obstacle avoidance RMPs on every task space d_i^o as described in Section 5.6.3 with a position-velocity dependent metric (a barrier function for the position part and a negative half-gate for the velocity part) and a barrier function for the potential. We also place an attractor RMP as described in Section 5.6.3 with position dependent metric that stretches the task space in the direction of the goal and a quadratic potential function. Once the task relevant RMPs are defined on the leaf nodes of the tree we apply RMPflow to evaluate the desired policy at any time step. In practice, we can extend this construction in a similar fashion to robots with more joints by starting with its respective main kinematic chain as well incorporating other tasks space for tasks like joint limit avoidance.

5.9 Evaluation

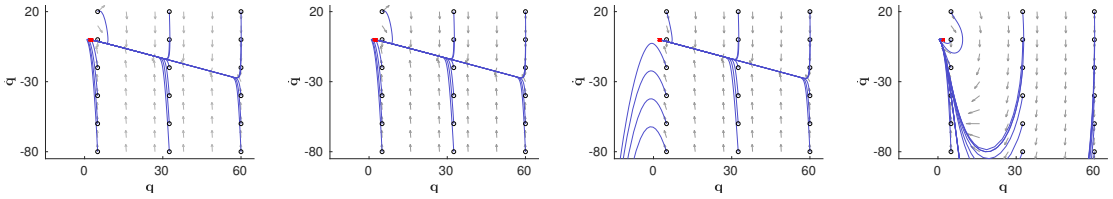
We perform controlled experiments to study the curvature effects of nonlinear metrics, which is important for stability and collision avoidance. We then perform several full-body experiments[8] to demonstrate the capabilities of RMPflow on high-DOF manipulation problems in clutter, and implement an integrated vision-and-motion system on two physical robots.

5.9.1 Controlled experiments

1D example

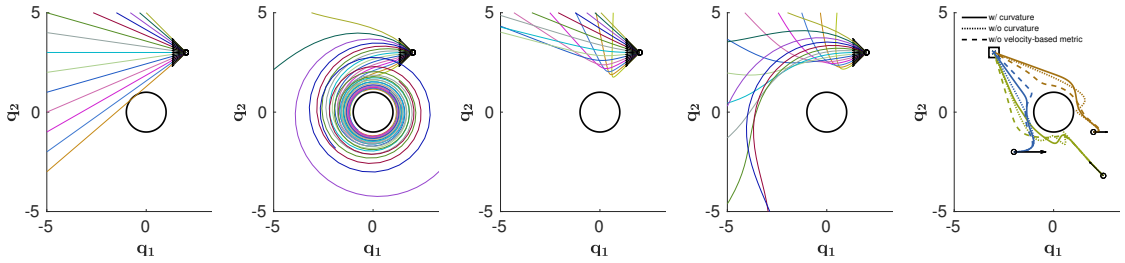
Let $\mathbf{q} \in \mathbb{R}$. We consider a barrier-type task map $\mathbf{x} = 1/\mathbf{q}$ and define a GDS in Eq. (5.20) with $\mathbf{G} = 1$, $\Phi(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^2$, and $\mathbf{B} = (1 + 1/\mathbf{x})$, where $\mathbf{x}_0 > 0$. Using the GDS,

[8] A video of experiments is available at `https://youtu.be/Fl4WvsXQDzo`



(a) (b) (c) (d)

Figure 5.3: Phase portraits (gray) and integral curves (blue; from black circles to red crosses) of 1D example. (a) Desired behavior. (b) With curvature terms. (c) Without curvature terms. (d) Without curvature terms but with nonlinear damping.



(a) (b) (c) (d) (e)

Figure 5.4: 2D example; initial positions (small circle) and velocities (arrows). (a-d) Obstacle (circle) avoidance: (a) w/o curvature terms and w/o potential. (b) w/ curvature terms and w/o potential. (c) w/o curvature terms and w/ potential. (d) w/ curvature terms and w/ potential. (e) Combined obstacle avoidance and goal (square) reaching.

we can define an RMP $[-\nabla_{\mathbf{x}}\Phi - \mathbf{B}\dot{\mathbf{x}} - \boldsymbol{\xi}_{\mathbf{G}}, \mathbf{M}]^{\mathbb{R}}$, where $\mathbf{M}$ and $\boldsymbol{\xi}_{\mathbf{G}}$ are defined according to Section 5.7. We use this example to study the effects of $\dot{\mathbf{J}}\dot{\mathbf{q}}$ in `pullback` Eq. (5.18), where we define $\mathbf{J} = \partial_{\mathbf{q}}\mathbf{x}$. Figure 5.3 compares the desired behavior (Figure 5.3a) and the behaviors of correct/incorrect `pullback`. If `pullback` is performed correctly with $\dot{\mathbf{J}}\dot{\mathbf{q}}$, the behavior matches the designed one (Figure 5.3b). By contrast, if $\dot{\mathbf{J}}\dot{\mathbf{q}}$ is ignored, the observed behavior becomes inconsistent and unstable (Figure 5.3c). While the instability of neglecting $\dot{\mathbf{J}}\dot{\mathbf{q}}$ can be recovered with a damping $\mathbf{B} = (1 + \frac{\dot{\mathbf{x}}^2}{\mathbf{x}})$ nonlinear in $\dot{\mathbf{x}}$ (suggested in [143]), the behavior remains inconsistent (Figure 5.3d).

2D example

We consider a 2D goal-reaching task with collision avoidance and study the effects of velocity dependent metrics. First, we define an RMP (a GDS as in Section 5.6.3) in $\mathbf{x} = d(\mathbf{q})$

(the 1D task space of the distance to the obstacle). We pick a metric $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) = w(\mathbf{x})u(\dot{\mathbf{x}})$, where $w(\mathbf{x}) = 1/\mathbf{x}^4$ increases if the particle is *close* to the obstacle and $u(\dot{\mathbf{x}}) = \epsilon + \min(0, \dot{\mathbf{x}})\dot{\mathbf{x}}$ (where $\epsilon \geq 0$), increases if it moves *towards* the obstacle. As this metric is non-constant, the GDS has curvature terms $\boldsymbol{\Xi}_{\mathbf{G}} = \frac{1}{2}\dot{\mathbf{x}}w(\mathbf{x})\partial_{\dot{\mathbf{x}}}u(\dot{\mathbf{x}})$ and $\boldsymbol{\xi}_{\mathbf{G}} = \frac{1}{2}\dot{\mathbf{x}}^2u(\dot{\mathbf{x}})\partial_{\mathbf{x}}w(\mathbf{x})$. These curvature terms along with $\dot{\mathbf{J}}\dot{\mathbf{q}}$ produce an acceleration that lead to natural obstacle avoidance behavior, coaxing the system toward isocontours of the obstacle (Figure 5.4b). On the other hand, when the curvature terms are ignored, the particle travels in straight lines with constant velocity (Figure 5.4a). To define the full collision avoidance RMP, we introduce a barrier-type potential $\Phi(\mathbf{x}) = \frac{1}{2}\alpha w(\mathbf{x})^2$ to create extra repulsive forces, where $\alpha \geq 0$. A comparison of the curvature effects in this setting is shown in Figure 5.4c and 5.4d (with $\alpha = 1$). Next, we use RMPflow to combine the collision avoidance RMP above (with $\alpha = 0.001$) and an attractor RMP. Let $\mathbf{q}_g$ be the goal. The attractor RMP is a GDS in the task space $\mathbf{y} = \mathbf{q} - \mathbf{q}_g$ with a metric $w(\mathbf{y})\mathbf{I}$, a damping $\eta w(\mathbf{y})\mathbf{I}$, and a potential that is zero at $\mathbf{y} = 0$, where $\eta > 0$. Figure 5.4e shows the trajectories of the combined RMP. The combined non-constant metrics generate a behavior that transitions smoothly towards the goal while heading away from the obstacle. When the curvature terms are ignored (for both RMPs), the trajectories oscillate near the obstacle. In practice, this can result in jittery behavior on manipulators. When the metric is not velocity-based ($\mathbf{G}(\mathbf{x}) = w(\mathbf{x})$) the behavior is less efficient in breaking free from the obstacle to go toward the goal.

5.9.2 System experiments

Task-map tree structure

Figure 5.5 depicts the tree of task maps used in the full-robot experiments. The chosen structure emphasizes potential for parallelization over fully exploiting the recursive nature of the kinematic chain, treating each link frame as just one forward kinematic map step from the configuration space. We could possibly save some computation by defining the forward kinematic maps recursively as $(\mathbf{T}_{i+1}, \mathbf{q}_{i+1}, \ldots, \mathbf{q}_d) = \psi_i(\mathbf{T}_i, \mathbf{q}_i, \ldots, \mathbf{q}_d)$. The

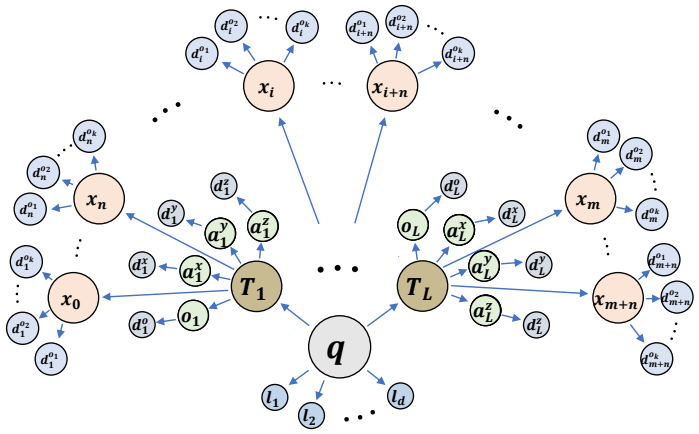


Figure 5.5: Task-map tree used in the system experiments.

configuration space $\mathbf{q}$ is linked to L link frames $\mathbf{T}_1, \ldots, \mathbf{T}_L$ through the robot's forward kinematics (the details of tasks will be described later on for each individual experiment). Each frame has 4 frame element spaces: the origin o_i and each of the axes $\mathbf{a}_i^x, \mathbf{a}_i^y, \mathbf{a}_i^z$, with corresponding distance spaces to targets $d_i^o, d_i^x, d_i^y, d_i^z$ (if they are active). Additionally, there are a number of obstacle control points $\mathbf{x}_j$ distributed across each of the links, each with k associated distance spaces $d_j^{o_1}, \ldots, d_j^{o_k}$, one for each obstacle $o_1, \ldots, o_k$. Finally, for each dimension of the configuration space there's an associated joint limit space $l_1, \ldots, l_d$.

Reaching-through-clutter experiments

We set up a collection of clutter-filled environments with cylindrical obstacles of varying sizes in simulation as depicted in Figure 5.6, and tested the performance of RMPflow and two potential field methods on a modeled ABB YuMi robot.

Compared methods:

1. RMPflow: We implement RMPflow using the RMPs in Section 5.6.3 and detailed in [138]. In particular, we place collision-avoidance controllers on distance spaces $s_{ij} = d_j(\mathbf{x}_i)$, where $j = 1, \ldots, m$ indexes the world obstacle o_j and $i = 1, \ldots, n$ indexes the n control point along the robot's body. Each collision-avoidance con-

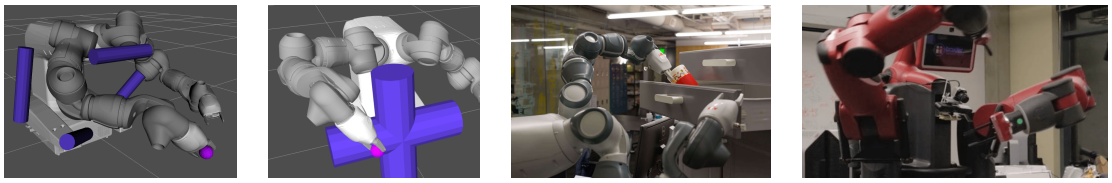
troller uses a weight function $w_o(\mathbf{x})$ that ranges from 0 when the robot is far from the obstacle to $w_o^{\max} \gg 0$ when the robot is in contact with the obstacle's surface. Similarly, the attractor potential uses a weight function $w_a(\mathbf{x})$ that ranges from $w_a^{\min}$ far from the target to $w_a^{\max}$ close to the target.

2. PF-basic: This variant is a basic implementation of obstacle avoidance potential fields with dynamics shaping. We use the RMP framework to implement this variant by placing collision-avoidance controllers on the same body control points used in RMPflow but with isotropic metrics of the form $\mathbf{G}_o^{\text{basic}}(\mathbf{x}) = w_o^{\max}\mathbf{I}$ for each control point, with $w_o^{\max}$ matching the value RMPflow uses. Similarly, the attractor uses the same attractor potential as RMPflow, but with a constant isotropic metric with the form $\mathbf{G}_a^{\text{basic}}(\mathbf{x}) = w_a^{\max}\mathbf{I}$.

3. PF-nonlinear: This variant matches PF-basic in construction, except it uses a *non-linear* isotropic metrics of the form $\mathbf{G}_o^{\text{nlin}}(\mathbf{x}_i) = w_o(\mathbf{x})\mathbf{I}$ and $\mathbf{G}_a^{\text{nlin}}(\mathbf{x}_i) = w_a(\mathbf{x})\mathbf{I}$ for obstacle-avoidance and attraction, respectively, using weight functions matching RMPflow.

A note on curvature terms: PF-basic uses constant metrics, so has no curvature terms; PF-nonlinear has nontrivial curvature terms arising from the spatially varying metrics, but we ignore them here to match common practice from the OSC literature.

Parameter scaling of PF-basic: Isotropic metrics do not express spacial directionality toward obstacles, and that leads to an inability of the system to effectively trade off the competing controller requirements. That conflict results in more collisions and increased instability. We, therefore, compare PF-basic under these baseline metric weights (matching RMPflow) with variants that incrementally strengthen collision avoidance controllers and C-space postural controllers ($f_{\mathcal{C}}(\mathbf{q}, \dot{\mathbf{q}}) = \gamma_p(\mathbf{q}_0 - \mathbf{q}) - \gamma_d\dot{\mathbf{q}}$) to improve these performance measures in the experiment. We use the following weight scalings (first entry denotes the obstacle metric scalar, and the second entry denotes the C-space metric scalar): "low"



simulated worlds real-world experiments

Figure 5.6: Two of the six simulated worlds in the reaching experiments (left), and the two physical dual-arm platforms in the full system experiment (right).

$(3, 10)$, "med" $(5, 50)$, and "high" $(10, 100)$.

Environments: We run each of these variants on 6 obstacle environments with 20 randomly sampled target locations each distributed on the opposite side of the obstacle field from the robot. Three of the environments use four smaller obstacles (Figure 5.6 left), and the remaining three environments used two large obstacles (Figure 5.6 middle left). Each environment used the same 20 targets to avoid implicit sampling bias in target choice.

Performance measures: We report results in Figure 5.7 in terms of mean and one standard deviation error bars calculated across the 120 trials for each of the following performance measures:[9]

1. *Time to goal ("time"):* Length of time, in seconds, it takes for the robot to reach a convergence state. This convergence state is either the target, or its best-effort local minimum. If the system never converges, as in the case of many potential field trials for infeasible problems, the trial times out after 5 seconds. This metric measures time-efficiency of the movement.

2. *C-space path length ("length"):* This is the total path length $\int \|\dot{\mathbf{q}}\| dt$ of the movement through the configuration space across the trial. This metric measures how economical the movement is. In many of the potential-field variants with lower weights, we see significant fighting among the controllers resulting in highly inefficient extraneous motions.

[9]There is no guarantee of feasibility in planning problems in general, so in all cases, we measure performance relative to the performance of RMPflow, which is empirically stable and near optimal across these problems.

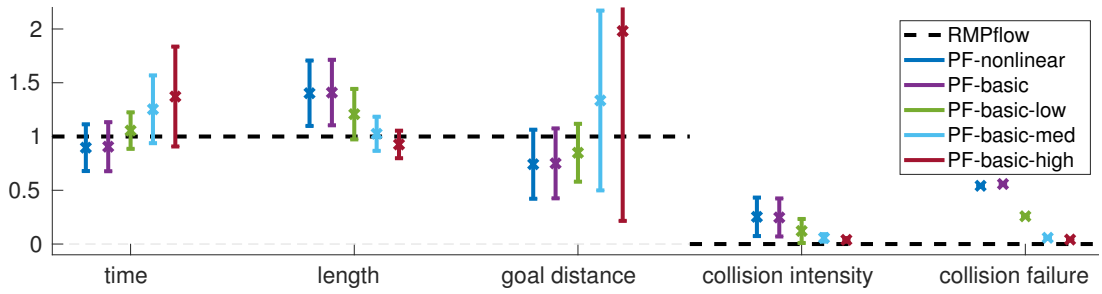


Figure 5.7: Results for reaching experiments. Though some methods achieve a shorter goal distance than RMPflow in successful trials, they end up in collision in most the trials.

3. *Minimal achievable distance to goal ("goal distance"):* Measures how close, in meters, the system is able to get to the goal with its end-effector.

4. *Percent time in collision for colliding trials ("collision intensity"):* Given that a trial has a collision, this metric measures the fraction of time the system is in collision throughout the trial. This metric indicates the intensity of the collision. Low values indicate short grazing collisions while higher values indicate long term obstacle penetration.

5. *Fraction of trails with collisions ("collision failure"):* Reports the fraction of trials with any collision event. We consider these to be collision-avoidance controller failures.

Analysis: In Figure 5.7, we see that RMPflow outperforms each of these variants significantly, with some informative trends:

1. RMPflow never collides, so its collision intensity and collision failure values are 0.

2. The other techniques, progressing from no scaling of collision-avoidance and C-space controller weights to substantial scaling, show a profile of substantial collision in the beginning to fewer (but still non-zero) collision events in the end. But we note that improvement in collision-avoidance is achieved at the expense of time-efficiency and the robot's ability to reach the goal (it is too conservative).

3. Lower weight scaling of both PF-basic and PF-nonlinear actually achieve some faster times and better goal distances, but that is because the system pushes directly through obstacles, effectively "cheating" during the trial. RMPflow remains highly economical with its best effort reaching behaviors while ensuring the trials remain collision-free.

4. Lower weight scalings of PF-basic are highly uneconomical in their motion reflective of their relative instability. As the C-space weights on the posture controllers increase, the stability and economy of motion increase, but, again, at the expense of time-efficiency and optimality of the final reach.

5. There is little empirical difference between PF-basic and PF-nonlinear indicating that the defining feature separating RMPflow from the potential field techniques is its use of a highly nonlinear metric that explicitly stretches the space in the direction of the obstacle as well as in the direction of the velocity toward the target. Those stretchings penalize deviations in the stretched directions during combination with other controllers while allowing variation along orthogonal directions. By being more explicit about how controllers should instantaneously trade off with one another, RMPflow is better able to mitigate the otherwise conflicting control signals.

Summary: Isotropic metrics do not effectively convey how each collision and attractor controller should trade off with one another, resulting in a conflict of signals that obscure the intent of each controller making simultaneous collision avoidance, attraction, and posture maintenance more difficult. Increasing the weights of the controllers can improve their effectiveness, but at the expense of decreased overall system performance. The resulting motions are slower and less effective in reaching the goal in spite of more stable behavior and fewer collisions. A key feature of RMPflow is its ability to leverage highly nonlinear metrics that better convey information about how controllers should trade off with one another, while retaining provable stability guarantees. In combination, these features result in

Figure 5.8: From left to right using RMPflow the YuMi robot opens a drawer with one arm and with the other arm picks and places a banana in that drawer.

efficient and economical obstacle avoidance behavior while reaching toward targets amid clutter.

System integration for real-time reactive planing

We demonstrate the integrated vision and motion system on two physical dual arm manipulation platforms: a Baxter robot from Rethink Robotics, and a YuMi robot from ABB. Footage of our fully integrated system (see start of Section 4.4 for the link) depicting tasks such as pick and place amid clutter, reactive manipulation of a cabinet drawers and doors with human interaction, *active* leadthrough with collision controllers running, and pick and place into a cabinet drawer (see Figure 5.8).

This full integrated system, uses the RMPs described in Section 5.6.3 and in [138] with a slight modification that the curvature terms are ignored. Instead, we maintain theoretical stability by using sufficient damping terms as described in Section 5.9.1 and by operating at slower speeds. Generalization of these RMPs between embodiments was anecdotally pretty consistent, although, as we demonstrate in our experiments, we would expect more empirical deviation at higher speeds. For these manipulation tasks, this early version of the system worked well as demonstrated in the video.

For visual perception, we leveraged consumer depth cameras along with two levels of perceptual feedback:

1. *Ambient world:* For the Baxter system we create a voxelized representation of the unmodeled ambient world, and use distance fields to focus the collision controllers on just the closest obstacle points surrounding the arms. This methodology is similar in nature to [136], except we found empirically that attending to only the closest

point to a skeleton representation resulted in oscillation in concaved regions where distance functions might result in nonsmooth kinks. We mitigate this issue by finding the closest points to a *volume* around each control point, effectively smoothing over points of nondifferentiability in the distance field.

2. *Tracked objects:* We use the Dense Articulated Real-time Tracking (DART) system of [155] to track articulated objects in real time through manipulations. This system is able to track both the robot and environmental objects, such as an articulated cabinet, simultaneously to give accurate measurements of their relative configuration effectively obviating the need for explicit camera-world calibration. As long as the system is initialized in the general region of the object locations (where for the cabinet and the robot, that would mean even up to half a foot of error in translation and a similar scale of error in rotation), the DART optimizer will snap to the right configuration when turned on. DART sends information about object locations to the motion generation, and receives back information about expected joint configurations (priors) from the motion system generating a robust world representation usable in a number of practical real-world manipulation problems.

Each of our behaviors are decomposed as state machines that use visual feedback to detect transitions, including transitions to reaction states as needed to implement behavioral robustness. Each arm is represented as a separate robot for efficiency, receiving real-time information about other arm's current state enabling coordination. Both arms are programmed simultaneously using a high level language that provides the programmer a unified view of the surrounding world and command of both arms.

5.10 Discussion

We propose an efficient policy synthesis framework, RMPflow, for generating policies with non-Euclidean behavior, including motion with velocity dependent metrics that are new to

the literature. In design, RMPflow is implemented as a computational graph, which can geometrically consistently combine subtask policies into a global policy for the robot. In theory, we provide conditions for stability and show that RMPflow is intrinsically coordinate-free. In the experiments, we demonstrate that RMPflow can generate smooth and natural motion for various tasks, when proper subtask RMPs are specified. We have also extended this framework to multi-agent systems, where formation behaviors can also additionally be encoded [156] in both centralized and decentralized settings.

While here we focus on the special case of RMPflow with GDSs, this family already covers a wide range of reactive policies commonly used in practice. For example, when the task metric is Euclidean (i.e. constant), RMPflow recovers OSC (and its variants) [133, 142, 134, 135, 143]. When the task metric is only configuration dependent, RMPflow can be viewed as performing energy shaping to combine multiple SMSs in geometric control [137]. Further, RMPflow allows using velocity dependent metrics, generating behaviors all those previous rigid mechanics-based approaches fail to model. We also note that RMPflow can be easily modified to incorporate exogenous time-varying inputs (e.g. forces to realize impedance control [141] or learned perturbations as in DMPs [150]). In computation, the structure of RMPflow in natural-formed RMPs resembles the classical Recursive Newton-Euler algorithm [140, 157] (see Appendix D). Alternatively, the canonical form of RMPflow in Eq. (5.19) resembles Gauss' Principle [134, 135], but with a curvature correction $\boldsymbol{\Xi}_{\mathbf{G}}$ on the inertia matrix (suggested by Theorem 1) to account for velocity dependent metrics. Thus, we can view RMPflow as a natural generalization of these approaches to a broader class of non-Euclidean behaviors.

Limitations

With the help of velocity (and position) dependent metrics we are able to design behaviors that can get close to planning-like without incurring the relatively larger computational

overhead of planning. However, the performance relies primarily on how well the RMPs can be designed and therefore presents a major challenge for non-expert practitioners. We begin to address some of these challenges with the help of learning in the next chapters. Another consequence of imperfect design is that it can get stuck in local minima without having access to long term objectives. Our framework can currently only support fully actuated systems without any actuation or dynamic constraints. For our current experiments this was sufficient, but further research is necessary to adapt the presented framework such that it can incorporate constraints like contact, nonholonomicity, or dynamics and is applicable to high-speed mobile manipulation problems. Our theoretical analysis with respect to stability is currently limited to the system's kinematics as opposed to stability of the dynamics of the system as seen in many OSC and geometric control formulations. OSC and geometric control also provide formal analysis on controllability which we have not currently established in our work. Our method also currently relies on perfect state information that can be provided from a reliable perception system since it cannot internally handle uncertainty.

Part IV

Learning on Task-Map Trees

CHAPTER 6
LEARNABLE POLICY FUSION

6.1 Introduction and related work

In the previous chapter, we showed the capabilities of RMPflow. However, practical usage difficulties still remain to be addressed. Particularly, the user must provide RMPs with matrix functions that properly describe the characteristics of the correspondent subtask motion policies in order to build an effective RMPflow system. Otherwise, the final global policy may have unsatisfactory performance, though still being geometrically consistent (with respect to some unreasonable geometric structure). This poses a challenge for practitioners who are inexperienced in control/dynamical systems, or for designing policies of unstructured tasks where the full state is hard to describe.

In this chapter, we leverage the task-map tree structure presented in Chapter 5 and introduce a hierarchical energy reshaping scheme into RMPflow to remedy the requirement of providing high-quality subtasks RMPs from the user. The modified algorithm, called RMPfusion, adds a set of multiplicative weight *functions* in the policy fusion step of RMPflow, which can be manually parametrized or modeled by function approximators (like neural networks) that can be learned. An immediate benefit of our new algorithm is the extra design and parameterization flexibilities added to RMPflow. These weight functions do not just linearly combine motion policies [158, 159], but *hierarchically* reshape the inherent kinematic and potential energies of RMPflow, overall creating a nonlinear effect on the global policy it outputs. Effectively RMPfusion adapts between multiple versions of RMPflow according to the robot's configuration and the environment, so it can work with imperfect subtask RMPs from the user, which the vanilla approach fails to handle.

We prove that RMPfusion inherits the Lyapunov-type stability from RMPflow as long

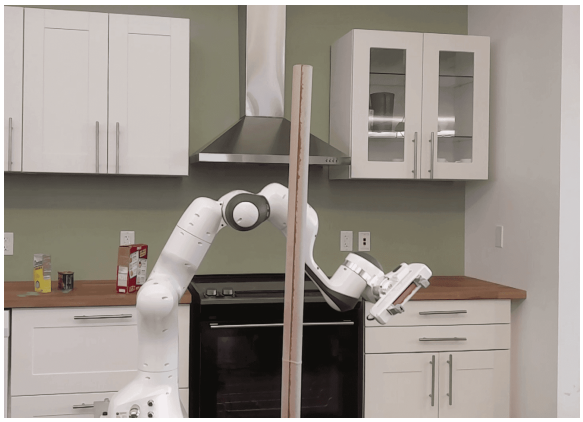


Figure 6.1: Franka robot navigating around an obstacle using RMPfusion.

as the weight functions are positive. That is, RMPfusion remains stable under almost arbitrary choice of weight functions. As a result, RMPfusion can be treated as a class of structured parameterized policies, which are suitable for learning with safety constraints and prior information. We show that the new policy fusion step is differentiable and therefore any parameterized weight functions can be conveniently learned in an end-to-end fashion. To corroborate our theoretical analysis, we verify properties of RMPfusion in imitation learning tasks, in both simulations and on a real-world robot (Figure 6.1). We show that RMPfusion can learn to mimic the expert policy when sufficient demonstrations are provided, and importantly, it always yields stable policies even during the immature phase of learning.

6.2 Modifying RMPflow with multiplicative weight functions

RMPflow provides a control-theoretic framework for combining subtask policies. However, certain limitations exist. Particularly, it requires the user to provide sensible inertia matrices (cf. section 5.5) to describe the subtask policies' characteristics in the leaf-nodes RMPs; failing to do so may result in a global policy with undesirable performance, albeit still being geometrically consistent with the meaningless geometric structure induced by the bad inertia matrices. This can be challenging especially for users inexperienced in control/dynamical systems, or when designing policies that use less structured information

(e.g. camera image).

In this work, we propose a modified algorithm, RMPfusion, which adds extra flexibilities into RMPflow to address this difficulty. The main idea is to introduce an additional set of weight functions as gates to switch on and off the child-node policies in an RMP-tree, based on the current state of the robot and the environment. These functions can either be designed by hand, or be parameterized as function approximators (like neural networks) which are then learned end-to-end from data (see section 6.3). As a result, RMPfusion can combine imperfect subtask RMPs into a better global policy, thereby lessening the burden on the user to provide high-quality subtasks RMPs.

RMPfusion adds new features to the RMP-tree and RMP-algebra in RMPflow as RMP-tree* and RMP-algebra*, respectively. RMP-tree* shares the same tree structure of RMP-tree, but augments each node with extra information and each edge with a weight function. RMP-algebra* consists of `pushforward` and `resolve` from RMP-algebra and a modified backward operator `pullback`*. Below we define these modifications. In addition, we show that RMPfusion retains the nice structural properties of RMPflow: under mild conditions on the weights, the global policy of RMPfusion can retain Lyapunov-type stability, as in Section 5.7. Later in Section 6.3, we will show how to learn the weight functions in RMPfusion from data.

*RMP-tree**

In addition to the RMP and its state, each node in RMP-tree* also stores the values of a scalar function L (called the Lagrangian) and the metric matrix $\mathbf{G}$. When a leaf-node RMP is a GDS, $\mathbf{G}$ is defined as Eq. (5.20) and $L = \frac{1}{2}\dot{\mathbf{x}}^\top \mathbf{G}\dot{\mathbf{x}} - \Phi(\mathbf{x})$.

Each edge in an RMP-tree* has a weight function in addition to the transformation map between manifolds in the RMP-tree. This weight is a function of the parent-node configuration and some auxiliary state (which is additional information to describe the task at hand, such as the location of the goal in a reaching task), and it can be specified manually

or learned from data.

`pullback`* *in RMP-algebra**

We modify `pullback` into `pullback`* so it can uses the weight functions on edges of the RMP-tree* to combine child-node RMPs. This new operator is defined below: for the parent and child nodes given in Eq. (5.18), we set

$$\begin{aligned} \mathbf{f} &= \textstyle\sum_{i=1}^{K} w_i \mathbf{J}_i^\top (\mathbf{f}_i - \mathbf{M}_i \dot{\mathbf{J}}_i \dot{\mathbf{x}}) + \mathbf{h}_i \\ \mathbf{M} &= \textstyle\sum_{i=1}^{K} w_i \mathbf{J}_i^\top \mathbf{M}_i \mathbf{J}_i \\ \mathbf{G} &= \textstyle\sum_{i=1}^{K} w_i \mathbf{J}_i^\top \mathbf{G}_i \mathbf{J}_i \\ L &= \textstyle\sum_{i=1}^{K} w_i L_i \end{aligned} \tag{6.1}$$

where $\mathbf{h}_i = L_i \nabla_{\mathbf{x}} w_i - (\dot{\mathbf{x}}^\top \nabla_{\mathbf{x}} w_i) \mathbf{J}_i^\top \mathbf{G}_i \mathbf{J}_i \dot{\mathbf{x}}$. From Eq. (6.1), we see that `pullback`* does *not* simply linearly combine child-node motion policies. It adds a correction term $\mathbf{h}_i$, which is designed to anticipates the change of weighting w_i so that the system remains stable. This turns out to be a form of *hierarchical* energy reshaping as discussed later in Section 6.2.

Stability

We show RMPfusion is also Lyapunov stable like RMPflow. To state the stability property, let us introduce additional notation to precisely describe the functions in the RMP-tree*. We will use $(i; j)$ to denote the ith node in depth j of an RMP-tree* and we use $C_{(i;j)}$ to denote the indices of its child nodes. For example, node $(1; 0)$ denotes the root node, which we will also write as node r for short. In addition, we will refer to the functions on the edges using the indices of the child nodes, as each node in the RMP-tree* only has one parent node. For example, the Jacobian of the transformation to the ith node in depth j is denoted as $\mathbf{J}_{(i;j)}$.

We show the stability property of RMPfusion when all the leaf nodes are of GDSs. The proof is given in Appendix E.

Theorem 3. *Suppose an RMP-tree* has leaf-node policies as GDSs with energy functions given as* $V(\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2}\dot{\mathbf{x}}^\top \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} + \Phi(\mathbf{x})$. *Define the energy function* $V_{(i;j)}$*, damping matrix* $\mathbf{B}_{(i;j)}$*, and potential* $\Phi_{(i;j)}$ *on the tree through the recursion*

$$\begin{aligned} V_{(i;j)} &= \textstyle\sum_{k \in C_{(i;j)}} w_{(k;j+1)} V_{(k;j+1)} \\ \mathbf{B}_{(i;j)} &= \textstyle\sum_{k \in C_{(i;j)}} w_{(k;j+1)} \mathbf{J}_{(k;j+1)}^\top \mathbf{B}_{(k;j+1)} \mathbf{J}_{(k;j+1)} \\ \Phi_{(i;j)} &= \textstyle\sum_{k \in C_{(i;j)}} w_{(k;j+1)} \Phi_{(k;j+1)} \end{aligned} \tag{6.2}$$

in which the boundary condition is given by the leaf-node GDSs. Let V_r *be a Lyapunov candidate.*

1. *If* $\mathbf{M}_r \succ 0$*, then* $\dot{V}_r = -\dot{\mathbf{q}}^\top \mathbf{B}_r \dot{\mathbf{q}} \leq 0$.
2. *If further* $\mathbf{G}_r, \mathbf{B}_r \succ 0$*, then the system converges to the forward invariant set* $\mathcal{C}_\infty = \{(\mathbf{q}, \dot{\mathbf{q}}) : \nabla_{\mathbf{q}} \Phi_r(\mathbf{q}) = 0, \dot{\mathbf{q}} = 0\}$.

where we recall the subscript $_r$ *stands for the root node.*

Theorem 3 shows that the system is Lyapunov stable with respect to the energy V_r. To satisfy the conditions required in Theorem 3, a sufficient condition is to select leaf-node GDSs with certain monotone metrics 2 and positive weight functions. Therefore, in addition to the conditions needed by RMPflow, RMPfusion only imposes mild constraints on the choice of weight functions. This is a useful feature when the weight functions are learned from data, because Theorem 3 essentially guarantees the output policy is always stable even in the middle and premature stage of learning.

Note that it is straightforward to extend RMP-tree* and the above analysis to include, in Eq. (6.1), an extra time-varying forcing term that vanishes as $t \to \infty$ (like the one used in DMPs [150]) and to consider time-varying potential functions (e.g. in tracking applications).

Advantages of RMPfusion over RMPflow

RMPfusion is a strict generalization of RMPflow. In the special case where each weight is constant one, RMPfusion becomes RMPflow (i.e. `pullback*` is the same as `pullback` and Theorem 3 reduces to Theorem 1). More generally, RMPfusion allows mixing local policies through reweighting their energy functions, while retaining the nice structural properties of RMPflow, as shown in Theorem 3.

In comparison, RMPfusion has a more flexible way to express policies and compose the subtask energy functions into the Lyapunov candidate V_r in Eq. (6.3). Whereas Theorem 1 uses the simple summation of subtask energies $V_r = \sum_{i=1}^{K} V_i$, Theorem 3 uses the energy function

$$V_r = \sum_{k_1 \in C_{(1;0)}} w_{(k_1;1)} \sum_{k_2 \in C_{(k_1;1)}} \cdots \sum_{k_D \in C_{(k_{D-1};D-1)}} w_{(k_D;D)} V_{(k_D;D)} \tag{6.3}$$

for a depth-D RMP-tree* (cf. Eq. (6.2)) and each weight $w_{(i;j)}$ can be a function of the configuration and auxiliary state of the parent of node $(i; j)$. Therefore, from Eq. (6.2) and Eq. (6.3), RMPfusion can be viewed as a form of hierarchical energy reshaping scheme along the hierarchy structure naturally provided by the RMP-tree*. Consequently, the recursive formulation of RMPfusion allows the user only to provide basic subtask policies on the leaf nodes, because the expressiveness of those policies will be amplified by the weight functions whenever they pass through `pullback*`. Conversely, using RMPflow requires the user to provide a set subtask policies with complicated behaviors.

We use an example to illustrate the extra flexibility offered by RMPfusion. Consider a simple Y-shape RMP-tree* with a root node and two child nodes with weight functions w_1 and w_2. For the child nodes, suppose they are GDS $(\mathcal{N}_i, \mathbf{G}_i, \mathbf{B}_i, \Phi_i)$ and have coordinate $\mathbf{y}_i$, for $i = 1, 2$. For simplicity, let us assume $\mathbf{G}_i$ only depends on the configuration $\mathbf{y}_i$. From Theorem 3, we see that the root node has an energy function $V_r = \frac{1}{2}\dot{\mathbf{q}}^\top \mathbf{G}_r \dot{\mathbf{q}} + \Phi_r$,

where

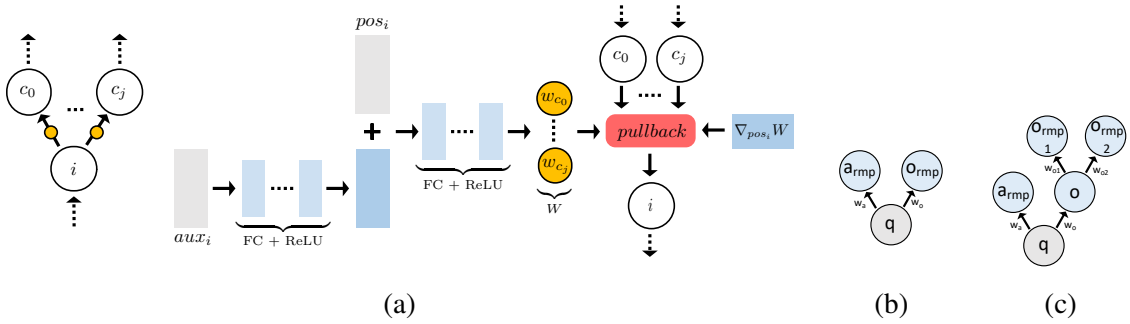
$$\mathbf{G}_r(\mathbf{q}) = w_1(\mathbf{q})\mathbf{G}_1(\mathbf{y}_1(\mathbf{q})) + w_2(\mathbf{q})\mathbf{G}_2(\mathbf{y}_2(\mathbf{q}))$$
$$\Phi_r(\mathbf{q}) = w_1(\mathbf{q})\Phi_1(\mathbf{y}_1(\mathbf{q})) + w_2(\mathbf{q})\Phi_2(\mathbf{y}_2(\mathbf{q}))$$

Because w_i is a function of $\mathbf{q}$ not $\mathbf{y}_i$ and the energy function of RMPflow only allows summing child-node functions, this example root node policy does not admit a tree structure decomposition in the original RMP-tree and can only be implemented as a single large node. Conversely, because of the weight function on the edges, RMP-tree* can further exploits potential sparsity inside the policy representation so that building complicated global polices with only basic elementary policies becomes possible. We note that the example above does not imply that RMPfusion can generate more expressive policies than RMPflow. More precisely, RMP-tree* allows representing the same global policy using more basic leaf-node policies. This property has two implications: it suggests (i) RMPfusion can be more efficient to compute and (ii) RMPfusion can offload the difficulties of designing leaf-nodes policies into the weight functions, which are learnable.

6.3 End-to-end learning of weight functions

We presented a new computational graph, RMPfusion, which supplements RMPflow with a set of multiplicative weight functions to achieve extra flexibility in policy fusion. Here we show these weight functions can be learned from data, and therefore RMPfusion can be treated as a parameterized policy class in policy optimization.

Suppose we have an RMP-tree* in which some weight functions are parameterized as function approximators. For example, we can consider the neural network presented in Figure 6.2 as the weight function. To show the weights are learnable, it is sufficient to check if we can differentiate through the output of the final policy $\pi = \mathbf{a}_r$ with respect to the parameters that specify the weight functions. As the computation of $\mathbf{a}_r$ is accomplished



(a) (b) (c)

Figure 6.2: (a) Shows the network used for learning with RMPfusion, specifically for any node i on the RMP-tree*, with children $c_0, \ldots, c_j$. If i is a leaf node, then it is evaluated from the designed RMP policy. The global policy is obtained by applying `resolve` on the root node RMP. RMP-tree* used in experiments for (a) `2d1level` and (b) `2d2level`.

recursively in the backward pass using `pullback`*, we will only illustrate that `pullback`* is differentiable. This can be seen by treating `pullback`* as a computation graph, as illustrated in Figure 6.2. Take the nodes in Eq. (6.1) as an example. `pullback`* receives $\mathbf{f}_i$, $\mathbf{M}_i$, $\mathbf{G}_i$, $\mathbf{B}_i$, $\mathbf{J}_i$, $\dot{\mathbf{J}}_i$, L_i from the edges to the child nodes, the current state $(\mathbf{x}, \dot{\mathbf{x}})$ and the auxiliary state to define the weight function w_i and the correction term $\mathbf{h}_i$. As these inputs values do not depend on the weight functions $\{w_i\}$ at the current node (i.e. they do not form a loop), the derivative of $\mathbf{a}_r$ with respect to the weight functions in the RMP-tree* can be computed recursively by back-propagating the derivatives through each `pullback`* operator. This, for example, can be implemented easily through computational graph libraries like tensorflow [160] or pytorch [122].

It is important to note that we do not have to learn all the weight functions in an RMP-tree*. If we know that certain leaf-node RMPs have to be turned on, we can adopt a semi-parametric scheme of weight functions. For example, we can design parameterization of the weight functions such that only collision avoidance RMPs are turned on, when the robot is extremely close to an obstacle. This property is due to the structure of RMP-tree*, which is interpretable, unlike policies purely based on general function approximators. Interpretability allows for prior knowledge (like constraints and preferences) to be easily incorporated into the policy structure. This feature is particularly valuable for policy

learning with safety constraints [161].

6.4 Evaluation

We validate our approach with experiments of imitation learning. The goal is to show that RMPfusion with an RMP-tree* that is parametrized by randomly initialized neural networks (as in Figure 6.2) is able to mimic the expert policy's behavior through observing expert demonstrations. This setup simulates the situation where the user of RMPfusion only provides imperfect subtask policies. We also use these experiments to validate the stability properties of RMPfusion by studying if the energy function of the policies generated by RMPfusion (even the premature ones obtained before learning converges) decay monotonically over time. We perform these experiments with a 2D particle robot and with a Franka Panda 7-DOF robot. The supplementary video shows example executions with the Franka robot in simulation and on the real world platform (Figure 6.1).

As our aim it not invent a new imitation learning algorithm, we adopt the most basic approach, behavior cloning [162], in which the demonstrations are purely generated by running the expert policy alone without any active intervention from the learner. The objective of these experiments is to study how well RMPfusion can recover the behaviors of an expert that is within its effective policy class, and therefore we use a known RMP-tree* with fixed weights as the expert policy. We choose this setting to rule out bias due to mismatches between policy classes, because properly handling policy class biases in imitation learning is a non-trivial research question on its own right [163, 164, 165].

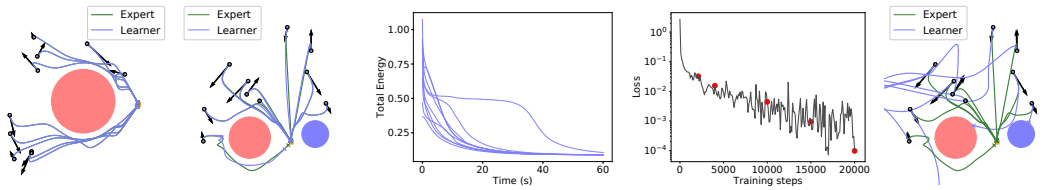
6.4.1 2D robot

We first validate our approach on two problems where a 2D robot is tasked with reaching a goal while avoiding one obstacle (`2d1level`) or two obstacles (`2d2level`). The RMP-tree* for these problems are shown in Figure 6.2b-6.2c. `2d1level` consists of a 2D particle that aims to reach a goal while avoiding an obstacle. The RMP-tree* for

`2d1level` is of depth one (see Figure 6.2b), where the root node q (configuration space of the robot) has one child obstacle RMP node (o_{rmp}) and one child attractor RMP node (a_{rmp}). `2d2level` consists of a 2D particle that aims to reach a goal while avoiding two obstacles. The RMP-tree* for `2d2level` is of depth two (see Fig 6.2c), where the root node (q) has one child attractor RMP node (a_{rmp}) and one all-obstacle RMP (o) that is meant to combine two child obstacle RMPs (o_{rmp}, one for each obstacle). The respective weight functions are shown on the edges of both these trees. In the `2d1level` problem, the aim is to show near-perfect recovery of the weights given that the problem is convex in the weight functions. The `2d2level` problem adds extra complexity to the learning process. It introduces multiplication between weights so the weights cannot be uniquely identified. The aim here is to show that close-to-expert behavior can still be achieved.

Data

For each problem, the expert policy is generated by the respective RMP-tree* with some fixed assigned weights, which are unknown to the learner. The training data consist of 20 *randomly selected environments* with varying placements and sizes of obstacles. In each environment, the expert is run to generate 50 trajectories from unique initial states, and 60 temporally equidistant data points on each trajectory are recorded. Each data point is a pair of input and output: the input consists of the state (position and velocity) of the 2D particle and the auxiliary state (obstacle location and dimension, goal location) i.e. the meta information about the environment; the output consists of the action (acceleration) as specified by the expert given the input state visited by running the expert policy. Test data are collected by repeating this process with 5 new environments with 10 trajectories in each environment.



(a) (b) (c) (d) (e)

Figure 6.3: Trajectories generated in (a) `2d1level` and in `2d2level` by (b) `learner-rmp` and (e) `learner-un`, compared to the expert are shown. Initial state is a black circle for position and black arrow for velocity. The environment has obstacles (red and blue) and goal (orange square). (c) shows the corresponding energy function for `learner-rmp` trajectories in (b). (d) shows the learning curve for `learner-rmp`.

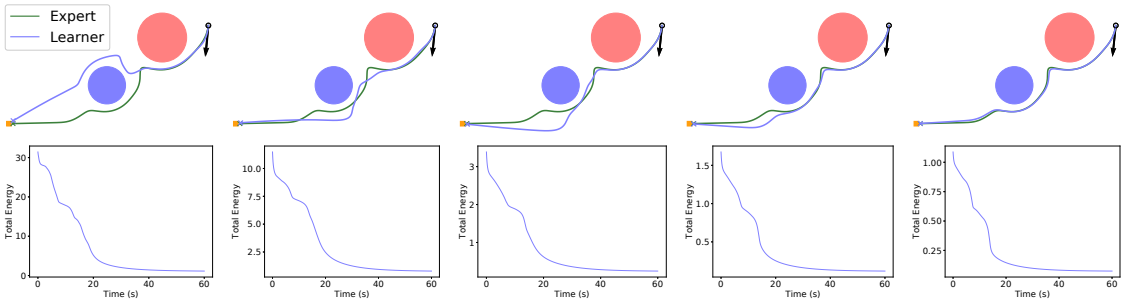


Figure 6.4: Improvement of the behavior produced by `learner-rmp` at various stages during training for `2d2level`. The top row shows the trajectories and the bottom row shows the corresponding energy function. From left to right these plots correspond to the red dots from left to right on the training curve in Figure 6.3d.

Unstructured network

For `2d2level` we also compare our RMPfusion `learner-rmp` with an unstructured neural network `learner-un`. This is a fully connected feed forward network with similar number of learnable parameters compared to `learner-rmp`. This network takes robot state and auxiliary state as the inputs, and outputs the acceleration. Our aim with this comparison is to show that an unstructured approach cannot offer any stability or safety guarantees, and with the same amount of data and training underperforms compared to the structured approach.

Training

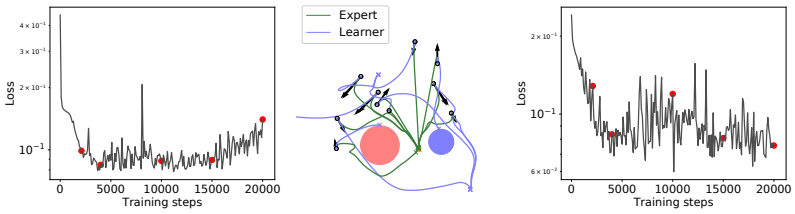
We use the mean squared error between the action generated by any learner and the action specified by the expert as the loss function for imitation learning. All learners are trained using RMSprop [166] with a minibatch size of 200 for 20K iterations.

Results

We report two types of test loss: the batch-loss is the average loss on the entire test dataset generated by the expert policy, and the online-loss is the average loss at every time step (1 second interval) on the trajectories generated by the learner's policy starting from the initial states in the test dataset. In `2d1level`, the batch-loss is 5.42e-5 and the online-loss is 5.82e-5. In `2d1level`, for `learner-rmp` the batch-loss is 2.45e-4 and the online-loss is 2.78e-4, while for `learner-un` the batch-loss is 0.111 and the online-loss is 12.203. The higher batch-loss for `learner-un` indicates that with the same amount of data and training the network is unable to learn the policy from the expert, while the much worse online-loss indicates that it cannot generalize well and succumbs to covariate shift problems.

Figures 6.3a, 6.3b and 6.3e show the evaluation of the trained networks on an example test environment. These results show that RMPfusion can perfectly match the behavior of the expert in the convex case (`2d1level`), while achieving near-expert performance in the non-identifiable case (`2d12evel`). From the overall results we also observe that `learner-un` is never able to reach the goal and also has a collision rate of 28% (e.g. Figure 6.3e), whereas `learner-rmp` successfully finishes the task 100% of the time.

Figure 6.4 shows the improving progression of `learner-rmp` during training, in which each snapshot corresponds to an associated point on the training curve in Figure 6.3d. This verifies that with training we can progressively improve the behavior of the learner. In addition, we verify that the stability properties of RMPfusion in the associated energy functions in Figure 6.3c and Figure 6.4. We see that, regardless of the setting, the energy



(a) (b) (c)

Figure 6.5: (b) Trajectories generated in `2d2level` by `learner-rmp-large` compared to the expert is shown. Initial state is a black circle for position and black arrow for velocity. The environment has obstacles (red and blue) and goal (orange square). Learning curves for (a) `learner-rmp` and (c) `learner-rmp-large` on `2d2level` is also shown.

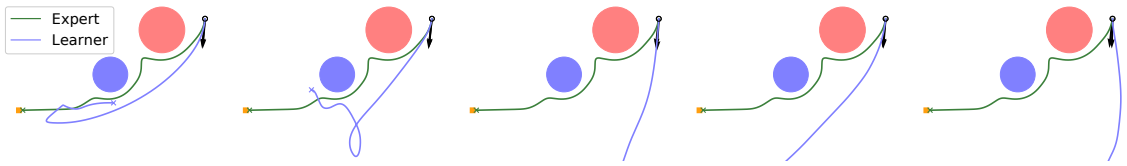


Figure 6.6: Trajectories produced by `learner-un` at various stages during training for `2d2level`. From left to right these plots correspond to the red dots from left to right on the training curve in Figure 6.5a.

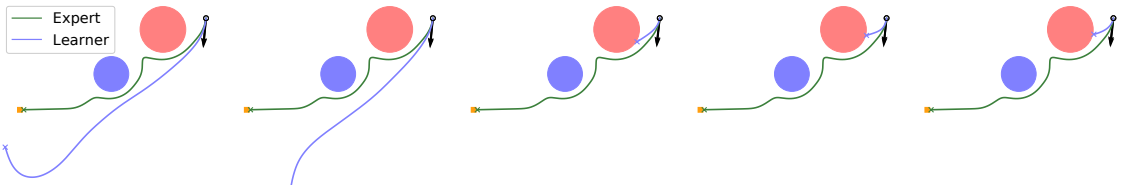


Figure 6.7: Trajectories produced by `learner-un-large` at various stages during training for `2d2level`. From left to right these plots correspond to the red dots from left to right on the training curve in Figure 6.5c.

functions always decays monotonically as indicated by Theorem 3. This suggests RMPfusion produces a stable policy even when the learned weight functions are premature before the learning has converged (Figure 6.4). On the other hand, `learner-un` does not always avoid collision or provide any stability during or after training (see Figure 6.6).

We also compared with a unstructured network, `learner-un-large`, that has 5.8 times more learnable parameters compared to `learner-un`. We see improvement over loss values where the batch-loss is 0.065 and the online-loss is 0.393, and the collision rate decreases to 16%. However, it is still never able to complete the task (e.g. see Figure 6.5b).

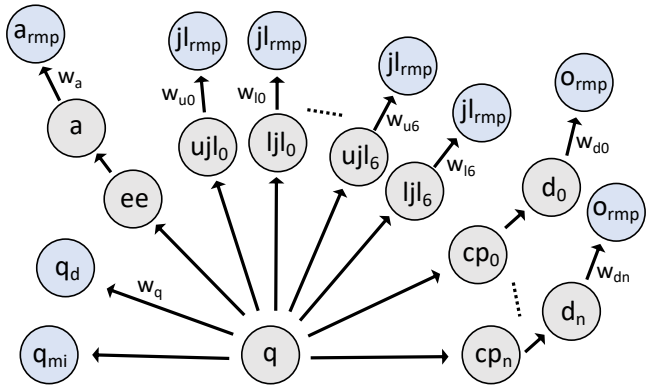


Figure 6.8: RMP-tree* used for the Franka robot. Gray nodes show task spaces, blue nodes show subtask RMPs, and weight functions are shown on the respective edges where they are defined. See text for more details.

Figure 6.7 shows the progression of `learner-un-large` during training, in which each snapshot corresponds to an associated point on the training curve in Figure 6.5c.

6.4.2 Franka robot

We also validate our approach on a more realistic setup with a Franka Panda 7-DOF robot arm. In these experiments, the tasks is to reach a goal while navigating around an obstacle. The RMP-tree* used is shown in Figure 6.8, where the configuration space of the robot is the root node (q) and weights functions are shown on the edges where they are defined. From the root node we have various task spaces, like the end-effector position (ee) on which the attractor space (a) is defined by a change of coordinates such that the goal position is at the origin. The attractor RMP (a_{rmp}) is then defined on the attractor space for a goal reaching subtask. Each joint of the robot is mapped to a one dimensional upper (ujl_i) and lower (ljl_i) joint limit space where a joint limit RMP (jl_{rmp}) is defined for joint limit avoidance subtasks. The root node is also mapped to a pre-specified number of control points on the robot (cp_i) such that they collectively approximate the robot's body and can be used for collision avoidance. On any control point space we add a distance space to

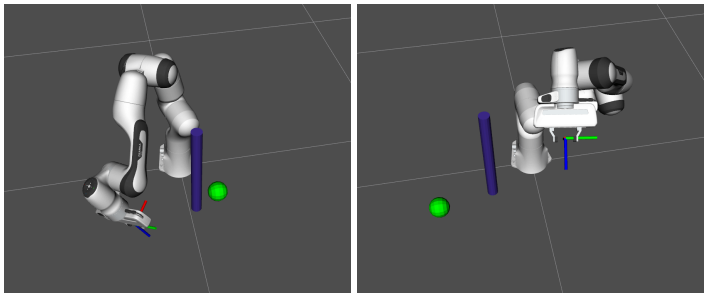


Figure 6.9: An example from the test and training dataset (left) and the validation dataset (right). The robot is shown in its start configuration with an obstacle (cylinder) and a goal (sphere).

the obstacle (d_i) where the obstacle RMP (o_{rmp}) is defined.[1] Finally, there are also native RMPs defined on the root node like a constant damper RMP (q_d) and a RMP which is just an identity metric (q_{mi}) with no learnable weight function to ensure the `resolve` operator is numerically stable.

Data and training

The expert policy is given by the RMP-tree* with some fixed known weights, while the learner's policy is defined by the RMP-tree* with neural network weight functions that will be learned through behavior cloning.

For training and test data we place an obstacle in a fixed location near the robot and sample different start configurations and goal locations that are in a region in front of the obstacle from the robot's perspective such that it forces the robot to interact with the obstacle while trying to reach the goal. We run the expert to generate 110 and 30 unique trajectories respectively for training and test data. The trajectories are 5-10 seconds long and data is collected every 0.1 seconds. Any data point consists of the state (configuration position and velocity of the robot), the auxiliary state (distances to goal and obstacle), and the expert action (acceleration). In a new environment with a different placement of the

[1]Note that when multiple obstacles are present we can add distance spaces and the obstacle RMPs for each obstacle on every control point. Now, since the tree structure can change with the number of obstacles, in practice, shared weights can be specified across all obstacles on a given control point, such that training can be performed with only one obstacle to learn the weight function and then can be applied to arbitrary number of obstacles during execution.

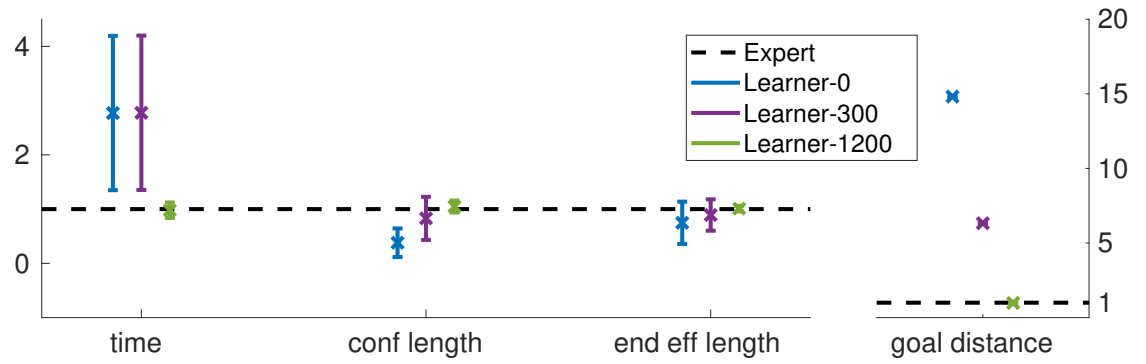


Figure 6.10: Learner's performance with respect to the expert on the validation dataset for the experiments with the Franka robot.

obstacle this process is repeated to gather the validation dataset where the expert is used to generate 20 unique trajectories. An example from the training and validation dataset is shown in Figure 6.9.

The loss function is the same as in the experiments with the 2D robot and we train the policy using ADAM [167] with a minibatch size of 200 for 1500 iterations.

Results

We compare the performance of the learner, against the expert, at various stages of training, `learner-0` at no training (the neural network is initialized with random weights), `learner-300` at 300 iterations, and `learner-1200` at 1200 iterations when the learning converges. We record the following metrics on the validation dataset for the expert and all the learners: (i) time: the time to reach within a precision of $0.05m$ of the goal; we time-out the execution at 10 seconds, (ii) conf length: the distance traveled in configuration space, (iii) end eff length: the distance traveled by the end effector in workspace, and (iv) goal distance: the distance to the goal from the end effector at the end of an execution.

Figure 6.10 shows the performance of the learners relative to the expert on the validation dataset (it plots the mean and the standard deviation of the ratios of the learner's metric and the expert's metric across trajectories; the expert is shown as the dotted horizontal baseline). From these results we see that when the learner is not trained the robot does not move much

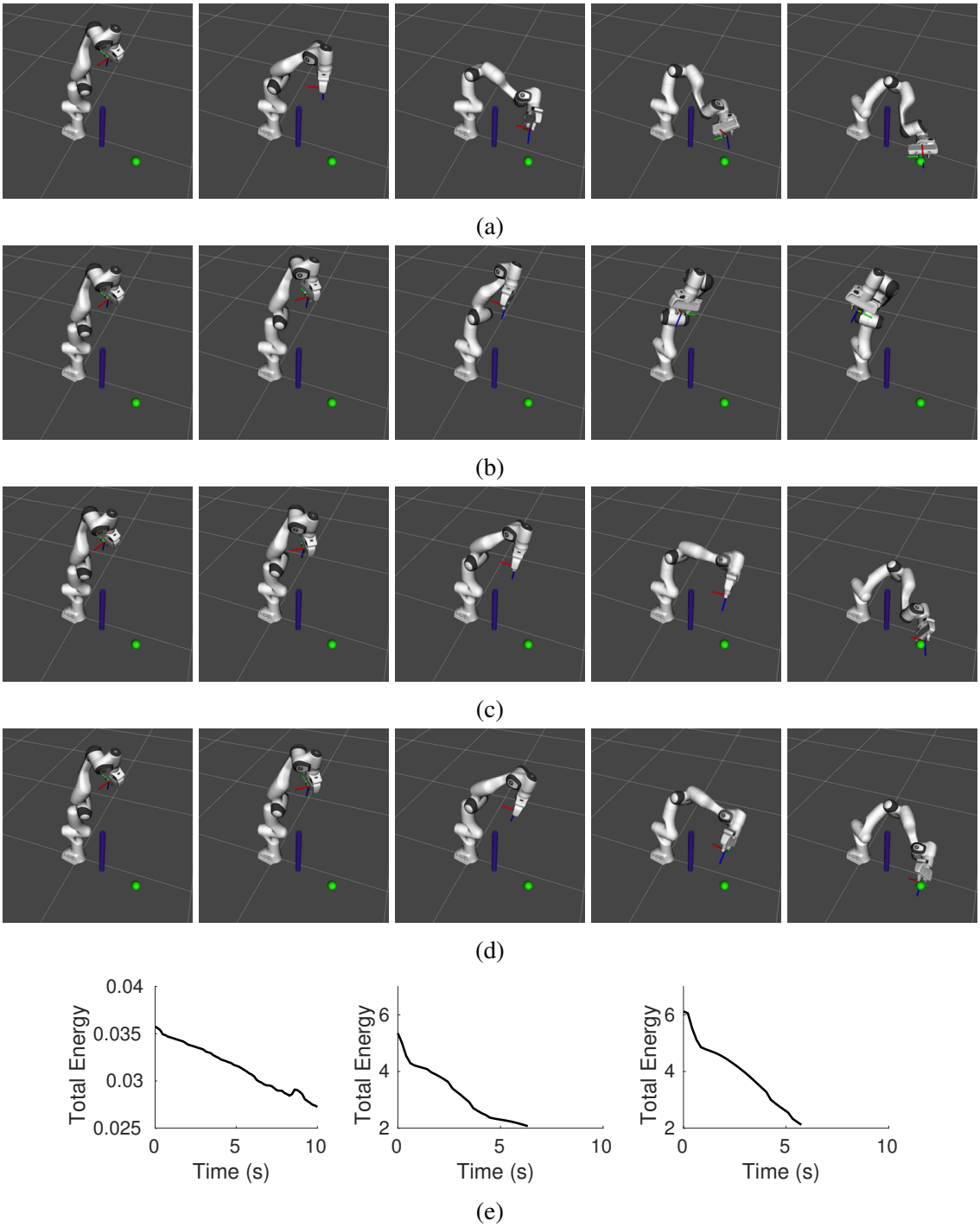


Figure 6.11: (a)-(d) An example execution (left to right) from the validation dataset, comparing (a) the expert with (b) `learner-0`, (c) `learner-300`, and (d) `learner-1200`. (e) The respective total energy profiles of the learners' trajectories (`learner-0` (left), `learner-300` (middle), `learner-1200` (right)).

and incurs a high goal distance before timing out. With more training the goal error reduces as the robot start traveling towards the goal but it still often times out. As the learning converges so does the performance of the learner towards the expert's performance. In all the trajectories across all the learners there are no collision, which verifies that constraints like safety can be incorporated through the structured learning approach that RMPfusion allows. We also show a qualitative comparison on an example execution with the expert and the learners in Figure 6.11 and also verify the stability properties of RMPfusion (even during learning) with the monotonically decreasing energy plots on these executions. Note that the scale on the energy plot for `learner-0` is very small and the tiny kink on the plot is due to numerical issues with Euler integration.

6.5 Discussion

We introduce extra parametrization flexibility into RMPflow and propose a new algorithm called RMPfusion. RMPfusion features a set of learnable weight functions that specifies the importance of subtask policies based on the robot's configuration and the environment. Consequently, RMPfusion can combine imperfect subtask policies into a global policy with good performance, where the original RMPflow fails. We demonstrate the ability of RMPfusion to learn weight functions for policy fusion in simulation, and further theoretically prove that RMPfusion inherits the Lyapunov-type stability from RMPflow with only mild conditions on the weight functions. These structural properties and encouraging experimental results of RMPfusion suggest that RMPfusion can be treated as a class of structural policies suitable for policy learning with safety and interpretability requirements.

CHAPTER 7
LEARNING REACTIVE MOTION POLICIES

7.1 Introduction

Performing complex manipulation tasks in unstructured environments involve *specific*, *coordinated*, and *adaptive* movements of different parts of the robot, and thus can be viewed as being composed of several inter-related subtasks, each associated with a desired behavior. For example, consider a wiping task. In addition to requiring the end-effector to maintain contact, it is desirable to maintain a specific elbow orientation in order to effectively wipe the surface. Furthermore, the movements of the elbow and the end-effector are inter-dependent, requiring coordination and adaptation based on the task and environmental changes such as the presence of a new obstacle.

In this chapter, we introduce a framework to learn time-invariant *reactive* policies, each associated with a certain subtask, building on the structured framework introduced in Chapter 5. We characterize the desired behavior in each task space as an RMP. As we have seen in Chapter 5, specifying RMPs for complex behavior is not straightforward. To handle complex tasks, we propose to learn RMPs directly from a demonstration provided by the human. This framework involves learning a subset of RMPs from demonstration, while manually specifying the rest. For example, RMPs for obstacle avoidance and joint limits can be specified with ease, while RMPs associated with desired behaviors at different robot links can be learned from demonstration. Specifically, we learn a potential function associated with the desired behavior. The learned potential has a guaranteed unique global minimum and thus ensures convergence. We utilize the learned potential to define a Lyapunov-stable (GDS). Furthermore, based on the learned potential we identify a Riemannian metric that is consistent with the underlying geometry and defines a notion of

Figure 7.1: Sawyer robot performing a complex manipulation task in a constrained environment, by coordinating the movements of different parts of its arm.

distance in the corresponding subtask space.

Indeed, the subtask spaces of the robot are not independent. They must satisfy certain constraints, such as those required by robot kinematics, while simultaneously accomplishing task goals. Our framework explicitly considers the kinematics of the robot and the dependencies between the policies during motion generation. Toward this end, we utilize RMPflow to combine the policies associated with different subtask spaces of the robot. When combining multiple such policies, the identified Riemannian metric associated with a given policy provides a notion of directional importance and hence enables policy resolution. As known from Section 5.7, if each individual policy is given by a Lyapunov-stable GDS, then the combined policy is also Lyapunov-stable. We demonstrate the effectiveness of this approach on a complex robot manipulation task and illustrate the necessity of learning policies in multiple subtasks spaces and combining them in a geometrically consistent manner.

7.2 Related work

Motion generation techniques for articulated robots can be broadly grouped into motion planning [6, 9], reactive policy synthesis [133, 138], and learning from demonstration

(LfD) [96]. Trajectory optimization-based planning methods minimize a cost function to yield smooth collision free solutions that consider the robot kinematics[9, 45, 41]. In practice, their performance is limited by the parameters in the cost function and their slower evaluation speeds compared to reactive policy methods.

Reactive methods are very more well suited for dynamic environments. Recent work in this area has explored the explicit use of non-Euclidean geometry [139, 138] and addresses the problem of systematically combing multiple policies, each defined in a different task space. Unlike existing methods that utilize user-designed behaviors in each task space, our approach enables learning the desired behaviors directly from an expert demonstration.

In contrast to trajectory-based planning, trajectory-based LfD methods, do not require user-specified cost functions and can exploit human demonstrations to learn how to perform complex tasks. Such demonstrations have been used to learn either an underlying cost function [107, 110, 168] or the state-to-action policy [150, 101, 169, 170]. Our work belongs to the category of methods that learn policies directly from the demonstration. These methods have appealing properties such as convergence guarantees, time-invariance, and instantaneous adaptation. However, existing methods learn the end-effector policies without accounting for the behaviors of other parts of the robot, and they do not explicitly consider the robot kinematics and joint limits while generating motions.

A potential approach to simultaneously encoding the role of different parts of the robot is to consider the movement in the robot's configuration space. Prior work has explored learning policies that simultaneously consider both configuration space and task space constraints [171, 172, 173] (also see Chapter 3. These methods however provide time-dependent policies, and are thus prone to perturbations. An approach to combine learned reactive configuration space policies is proposed in [174], but this formulation is coordinate-specific and does not generalize to robots across different embodiments. Our framework, on the other hand, can learn coordinate-free policies in an arbitrary number of subtask spaces as necessary given the the task.

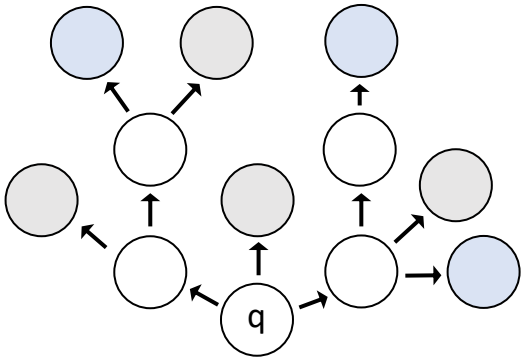


Figure 7.2: Example RMP-tree with root node as the configuration space q, white nodes are other task spaces on the kinematic tree. There two types of leaf nodes with learned (blue) and hand designed (grey) RMPs.

To the best of our knowledge, no existing work combines multiple learned stable time-invariant policies defined on different task spaces of the robot in a geometrically consistent manner.

7.3 Skill reproduction via RMPflow

To accomplish a desired skill, we aim to generate a configuration space policy $\mathbf{a}_r = \pi(\mathbf{q}, \dot{\mathbf{q}})$ via RMPflow such that the all the constraints in the relevant subtask spaces are simultaneously satisfied. The constraints can be either imposed by the requirements of the skill itself, or the environment where the skill has to be accomplished. With the root node of the RMP-tree associated with the configuration space $\mathcal{C}$, all these constraints can be encoded as individual leaf RMPs collected in a set $\mathcal{V} = \{v_{l_i}\}$ with task map $\mathcal{T} = \{\mathcal{T}_{l_i}\}$. A subset of these leaf RMPs can be chosen to be learned from human demonstrations in order to encode the skill constraints. While the remaining leaf RMPs can be hand-specified. A noteworthy class of hand-specified leaf RMPs here are the obstacle avoidance RMPs. Adding obstacle avoidance leaf RMPs to the RMP-tree guarantees collision-free behaviors.

For a given skill, we argue that an RMP learned in the end-effector subtask space alone is not sufficient to encode all the relevant skill constraints. This is specially true for highly redundant manipulators where a given end-effector trajectory can be achieved with many

different configuration space trajectories. Hence, we require definition of additional subtask spaces along the robot's body to fully encode a demonstrated skill. A human can choose to provide demonstrations for each subtask space either independently or simultaneously. While the choice of subtask spaces is user and task dependent, a trivial choice of subtask spaces are the locations of the reference frames located at the end of all the robot links. Hence, for a N-link robot manipulator, we choose to define N human-guided leaf RMPs $\mathcal{V}_{1:N} \subseteq \mathcal{V}$ with associated subtask space $\mathcal{T}_{1:N} \subseteq \mathcal{T}$ governed by the kinematics of the robot. Here the N-th human-guided leaf RMP refers to the desired end-effector behavior.

Before we elaborate our learning approach, a few assumptions are presented in order. For the skills we consider in this paper, a single human demonstration per subtask space is assumed to be sufficient. Furthermore, successful execution of a skill is assumed to require satisfaction of positional and curvature constraints, as opposed to velocity profile.

7.3.1 Human-guided Riemannian motion policies

Given a single subtask space human demonstration, our aim is to learn an RMP which encodes the desired behavior as a motion policy, alongside a metric which gives directional importance to this policy when combined with other policies using RMPflow. We desire that all the trajectories on the subtask manifold smoothly converge to the demonstration and are stabilized at the demonstrated goal position. Let us define this desired behavior as an acceleration policy as $\ddot{\mathbf{x}}^d = \mathbf{f}(\mathbf{x}, \dot{\mathbf{x}})$.

We choose to imitate this desired acceleration policy using a geometric dynamical system (GDS). A GDS dictates the motion on a non-Euclidean manifold, which is also inherently Lyapunov stable. The task of learning an RMP is then equivalent to learning the parameters of a GDS from human demonstration. To achieve this, we employ a simpler form of GDS whereby the manifold defining Riemannian metric is solely dependent on

position,

$$\mathbf{M}(\mathbf{x})\ddot{\mathbf{x}} = -\nabla_{\mathbf{x}}\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} - \boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}}). \tag{7.1}$$

For this special case, the inertia matrix $\mathbf{M}(\mathbf{x})$ is equivalent to the Riemannian metric itself. To make the connection between a desired a acceleration policy and a GDS, let us rewrite Eq. (7.1) as,

$$\ddot{\mathbf{x}}^d = \mathbf{M}(\mathbf{x})^{-1}\big(- \nabla_{\mathbf{x}}\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} - \boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}})\big) \tag{7.2}$$

In essence, we require the accelerations generated by the GDS to match with those desired. Let us decompose the desired acceleration from GDS into three components: (i) desired *potential-based acceleration*: $-\mathbf{M}(\mathbf{x})^{-1}\nabla_{\mathbf{x}}\Phi(\mathbf{x})$; (ii) *damping acceleration* for stability: $-\mathbf{M}(\mathbf{x})^{-1}\mathbf{B}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}}$; and (iii) *curvature acceleration* for geometrically consistent behavior: $-\mathbf{M}(\mathbf{x})^{-1}\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}})$. Hence Eq. (7.2) can be concisely written as,

$$\ddot{\mathbf{x}}^d = -\nabla_{\mathbf{x}}\tilde{\Phi}(\mathbf{x}) - \tilde{\mathbf{B}}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} - \mathbf{M}(\mathbf{x})^{-1}\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}}) \tag{7.3}$$

where $\tilde{\Phi}$ and $\tilde{\mathbf{B}}$ refer to some other potential and damping matrix, which we call *warped potential* and *warped damping matrix* respectively. We can directly specify a warped potential and warped damping combination to achieve our desired subtask space behavior. Thereon, we can choose an admissible metric Riemannian metric $\mathbf{M}(\mathbf{x})$ which obeys the following relationship,

$$\nabla_{\mathbf{x}}\Phi(\mathbf{x}) = \mathbf{M}(\mathbf{x})\nabla_{\mathbf{x}}\tilde{\Phi}(\mathbf{x}); \quad \mathbf{B}(\mathbf{x}) = \mathbf{M}(\mathbf{x})\tilde{\mathbf{B}}(\mathbf{x}). \tag{7.4}$$

We observe here that the total desired acceleration generated by the GDS is mainly governed by the potential-based acceleration, while the remaining terms simply add stability

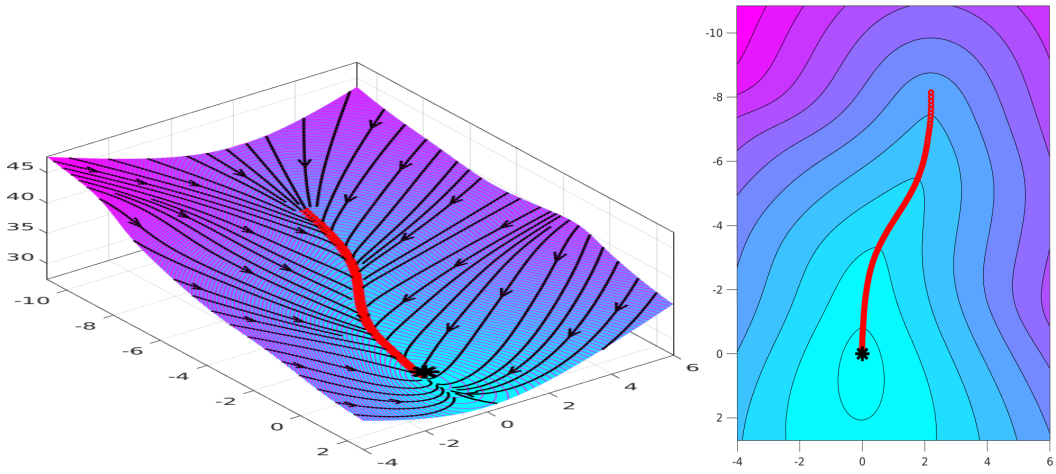


Figure 7.3: *Left*: Visualization of *warped potential* learned from a demonstration overlayed on top in red. The arrows show the negative gradient direction. *Right*: Isocontours representing points equidistant to the target on the underlying manifold defined by the metric that stretches space in the direction of the potential gradient.

and consistency. Hence, we choose to learn the warped potential from demonstration such that its negative gradient combined with a constant hand-specified warped damping matrix $\tilde{\mathbf{B}} = \gamma_d \mathbf{I}$, modulus the curvature acceleration, achieves the desired behavior.

The remainder of this section details our learning procedure followed by our methodology for generating an admissible Riemannian metric which enables combining the learned subtask policies via RMPflow.

7.3.2 Learning warped potentials from demonstration

We seek to learn a warped potential which generates acceleration in line with our desired accelerations. Let us assume the availability of a demonstration $\boldsymbol{\zeta} = \{\boldsymbol{\zeta}_i\}_{i=1:T}$ in a given subtask space composed of T data points. To learn a valid potential function, we restrict our search space to the class function which are positive. Furthermore, we also desire the potential function to be convex with a single unique global minima located at the final location $\mathbf{x}_* = \boldsymbol{\zeta}_T$. The latter requirement prevents the introduction of spurious attractors in the state-space. To achieve this, we use an approach similar to that used by [175]. Specifically, we carry out kernelized regression whereby the overall warped potential at

a query point is given by a convex combination of potential elements centered at each trajectory point $\boldsymbol{\zeta}_i$, i.e.,

$$\tilde{\Phi}(\mathbf{x}) = \sum_{i=1}^{i=T} w_i(\mathbf{x})\tilde{\phi}_i(\mathbf{x}), \quad \text{where,} \sum_{i=1}^{i=T} w_i(\mathbf{x}) = 1 \tag{7.5}$$

Each weight $w_i(\mathbf{x})$ is defined by a normalized radial basis function centered at $\boldsymbol{\zeta}_i$ with bandwidth σ,

$$w_i(\mathbf{x}) = \frac{k(\mathbf{x}, \boldsymbol{\zeta}_i)}{\sum_{j=1}^{j=T} k(\mathbf{x}, \boldsymbol{\zeta}_j)}, \tag{7.6}$$

$$\text{where,} k(\mathbf{x}, \boldsymbol{\zeta}_i) = e^{-\frac{1}{2\sigma^2}(\mathbf{x}-\boldsymbol{\zeta}_i)^T(\mathbf{x}-\boldsymbol{\zeta}_i)} \tag{7.7}$$

Furthermore, each contributing potential element here is a summation of two components,

$$\tilde{\phi}_i(\mathbf{x}) = \tilde{\phi}_i^{attract}(\mathbf{x}) + \tilde{\phi}_i^0 \tag{7.8}$$

where the component $\tilde{\phi}_i^{attract} : \mathbb{R}^m \mapsto \mathbb{R}_+$ is a strictly convex function with a global minima at ζ_i and $\phi_i^0 \in \mathbb{R}_+$ is a constant bias term. As a consequence of the aforementioned decomposition, the gradient of the overall warped potential in Eq. (7.5) can also be decomposed as,

$$\nabla_{\mathbf{x}}\tilde{\Phi}(\mathbf{x}) = \nabla_{\mathbf{x}}\tilde{\Phi}^{attract}(\mathbf{x}) + \nabla_{\mathbf{x}}\tilde{\Phi}^{nominal}(\mathbf{x}) \tag{7.9}$$

where the gradient component $\nabla_{\mathbf{x}}\tilde{\Phi}^{attract}(\mathbf{x})$ causes attractive pull towards the demonstration while the nominal component $\nabla_{\mathbf{x}}\tilde{\Phi}^{nominal}(\mathbf{x})$ produces accelerations along the direction of motion of the demonstration. The attractive gradient component can be further

written as a weighted summation of gradients of individual attractive potential elements,

$$\nabla_{\mathbf{x}}\tilde{\Phi}^{attract}(\mathbf{x}) = \sum_{i=1}^{i=T} w_i(\mathbf{x})\nabla_{\mathbf{x}}\tilde{\phi}_i^{attract}(\mathbf{x}) \tag{7.10}$$

Fortunately due to this decomposition, we can carry out potential function learning as two-step process. First, we independently design the function $\tilde{\phi}_i^{attract}(\mathbf{x})$ as per our desired attractive accelerations towards the demonstration. Second, we learn the bias terms $\tilde{\phi}_i^0$ such that the direction of motion governed by the potential at each datapoint ζ_i matches with the one demonstrated.

As a first step in this procedure, we choose to go with the following attractive potential element,

$$\tilde{\phi}_i^{attract}(\mathbf{x}) = \frac{1}{\eta}\log\left(e^{\eta\|\mathbf{x}-\boldsymbol{\zeta}_i\|} + e^{-\eta\|\mathbf{x}-\boldsymbol{\zeta}_i\|}\right) \tag{7.11}$$

which is a η-scaled softmax function where $\eta > 0$ defines the effective smoothing radius of the function at the origin. The corresponding gradient is

$$\nabla_{\mathbf{x}}\tilde{\phi}_i^{attract}(\mathbf{x}) = \left(\frac{1 - e^{-2\eta\|\mathbf{x}-\boldsymbol{\zeta}_i\|}}{1 + e^{-2\eta\|\mathbf{x}-\boldsymbol{\zeta}_i\|}}\right)\frac{\mathbf{x}-\boldsymbol{\zeta}_i}{\|\mathbf{x}-\boldsymbol{\zeta}_i\|} \tag{7.12}$$

$$= s_\eta\big(\|\mathbf{x}-\boldsymbol{\zeta}_i\|\big)\frac{\mathbf{x}-\boldsymbol{\zeta}_i}{\|\mathbf{x}-\boldsymbol{\zeta}_i\|} \tag{7.13}$$

where $s_\eta(0) = 0$ and $s_\eta(r) \to 1$ as $r \to 0$. For a sufficiently large η, this choice of potential function ensures that the attractive acceleration always has a unit magnitude except in the neighborhood of the center ζ_i where it smoothly decreases to zero. A trivial alternative to this function is the quadratic function as used by Khansari-Zadeh et al. [175]. However, the gradient of a quadratic function increases linearly with distance which can cause undesirably large accelerations far away from the data points.

Towards the second step in the procedure, we learn the bias terms $\tilde{\boldsymbol{\phi}}^0 = \{\tilde{\phi}_i^0\}$ such that at the data points $\boldsymbol{\zeta}_i$, the negative potential gradient aligns with the direction of the

demonstrated motion at these points with unit magnitude, i.e. $\hat{\dot{\boldsymbol{\zeta}}}_i = \frac{\dot{\boldsymbol{\zeta}}_i}{\|\dot{\boldsymbol{\zeta}}_i\|}$. The corresponding optimization problem is

$$\begin{aligned} \min_{\boldsymbol{\phi}^0} \ & \frac{1}{T}\sum_{i=1}^{i=T} \|\hat{\dot{\boldsymbol{\zeta}}}_i + \nabla_{\mathbf{x}}\tilde{\Phi}(\boldsymbol{\zeta}_i;\boldsymbol{\phi}^0)\|^2 + \lambda\|\phi_i^0\|^2 \\ \text{s.t.} \quad & \tilde{\phi}^0_{i+1} \leq \tilde{\phi}^0_i \quad \forall i = 1,\ldots,(T-1) \\ & 0 \leq \tilde{\phi}^0_T \\ & \nabla_{\mathbf{x}}\tilde{\Phi}(\boldsymbol{\zeta}_T) = 0 \end{aligned} \tag{7.14}$$

where the optimization constraints enforce the potential to decrease monotonically along the demonstration with an unique global minima at the goal location $\boldsymbol{\zeta}_T$. The aforementioned optimization problem is convex, and hence can be solved efficiently with off-the-shelf solvers, e.g. CVX [176]. Slack variables can be added to ensure feasibility of the optimization problem.

7.3.3 Admissible metric for policy resolution

While there can be multiple such metrics $\mathbf{M}(\mathbf{x})$, we base our selection on the following key insight:

any arbitrarily curved trajectory on a Euclidean manifold can be viewed as a straight-line trajectory on a curved/non-Euclidean manifold.

In view of this, once a warped potential is learned, we seek to extract a Riemannian metric which defines the underlying subtask manifold. Concretely, this metric warps a simple potential which generates straight-line trajectories on a non-Euclidean manifold, into a more convoluted potential defined on a Euclidean manifold. Using this metric to combine multiple such learned policies via RMPflow allows giving each policy the appropriate directional importance as dictated by the underlying subtask geometry. A natural

choice for a metric in this regard is,

$$\mathbf{M}_{\text{stretch}}(\mathbf{x}) = \big(1 - \alpha(\mathbf{x})\big)\nabla_{\mathbf{x}}\tilde{\Phi}(\mathbf{x})\tilde{\Phi}(\mathbf{x})^T + (\alpha(\mathbf{x}) + \epsilon)\mathbf{I} \tag{7.15}$$

where $\alpha(\mathbf{x}) = \exp(-\frac{\|\mathbf{x}\|^2}{2\sigma_\alpha^2})$ is a decaying exponential and $\epsilon > 0$. This metric defines a manifold which is stretched in the direction of the warped potential when away from the goal $\mathbf{x}_*$, while becoming increasingly Euclidean when near the goal. In other words, warping a straight-line vector field generated by a simple potential with this metric, generates the desired behavior in the subtask space. For this choice of metric, we find the curvature term $\xi_{\mathbf{M}}$ by making use of numerical differentiation. It should be reiterated that the curvature terms keep the subtask trajectories consistent with the underlying geometry. Figure 7.3(right) shows a visualization of an example manifold found using the aforementioned method. The isocontours in the visualization represent points that are equidistant to the origin on this manifold. Indeed, the paths between the origin and every point on a given contour can be seen as geodesics of equal length.

7.4 Evaluation

We evaluated the proposed approach on a *constrained-placing* task executed on Rethink Sawyer robot. The task involved placing a green cube inside a narrow hole in a wall-like structure situated closely in front of the robot (see Figure 7.1). Accomplishing this task requires simultaneous satisfaction of constraints on multiple links. Specifically, the robot's end-effector must go around the wall, while the elbow is held high and away from the structure to avoid collision. Furthermore, the wrist and end-effector movements must enable a certain angle of approach in order to place the object at the goal position. Indeed, it is not straight-forward to manually specify RMPs that are appropriate for this task. Hence, we utilized our human-guided learning framework to acquire this skill.

To learn the desired skill, we used a single kinesthetic demonstration configuration

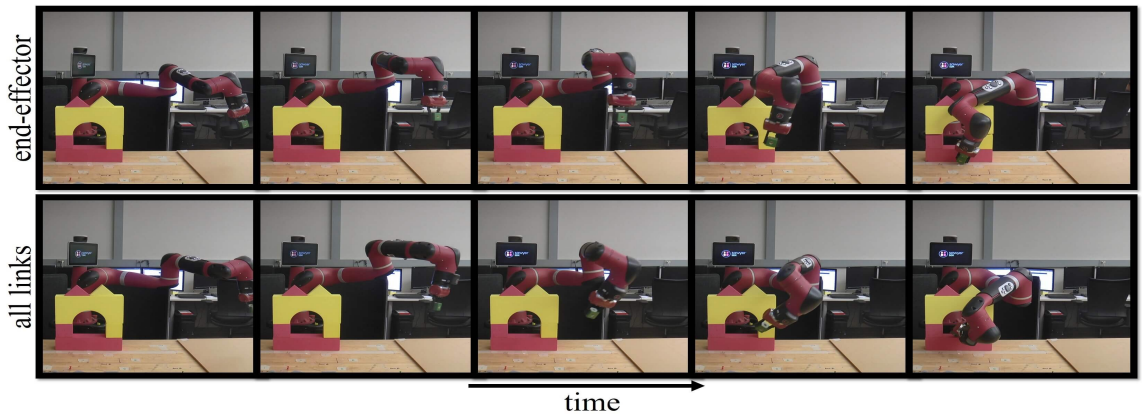


Figure 7.4: Sequence of images illustrating qualitative differences between learning leaf RMPs (a) only for the end-effector (*top row*), and (b) for all the robot link reference frames (*bottom row*) when starting from similar configurations.

space of the robot $\boldsymbol{\zeta}^{\mathcal{C}}$. To capture all the relevant constraints of the task, we setup an RMP-tree and learned four leaf RMPs on different subtask spaces, each associated with a link reference frame placed at the end of the corresponding link. We transformed the configuration space demonstration to all four subtask spaces based on the task maps given by the forward kinematics of the robot, yielding 7 transformed demonstrations $\{\boldsymbol{\zeta}^{\mathcal{T}_{l_i}}\}_{i=1}^{7}$, one for each subtask space. In each of these subtask spaces, we then learned a warped potential (Section 7.3.2). We identified an admissible metric for each subtask space to capture the underlying geometry (Section 7.3.3). We then derived the final GDS equation which defines the desired behavior in each subtask space. As in Eq. (7.3), we added damping and curvature terms to the GDS to ensure stability and geometric consistency. We empirically chose a constant damping matrix that minimizes the oscillations in the system. We found $\tilde{\mathbf{B}} = 10\mathbf{I}$ to be sufficient in all our experiments. After this procedure, we end up with $\{v_i\}_{i=1,\ldots,7}$ learned leaf RMPs defined by learned GDSs.

To execute the task, we combined these learned policies into a global configuration space policy as dictated by the RMP-algebra detailed in Section 5. It should be noted that the RMP-tree also contained other hand-specified leaf nodes which incorporate joint limits and joint damping factors. For further details on hand-specified RMPs, we refer the reader to [138].

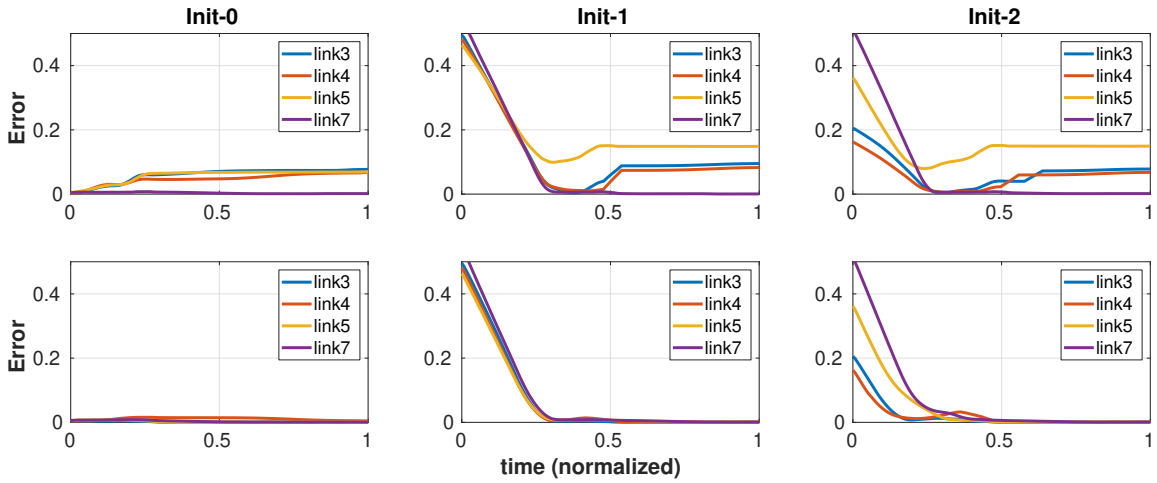


Figure 7.5: Time evolution of mean squared errors between reproduced link frame trajectories and the corresponding demonstrations over time for three different robot initial configurations (init-0 through init-1). The *top-row* visualizes errors for an RMP-tree with a human-guided leaf RMP defined solely on the end-effector, while *bottom-row* corresponds to an RMP-tree with human-guided leaf RMPs defined on all the robot link reference frames.

We evaluated the effectiveness of proposed approach and compared it to that of a *baseline*, which involves learning a single leaf RMP located at the end-effector. We evaluated the performances of both the proposed algorithm and the baseline from multiple initial configurations, i.e. *init-0*, *init-1* and *init-2*. Here, *init-0* refers to an initial configuration that is identical to the demonstration's initial configuration, while *init-1* and *init-2* are configurations obtained by perturbing the joint angles in two random directions.

Sequences of images illustrating qualitative differences between the proposed and baseline approaches when starting from *init-2* are are shown in Figure 7.4. The baseline method results in the robot's elbow colliding with the wall and fails to place the object in the hole. In contrast, the proposed approach coordinates the movement of each of its links and successfully accomplishes the task.

To quantify performance, we compute errors between the demonstrated and reproduced tasks-space trajectories for each of the four link frames. The error associated with a particular link is measured by the norm of the difference between demonstrated and reproduced task space trajectories associated with that link. We aligned the demonstrated and repro-

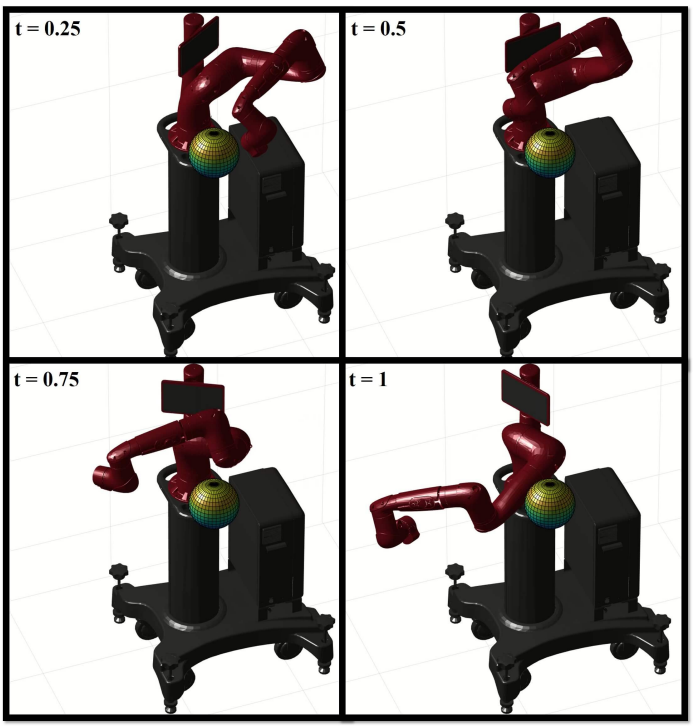


Figure 7.6: The combination of obstacle avoidance RMP and human-guided RMPs results in motions that are guaranteed to avoid collisions while simultaneously attempting to satisfy the task constraints.

duced trajectories using dynamic time warping to ensure that they have an equal number of samples.

As seen in Figure 7.5, the proposed approach consistently outperforms the baseline. Specifically, the errors of our framework diminish to zero for all initial configurations. In contrast, the baseline's error stays relatively high for all the links except the end-effector (Link 7). This is expected since the baseline has no incentive to force the other links to track the demonstration. Moreover, the baseline resulted in poorer performances when the the initial configuration of the evaluation scenario was different than that of the demonstration (*init-1* and *init-2*).

Finally, we validated the ability of the proposed approach to avoid obstacles. Figure 7.6 demonstrates this behavior when an obstacle (shown as a sphere) hinders the demonstrated path. As seen, the motion generated by proposed approach reaches the desired goal while simultaneously avoiding collisions on the entire arm.

7.5 Discussion

We introduced a coordinate-free learning and motion generation framework for complex tasks in constrained environments. This approach is capable of simultaneously learning inherently-stable reactive polices in multiple subtask spaces directly from an expert demonstration, while explicitly considering the underlying geometries. The structured learning set up here allows us to incorporate human guided expert data in to specific policies that are interpretable given that any of them can be analyzed in their respective task spaces. Furthermore, the compartmentalization offered by the structure allows us to carefully specify policies for safety, such that they cannot be interfered with by the learning process, thus ensuing safety during skill acquisition and reproduction. Experiments demonstrate that this approach can capture the desired behaviors of multiple robot links in order to accomplish a constrained manipulation task. Qualitative and quantitative comparisons indicate that our approach consistently outperforms a baseline method that only learns an end-effector policy. While here we focused on individually learning the RMPs from human demonstrations, the strategy presented in the previous chapter can be utilized to learn such RMPs in an end-to-end fashion (by parameterizing them as structured neural networks) where high dimensional data like images could be used, and remains future work.

CHAPTER 8
CONCLUSION

Based on current trends this thesis recognized the split between traditional methods and modern learning-based approaches towards robot motion generation. Building on strong mathematical foundations and prior work, I presented two novel techniques for robot motion generation that leverage structure to bridge the gap between the two paradigms, such that we can combine their strengths while mitigating each others weaknesses. Structure can be an overloaded term, however in this thesis it manifested as a tool that allows us to incorporate learning into our algorithms alongside domain knowledge and problem constraints.

Specifically, the first technique I presented, views motion planning with a lens of probabilistic inference. Here the prior distribution describes desired motion while the likelihood describes other costs and constraints. This problem is constructed using factor graphs and leads to an efficient planning algorithm. The structure induced by factor graphs is then leveraged to bring learning into this framework, by allowing learning factors from human demonstrations, as well as learning factors in an end-to-end setting. The second technique I presented, brings together dynamical systems and geometric mechanics for reactive policy synthesis. A full task policy is achieved by designing appropriate sub-task behaviors through dynamical systems, and then consistently and efficiently combining them. This is done in part by representing the full task space decomposed as a task-map tree. The structure induced by task-map trees is later utilized within this framework, to learn weight functions along the tree in an end-to-end fashion for policy fusion, and to also learn individual sub-task policies from human demonstrations. Across all these approaches the various benefits of structured learning, like interpretability, incorporation of constraints such as safety even during the learning process, and data efficiency, are shown.

While the methods presented here further the state-of-the-art and have provided useful

insights, they are not without their limitations as was discussed in relevant chapters. There are still many technical challenges ahead that are excellent directions for future work. Some of them were alluded to in previous chapters, but here we discuss a few overarching themes that arise.

How do we plan better?

For problems where we have to deal with high speed tasks, considering the underling dynamics and closely integrating planning with impedance control is critical. The algorithms presented here internally rely on least squares style optimization, and would need to be extended to handle hard requirements like dynamics, underactuation, and nonholonomic constraints, while at the same time following structured ways of integrating learning. This further open doors to problems that also involve discrete constraints, for example, in task planning, or when dealing with contact during manipulation.

Through the two major techniques in this thesis, we have seen that planning and reactive policies are very complementary. So far they were utilized independently, however, an ideal system would greatly benefit from using them both. The interesting research questions arise in finding ways to seamlessly integrate them. Loosely speaking, there are connections between GPMP2 and RMPflow. The covariance of a factor bears resemblance to a Riemannian metric, and the inference on a factor graph is reminiscent of the forward-backward pass on a task-map tree. The critical difference is that the former is built over a long time horizon while the latter operates over a single time step. Exploring this connection more formally can lead to possibilities of combining planning and reactive motion.

How do we learn better?

While we have currently explored the utility of structured learning more formal studies should be conducted that can inform on finding the right mix of priors and known models vs learning within the structure. Along these lines it is also still unclear whether this structure

should evolve as the system improves. The structure is helpful for data efficiency but it may also introduce model biases. It is also not clear if it is more effective to learn with an unstructured approach if say we have access to really large amounts of data and compute.

Being able to utilize data from multiple sources (beyond human demonstrations) will be an asset, as limitations in hardware can often lead to poor quality demonstrations. We relied primarily on data collected from expert algorithms or human users to learn via imitation. This is not always possible, and therefore it is beneficial to investigate strategies in extending the ideas presented here to leverage reinforcement learning as well.

How do we perceive better?

The quality and efficiency of motion generation partly depends on reliable perception. Therefore having a tighter coupling between them is highly necessary. We have shown this is possible with the GPMP2 framework [87], however RMPflow currently deals with estimation externally. The connection between the two discussed above could help cross pollinate ideas to incorporate uncertainty handling. RMPflow being reactive is able to handle dynamic environments, however GPMP2, and by extension CLAMP and dGPMP2, currently rely on a precomputed SDF for collision checking. For large scenes this is infeasible to evaluate online in a dynamic environment and thus requires research into generating SDFs incrementally online by leveraging GPUs.

Learning from vision has largely been restricted to 2D single view images (usually from the top). Incorporating modern learning tools to handle 3D scenes especially in the case of manipulators can address challenges in accelerating collision avoidance and planning in general. Beyond vision and joint encoders there are many other sensor modalities that we haven't explored with our systems. Tactile information can be of particular importance for problems in contact rich manipulation. Audio is also another sensor, less commonly used. The ability to fuse such diverse sensory information can highly benefit the motion generation process.

Getting robots to work and move in the real world means on some level we have to tackle motion generation, which after being around and constantly evolving since the early days of robotics, has still remained one of its most difficult challenges. Throughout this thesis the underlying philosophy has been to incorporate learning in a manner that adds value over traditional methods. Consequently, the techniques and algorithms presented here, brings us closer to solving motion generation and pushing the capabilities of current systems.

Appendices

APPENDIX A

PRIOR AND SPARSITY IN GAUSSIAN PROCESS MOTION PLANNING

A.1 The trajectory prior

First, we review conditioning a distribution of state $\boldsymbol{\theta}$ on observations $\boldsymbol{Y}$ in general (for a full treatment see [35]). Let the observation be given by the following linear equation

$$\boldsymbol{Y} = \mathbf{C}\boldsymbol{\theta} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \widetilde{\boldsymbol{\mathcal{K}}}_y). \tag{A.1}$$

We can write their joint distribution as

$$\mathcal{N}\left(\begin{bmatrix} \widetilde{\boldsymbol{\mu}} \\ \mathbf{C}\widetilde{\boldsymbol{\mu}} \end{bmatrix}, \begin{bmatrix} \widetilde{\boldsymbol{\mathcal{K}}} & \widetilde{\boldsymbol{\mathcal{K}}}\mathbf{C}^\top \\ \mathbf{C}\widetilde{\boldsymbol{\mathcal{K}}} & \mathbf{C}\widetilde{\boldsymbol{\mathcal{K}}}\mathbf{C}^\top + \widetilde{\boldsymbol{\mathcal{K}}}_y \end{bmatrix} \right). \tag{A.2}$$

The distribution of the state conditioned on the observations is then $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\mathcal{K}})$ where

$$\boldsymbol{\mu} = \widetilde{\boldsymbol{\mu}} + \widetilde{\boldsymbol{\mathcal{K}}}\mathbf{C}^\top(\mathbf{C}\widetilde{\boldsymbol{\mathcal{K}}}\mathbf{C}^\top + \widetilde{\boldsymbol{\mathcal{K}}}_y)^{-1}(\boldsymbol{Y} - \mathbf{C}\widetilde{\boldsymbol{\mu}}) \tag{A.3}$$

$$\boldsymbol{\mathcal{K}} = \widetilde{\boldsymbol{\mathcal{K}}} - \widetilde{\boldsymbol{\mathcal{K}}}\mathbf{C}^\top(\mathbf{C}\widetilde{\boldsymbol{\mathcal{K}}}\mathbf{C}^\top + \widetilde{\boldsymbol{\mathcal{K}}}_y)^{-1}\mathbf{C}\widetilde{\boldsymbol{\mathcal{K}}} \tag{A.4}$$

Now, we are interested in conditioning just on the goal state $\boldsymbol{\theta}_N$ with mean $\boldsymbol{\mu}_N$ and covariance $\boldsymbol{\mathcal{K}}_N$. Therefore in the above equations we use $\mathbf{C} = [\mathbf{0} \quad \dots \quad \mathbf{0} \quad \mathbf{I}]$ and $\widetilde{\boldsymbol{\mathcal{K}}}_y = \boldsymbol{\mathcal{K}}_N$ to get

$$\boldsymbol{\mu} = \widetilde{\boldsymbol{\mu}} + \widetilde{\boldsymbol{\mathcal{K}}}(t_N, \boldsymbol{t})^\top(\widetilde{\boldsymbol{\mathcal{K}}}(t_N, t_N) + \boldsymbol{\mathcal{K}}_N)^{-1}(\boldsymbol{\theta}_N - \boldsymbol{\mu}_N) \tag{A.5}$$

$$\boldsymbol{\mathcal{K}} = \widetilde{\boldsymbol{\mathcal{K}}} - \widetilde{\boldsymbol{\mathcal{K}}}(t_N, \boldsymbol{t})^\top(\widetilde{\boldsymbol{\mathcal{K}}}(t_N, t_N) + \boldsymbol{\mathcal{K}}_N)^{-1}\widetilde{\boldsymbol{\mathcal{K}}}(t_N, \boldsymbol{t}) \tag{A.6}$$

where $\widetilde{\boldsymbol{\mathcal{K}}}(t_N, \boldsymbol{t}) = [\widetilde{\boldsymbol{\mathcal{K}}}(t_N, t_0) \quad \dots \quad \widetilde{\boldsymbol{\mathcal{K}}}(t_N, t_N)]]$.

Using the Woodbury matrix identity we can write Eq. (A.4) as

$$\boldsymbol{\mathcal{K}} = (\widetilde{\boldsymbol{\mathcal{K}}}^{-1} + \mathbf{C}^\top \widetilde{\boldsymbol{\mathcal{K}}}_y^{-1} \mathbf{C})^{-1} \tag{A.7}$$

and substituting $\mathbf{C}$ and $\widetilde{\boldsymbol{\mathcal{K}}}_y$ as before for conditioning on the goal we get

$$\boldsymbol{\mathcal{K}} = \left(\widetilde{\boldsymbol{\mathcal{K}}}^{-1} + [\mathbf{0} \quad \dots \quad \mathbf{0} \quad \mathbf{I}]^\top \boldsymbol{\mathcal{K}}_N^{-1} [\mathbf{0} \quad \dots \quad \mathbf{0} \quad \mathbf{I}] \right)^{-1}. \tag{A.8}$$

From [71] we know that the precision matrix of the distribution obtained from the LTV-SDE in Eq. (2.5) can be decomposed as $\widetilde{\boldsymbol{\mathcal{K}}}^{-1} = \widetilde{\mathbf{A}}^{-\top} \widetilde{\mathbf{Q}}^{-1} \widetilde{\mathbf{A}}^{-1}$. Therefore,

$$\boldsymbol{\mathcal{K}}^{-1} = \begin{bmatrix} \widetilde{\mathbf{A}}^{-1} \\ \mathbf{0} \quad \dots \quad \mathbf{0} \quad \mathbf{I} \end{bmatrix}^\top \begin{bmatrix} \widetilde{\mathbf{Q}}^{-1} & \\ & \boldsymbol{\mathcal{K}}_N^{-1} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}}^{-1} \\ \mathbf{0} \quad \dots \quad \mathbf{0} \quad \mathbf{I} \end{bmatrix} \tag{A.9}$$

$$= \mathbf{B}^\top \mathbf{Q}^{-1} \mathbf{B} \tag{A.10}$$

where

$$\mathbf{B} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ -\boldsymbol{\Phi}(t_1, t_0) & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{\Phi}(t_2, t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & -\boldsymbol{\Phi}(t_N, t_{N-1}) & \mathbf{I} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{A.11}$$

and

$$\mathbf{Q}^{-1} = \operatorname{diag}(\boldsymbol{\mathcal{K}}_0^{-1}, \mathbf{Q}_{0,1}^{-1}, \dots, \mathbf{Q}_{N-1,N}^{-1}, \boldsymbol{\mathcal{K}}_N^{-1}), \tag{A.12}$$

$$\mathbf{Q}_{a,b} = \int_{t_a}^{t_b} \boldsymbol{\Phi}(b, s) \mathbf{F}(s) \mathbf{Q}_c \mathbf{F}(s)^\top \boldsymbol{\Phi}(b, s)^\top \, \mathrm{d}s \tag{A.13}$$

A.2 Sparsity of the likelihood

In Eq. (2.52) we argue that matrix $\boldsymbol{\mathcal{K}}^{-1} + \mathbf{H}^\top \boldsymbol{\Sigma}^{-1} \mathbf{H}$ is sparse. In Section 2.4.2, we proved the block-tridiagonal property of $\boldsymbol{\mathcal{K}}^{-1}$. In this section we prove that $\mathbf{H}^\top \boldsymbol{\Sigma}^{-1} \mathbf{H}$ is also block-tridiagonal.

Given the isotropic definition of $\boldsymbol{\Sigma}$ in Eq. (2.47) and Eq. (2.62)

$$\mathbf{H}^\top \boldsymbol{\Sigma}^{-1} \mathbf{H} = \sigma_{obs}^{-2} \mathbf{H}^\top \mathbf{H}. \tag{A.14}$$

Given the definition of $\boldsymbol{h}(\boldsymbol{\theta})$ in Eq. (2.46), the size of $\mathbf{H}$ is $M \times (N + 1 + N \times n_{ip})$ by $(N + 1) \times D$, therefore $\mathbf{H}^\top \mathbf{H}$ has size $(N + 1) \times D$ by $(N + 1) \times D$.

For simplicity, we partition $\mathbf{H}$ and $\mathbf{H}^\top \mathbf{H}$ by forming blocks corresponding to the system DOF D, and dimensionality M of $\boldsymbol{h}$, and work with these block matrices in the remaining section. So $\mathbf{H}$ and $\mathbf{H}^\top \mathbf{H}$ have block-wise size $N + 1 + N \times n_{ip}$ by $N + 1$ and $N + 1$ by $N + 1$ respectively. We define $\mathbf{A}(i, j)$ to be the block element at row i and column j of $\mathbf{A}$.

Given the definition of $\boldsymbol{h}(\boldsymbol{\theta})$ in Eq. (2.46), each element of $\mathbf{H}$ is defined by

$$\mathbf{H}(i, j) = \frac{\partial \mathbf{h}(\boldsymbol{\theta}_{s_i})}{\partial \boldsymbol{\theta}_j}\Big|_{\boldsymbol{\theta}} \tag{A.15}$$

for rows contain regular obstacle factors, where s_i is the support state index connects the regular obstacle factor of row i, or

$$\mathbf{H}(i, j) = \frac{\partial \mathbf{h}^{intp}(\boldsymbol{\theta}_{s_i}, \boldsymbol{\theta}_{s_i+1})}{\partial \boldsymbol{\theta}_j}\Big|_{\boldsymbol{\theta}} \tag{A.16}$$

for rows contain interpolated obstacle factors, where s_i is the before support state index of interpolated obstacle factor of row i. Since $\mathbf{h}(\boldsymbol{\theta}_{s_i})$ is only a function of $\boldsymbol{\theta}_{s_i}$, and $\mathbf{h}^{intp}(\boldsymbol{\theta}_{s_i}, \boldsymbol{\theta}_{s_i+1})$ is only function of $\boldsymbol{\theta}_{s_i}$ and $\boldsymbol{\theta}_{s_i+1}$, they have zero partial derivatives with

respect to any other states in $\boldsymbol{\theta}$, so for any block element in $\mathbf{H}$

$$\mathbf{H}(i,j) = \mathbf{0}, \text{ if } j \neq s_i \text{ or } s_i + 1. \tag{A.17}$$

For each block element in $\mathbf{H}^\top\mathbf{H}$

$$\mathbf{H}^\top\mathbf{H}(i,j) = \sum_{k=1}^{N+1+N\times n_{ip}} \mathbf{H}^\top(i,k)\mathbf{H}(k,j) \tag{A.18}$$

$$= \sum_{k=1}^{N+1+N\times n_{ip}} \mathbf{H}(k,i)^\top\mathbf{H}(k,j). \tag{A.19}$$

For each k, non-zero $\mathbf{H}(k,i)^\top\mathbf{H}(k,j)$ is possible when the following condition is satisfied,

$$\{i = s_k \text{ or } s_k + 1\} \text{ and } \{j = s_k \text{ or } s_k + 1\}. \tag{A.20}$$

So for non-zero $\mathbf{H}^\top\mathbf{H}(i,j)$

$$|i - j| \leq 1, \tag{A.21}$$

since if i and j has difference larger than 1, Eq. (A.20) is unsatisfied on every k, so $\mathbf{H}^\top\mathbf{H}(i,j)$ will be zero based on Eq. (A.19). Given we know that $\mathbf{H}^\top\mathbf{H}$ is block tridiagonal, and Eq. (A.14), we have proved that $\mathbf{H}^\top\boldsymbol{\Sigma}^{-1}\mathbf{H}$ is also block tridiagonal.

APPENDIX B

GEOMETRIC DYNAMICAL SYSTEMS

Here we summarize details and properties of GDSs introduced in Section 5.7.

B.1 From geometric mechanics to GDSs

Our study of GDSs is motivated by geometric mechanics. Many formulations of mechanics exist, including Lagrangian mechanics [153] and the aforementioned Gauss' Principle of Least Constraint [151]——They are all equivalent, implicitly sharing the same mathematical structure. In that sense, geometric mechanics, which models physical systems as geodesic flow on Riemannian manifolds, is the most explicit of these, revealing directly the underlying manifold structure and connecting to the broad mathematical tool set from Riemannian geometry. These connections enable us here to generalize beyond the previous simple mechanical systems studied in [137] to non-classical systems that more naturally describe robotic behaviors with non-Euclidean geometric properties.

B.2 Degenerate GDSs

Let us recall the definition of GDSs.

Definition 1. *Let* $\mathbf{B} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m\times m}$ *and let* $\mathbf{G} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m\times m}$ *and* $\Phi : \mathbb{R}^m \to \mathbb{R}$ *be differentiable. We say the tuple* $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)$ *is a* GDS *if*

$$\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})\ddot{\mathbf{x}} + \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) = -\nabla_{\mathbf{x}}\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \tag{B.1}$$

where $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})$.

For degenerate cases, $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})$ can be singular and Eq. (B.1) define rather a family of

differential equations. Degenerate cases are not uncommon; for example, the leaf-node dynamics could have $\mathbf{G}$ being only positive semidefinite. Having degenerate GDSs does not change the properties that we have proved, but one must be careful about whether differential equation satisfying Eq. (B.1) exist. For example, the existence is handled by the assumption on $\mathbf{M}$ in Theorem 1 and the assumption on $\mathbf{M}_r$ in Corollary 1. For RMPflow, we only need that $\mathbf{M}_r$ at the root node is non-singular. In other words, the natural-form RMP created by `pullback` at the root node can be resolved in the canonical-form RMP for policy execution. A sufficient and yet practical condition is provided in Theorem 2.

B.3 Geodesic and stability

For GDSs, they possess a natural conservation property of kinematic energy, i.e. it travels along a geodesic defined by $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ when there is no external perturbations due to Φ and $\mathbf{B}$. Note $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$ by definition may only be positive-semidefinite even when the system is non-degenerate; here we allow the geodesic to be defined for a degenerate metric, meaning a curve whose instant length measured by the (degenerate) metric is constant.

This geometric feature is an important tool to establish the stability of non-degenerate GDSs; We highlight this nice geometric property below, which is a corollary of Proposition 1.

Corollary 3. *All non-degenerate GDSs in the form* $(\mathcal{M}, \mathbf{G}, 0, 0)$ *travel on geodesics. That is,* $\dot{K}(\mathbf{x}, \dot{\mathbf{x}}) = 0$*, where* $K(\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2}\dot{\mathbf{x}}^\top \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}$.

Note that this property also hold for degenerate GDSs provided that differential equations satisfying Eq. (B.1) exist.

B.4 Curvature term and Coriolis force

The curvature term $\boldsymbol{\xi}_{\mathbf{G}}$ in GDSs is highly related to the Coriolis force in the mechanics literature. This is not surprising, as from the analysis in [138] we know that $\boldsymbol{\xi}_{\mathbf{G}}$ comes

from the Christoffel symbols of the asymmetric connection. Recall it is defined as

$$\boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) \coloneqq \overset{\mathbf{x}}{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} - \tfrac{1}{2}\nabla_{\mathbf{x}}(\dot{\mathbf{x}}^{\top}\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}})$$

We show their relationship explicitly below.

Lemma 1. *Let* $\Gamma_{ijk} = \frac{1}{2}(\partial_{x_k}G_{ij} + \partial_{x_j}G_{ik} - \partial_{x_j}G_{jk})$ *be the Christoffel symbol of the first kind with respect to* $\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}})$*, where the subscript* $_{ij}$ *denotes the* (i, j) *element. Let* $C_{ij} = \sum_{k=1}^{d} \dot{x}_k \Gamma_{ijk}$ *and define* $\mathbf{C}(\mathbf{x}, \dot{\mathbf{x}}) = (C_{ij})_{i,j=1}^{m}$*. Then* $\boldsymbol{\xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}$*.*

Proof of Lemma 1. Suppose $\boldsymbol{\xi}_{\mathbf{G}} = (\xi_i)_{i=1}^{m}$. We can compare the two definitions and verify they are indeed equivalent:

$$\begin{aligned}
\xi_i &= \sum_{j,k=1}^{d} \dot{x}_j \dot{x}_k \partial_{x_j} G_{ik} - \frac{1}{2}\sum_{j,k=1}^{d} \dot{x}_j \dot{x}_k \partial_{x_i} G_{jk} \\
&= \frac{1}{2}\sum_{j,k=1}^{d} \dot{x}_j \dot{x}_k \partial_{x_k} G_{ij} + \frac{1}{2}\sum_{j,k=1}^{d} \dot{x}_j \dot{x}_k \partial_{x_j} G_{ik} - \frac{1}{2}\sum_{j,k=1}^{d} \dot{x}_j \dot{x}_k \partial_{x_i} G_{jk} \\
&= \left(\mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}\right)_i
\end{aligned}$$

■

APPENDIX C

PROOFS OF RMPFLOW ANALYSIS

C.1 Proof of Theorem 1

Theorem 1. *Let the ith child node follow $(\mathcal{N}_i, \mathbf{G}_i, \mathbf{B}_i, \Phi_i)_{\mathcal{S}_i}$ and have coordinate $\mathbf{y}_i$. Let $\mathbf{f}_i = -\boldsymbol{\eta}_{\mathbf{G}_i;\mathcal{S}_i} - \nabla_{\mathbf{y}_i}\Phi_i - \mathbf{B}_i\dot{\mathbf{y}}_i$ and $\mathbf{M}_i = \mathbf{G}_i + \boldsymbol{\Xi}_{\mathbf{G}_i}$. If $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$ of the parent node is given by* `pullback` *with $\{[\mathbf{f}_i, \mathbf{M}_i]^{\mathcal{N}_i}\}_{i=1}^K$ and $\mathbf{M}$ is non-singular, the parent node follows $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$, where $\mathbf{G} = \sum_{i=1}^K \mathbf{J}_i^\top \mathbf{G}_i \mathbf{J}_i$, $\mathbf{B} = \sum_{i=1}^K \mathbf{J}_i^\top \mathbf{B}_i \mathbf{J}_i$, $\Phi = \sum_{i=1}^K \Phi_i \circ \mathbf{y}_i$, $\mathcal{S}$ preserves $\mathcal{S}_i$, and $\mathbf{J}_i = \partial_{\mathbf{x}}\mathbf{y}_i$. Particularly, if $\mathbf{G}_i$ is velocity-free and the child nodes are GDSs, the parent node follows $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)$.*

Proof of Theorem 1. We will use the non-degeneracy assumption that $\mathbf{G} + \boldsymbol{\Xi}_{\mathbf{G}}$ (i.e. $\mathbf{M}$ as we will show) is non-singular, so that the differential equation specified by an RMP in normal form or a (structured) GDS is unique. This assumption is made to simplify writing. At the end of the proof, we will show that this assumption only needs to be true at the root node of RMPflow.

The general case We first show the differential equation given by `pullback` is equivalent to the differential equation of pullback structured GDS $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$. Under the non-degeneracy assumption, suppose $\mathcal{S}_i$ factorizes $\mathbf{G}_i$ as $\mathbf{G}_i = \mathbf{L}_i^\top \mathbf{H}_i \mathbf{L}_i$, where $\mathbf{L}_i$ is some Jacobian matrix. On one hand, for `pullback`, because in the child node $\ddot{\mathbf{y}}_i$ satisfies $(\mathbf{G}_i + \boldsymbol{\Xi}_{\mathbf{G}_i})\ddot{\mathbf{y}}_i = -\boldsymbol{\eta}_{\mathbf{G}_i;\mathcal{S}_i} - \nabla_{\mathbf{y}_i}\Phi_i - \mathbf{B}_i\dot{\mathbf{y}}_i$ (where by definition $\boldsymbol{\eta}_{\mathbf{G}_i;\mathcal{S}_i} = \mathbf{L}_i^\top(\boldsymbol{\xi}_{\mathbf{H}_i} + (\mathbf{H}_i + \boldsymbol{\Xi}_{\mathbf{H}_i})\dot{\mathbf{L}}_i\dot{\mathbf{y}}_i)$), the `pullback` operator combines the child nodes into the differential equation at the parent node,

$$\mathbf{M}\ddot{\mathbf{x}} = \sum_{i=1}^K \mathbf{J}_i^\top \mathbf{M}_i(\ddot{\mathbf{y}}_i - \dot{\mathbf{J}}_i\dot{\mathbf{x}}) \tag{C.1}$$

where we recall $\mathbf{M} = \sum_{i=1}^{K} \mathbf{J}_i^\top \mathbf{M}_i \mathbf{J}_i$ is given by `pullback`. On the other hand, for $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$ with $\mathcal{S}$ preserving $\mathcal{S}_i$, its dynamics satisfy

$$(\mathbf{G} + \boldsymbol{\Xi}_{\mathbf{G}})\,\ddot{\mathbf{x}} + \boldsymbol{\eta}_{\mathbf{G};\mathcal{S}} = -\nabla_{\mathbf{x}}\Phi - \mathbf{B}\dot{\mathbf{x}} \tag{C.2}$$

where $\mathbf{G}$ is factorized by $\mathcal{S}$ into

$$\mathbf{G} = \begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_K \end{bmatrix}^\top \begin{bmatrix} \mathbf{G}_1 & & \\ & \ddots & \\ & & \mathbf{G}_K \end{bmatrix} \begin{bmatrix} \mathbf{J}_1 \\ \vdots \\ \mathbf{J}_K \end{bmatrix} = \begin{bmatrix} \mathbf{L}_1\mathbf{J}_1 \\ \vdots \\ \mathbf{L}_K\mathbf{J}_K \end{bmatrix}^\top \begin{bmatrix} \mathbf{H}_1 & & \\ & \ddots & \\ & & \mathbf{H}_K \end{bmatrix} \begin{bmatrix} \mathbf{L}_1\mathbf{J}_1 \\ \vdots \\ \mathbf{L}_K\mathbf{J}_K \end{bmatrix} =: \bar{\mathbf{J}}^\top \bar{\mathbf{H}} \bar{\mathbf{J}}$$

and the curvature term $\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}$ by $\mathcal{S}$ is given as $\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}} \coloneqq \bar{\mathbf{J}}^\top(\boldsymbol{\xi}_{\bar{\mathbf{H}}} + (\bar{\mathbf{H}} + \boldsymbol{\Xi}_{\bar{\mathbf{H}}})\dot{\bar{\mathbf{J}}}\dot{\mathbf{x}})$.

To prove the general statement, we will show Eq. (C.1) and Eq. (C.2) are equivalent. First, we introduce a lemma to write $\boldsymbol{\Xi}_{\mathbf{G}}$ in terms of $\boldsymbol{\Xi}_{\mathbf{G}_i}$ (proved later in this section).

Lemma 2. *Let $\mathcal{M}$ and $\mathcal{N}$ be two manifolds and let $\mathbf{x}$ and $\mathbf{y}(\mathbf{x})$ be the coordinates. Define $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{J}(\mathbf{x})^\top \mathbf{N}(\mathbf{y}, \dot{\mathbf{y}})\mathbf{J}(\mathbf{x})$, where $\mathbf{J}(\mathbf{x}) = \partial_{\mathbf{x}}\mathbf{y}(\mathbf{x})$. Then*

$$\boldsymbol{\Xi}_{\mathbf{M}}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{J}^\top(\mathbf{x})\boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}})\mathbf{J}(\mathbf{x})$$

Therefore, we see that on the LHSs

$$(\mathbf{G} + \boldsymbol{\Xi}_{\mathbf{G}})\ddot{\mathbf{x}} = \mathbf{M}\ddot{\mathbf{x}}$$

and on the RHSs

$$
\begin{aligned}
&\left(\sum_{i=1}^{K} \mathbf{J}_i^\top \mathbf{M}_i(\ddot{\mathbf{y}}_i - \dot{\mathbf{J}}_i\dot{\mathbf{x}})\right) \\
&= \left(\sum_{i=1}^{K} \mathbf{J}_i^\top(-\boldsymbol{\eta}_{\mathbf{G}_i;\mathcal{S}_i} - \nabla_{\mathbf{y}_i}\Phi_i - \mathbf{B}_i\dot{\mathbf{y}}_i - (\mathbf{G}_i + \boldsymbol{\Xi}_{\mathbf{G}_i})\dot{\mathbf{J}}_i\dot{\mathbf{x}})\right) \\
&= \left(\sum_{i=1}^{K} \mathbf{J}_i^\top(-\mathbf{L}_i^\top(\boldsymbol{\xi}_{\mathbf{H}_i} + (\mathbf{H}_i + \boldsymbol{\Xi}_{\mathbf{H}_i})\dot{\mathbf{L}}_i\dot{\mathbf{y}}_i) - (\mathbf{G}_i + \boldsymbol{\Xi}_{\mathbf{G}_i})\dot{\mathbf{J}}_i\dot{\mathbf{x}})\right) \\
&\quad + \left(\sum_{i=1}^{K} \mathbf{J}_i^\top(-\nabla_{\mathbf{y}_i}\Phi_i - \mathbf{B}_i\dot{\mathbf{y}}_i)\right) \\
&= \left(\sum_{i=1}^{K} -\bar{\mathbf{J}}_i^\top\boldsymbol{\xi}_{\mathbf{H}_i} - \bar{\mathbf{J}}_i^\top(\mathbf{H}_i + \boldsymbol{\Xi}_{\mathbf{H}_i})(\dot{\mathbf{L}}_i\mathbf{J}_i + \mathbf{L}_i\dot{\mathbf{J}}_i)\dot{\mathbf{x}}\right) - \nabla_{\mathbf{x}}\Phi - \mathbf{B}\dot{\mathbf{x}} \\
&= -\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}} - \nabla_{\mathbf{x}}\Phi - \mathbf{B}\dot{\mathbf{x}}
\end{aligned}
$$

where the first equality is due to Lemma 2, the second equality is due to Eq. (C.1), and the third equality is due to the definition of structured GDSs. The above derivations show the equivalence between the RHSs and LHSs of Eq. (C.1) and Eq. (C.2), respectively. Therefore, when the non-degenerate assumption holds, Eq. (C.1) and Eq. (C.2) are equivalent.

The special case With the closure of structured GDSs proved, we next show the closure of GDSs under `pullback`, when the metric is only configuration-dependent. That is, we want to show that, when the metric is only configuration-dependent, the choice of structure does not matter. This amounts to show that $\boldsymbol{\xi}_{\mathbf{G}} = \boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}$ because by definition $\boldsymbol{\Xi}_i = 0$ and $\boldsymbol{\Xi} = 0$. Below we show how $\boldsymbol{\xi}_{\mathbf{G}}$ is written in terms of $\boldsymbol{\xi}_{\mathbf{G}_i}$ and $\boldsymbol{\Xi}_{\mathbf{G}_i}$ for general metric matrices and specialize it to the configuration-dependent special case (proved later in this section).

Lemma 3. *Let $\mathcal{M}$ and $\mathcal{N}$ be two manifolds and $\mathbf{x}$ and $\mathbf{y}(\mathbf{x})$ be the coordinates. Suppose*

$\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})$ *is structured as* $\mathbf{J}(\mathbf{x})^\top \mathbf{N}(\mathbf{y}, \dot{\mathbf{y}}) \mathbf{J}(\mathbf{x})$*, where* $\mathbf{J}(\mathbf{x}) = \partial_{\mathbf{x}} \mathbf{y}(\mathbf{x})$*. Then*

$$\begin{aligned}\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{J}(\mathbf{x})^\top \Big(\boldsymbol{\xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}}) + (\mathbf{N}(\mathbf{y}, \dot{\mathbf{y}}) + 2\boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}}))\dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \Big) \\ - \dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})^\top \boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}})^\top \mathbf{J}(\mathbf{x})\dot{\mathbf{x}}\end{aligned}$$

When $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{M}(\mathbf{x})$*,* $\boldsymbol{\xi}_{\mathbf{M}} = \boldsymbol{\eta}_{\mathbf{M};\mathcal{S}}$ *regardless of the structure of* $\mathcal{S}$*.*

By Lemma 3, we see that structured GDSs are GDSs regardless of the chosen structure when the metric is only configuration dependent. Thus, the statement of the special case follows by combining Lemma 3 and the previous proof for structured GDSs.

Remarks: Proof of Corollary 1 We note that the non-degenerate assumption does not need to hold for every nodes in RMPflow but only for the root node. This can be seen from the proof above, where we propagate the LHSs and RHSs *separately*. Therefore, as long as the inertial matrix at the root node is invertible, the differential equation on the configuration space is well defined. ■

Proof of Lemma 2. Let $\mathbf{m}_i$, $\mathbf{n}_i$, $\mathbf{j}_i$ be the ith column of $\mathbf{M}$, $\mathbf{N}$, and $\mathbf{J}$, respectively. Suppose $\mathcal{M}$ and $\mathcal{N}$ are of m and n dimensions, respectively. By definition of $\boldsymbol{\Xi}_{\mathbf{M}}$,

$$\begin{aligned}2\boldsymbol{\Xi}_{\mathbf{M}}(\mathbf{x}, \dot{\mathbf{x}}) = \sum_{i=1}^{m} \dot{x}_i \partial_{\dot{\mathbf{x}}} \mathbf{m}_i(\mathbf{x}, \dot{\mathbf{x}}) &= \mathbf{J}(\mathbf{x})^\top \sum_{i=1}^{m} \dot{x}_i \partial_{\dot{\mathbf{x}}} (\mathbf{N}(\mathbf{y}, \dot{\mathbf{y}}) \mathbf{j}_i(\mathbf{x})) \\ &= \mathbf{J}(\mathbf{x})^\top \left(\sum_{i=1}^{m} \dot{x}_i \partial_{\dot{\mathbf{y}}} (\mathbf{N}(\mathbf{y}, \dot{\mathbf{y}}) \mathbf{j}_i(\mathbf{x})) \right) \mathbf{J}(\mathbf{x}) \\ &= \mathbf{J}(\mathbf{x})^\top \left(\sum_{j=1}^{n} \partial_{\dot{\mathbf{y}}} \mathbf{n}_j(\mathbf{y}, \dot{\mathbf{y}}) \sum_{i=1}^{m} \dot{x}_i J_{ji}(\mathbf{x}) \right) \mathbf{J}(\mathbf{x}) \\ &= \mathbf{J}(\mathbf{x})^\top \left(\sum_{j=1}^{n} y_j \partial_{\dot{\mathbf{y}}} \mathbf{n}_j(\mathbf{y}, \dot{\mathbf{y}}) \right) \mathbf{J}(\mathbf{x}) \\ &= 2\mathbf{J}(\mathbf{x})^\top \boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}}) \mathbf{J}(\mathbf{x})\end{aligned}$$

■

Proof of Lemma 3. Before the proof, we first note a useful identity $\partial_{\mathbf{x}} \dot{\mathbf{y}} = \dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})$. This

can be derived simply by the definition of the Jacobian matrix $(\partial_{\mathbf{x}}\mathbf{J}(\mathbf{x})\dot{\mathbf{x}})_{ij} = \sum_{k=1}^{m} \dot{x}_k \partial_{x_j} J_{ik} = \sum_{k=1}^{m} \dot{x}_k \partial_{x_j}\partial_{x_k} y_i = \sum_{k=1}^{m} \dot{x}_k \partial_{x_k} J_{ij} = (\dot{\mathbf{J}})_{ij}$.

To prove the lemma, we derive $\boldsymbol{\xi}_{\mathbf{M}}$ by its definition

$$
\begin{aligned}
\boldsymbol{\xi}_{\mathbf{M}} &= \overset{\text{x}}{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} - \frac{1}{2}\nabla_{\mathbf{x}}(\dot{\mathbf{x}}^\top \mathbf{M}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}}) \\
&= \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\mathbf{J}(\mathbf{x})\dot{\mathbf{x}} + \mathbf{J}(\mathbf{x})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \mathbf{J}(\mathbf{x})^\top \overset{\text{x}}{\mathbf{N}}(\mathbf{y},\dot{\mathbf{y}})\mathbf{J}(\mathbf{x})\dot{\mathbf{x}} \\
&\quad - \frac{1}{2}\nabla_{\mathbf{x}}(\dot{\mathbf{x}}^\top \mathbf{J}(\mathbf{x})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\mathbf{J}(\mathbf{x})\dot{\mathbf{x}}) \\
&= \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}} + \mathbf{J}(\mathbf{x})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \mathbf{J}(\mathbf{x})^\top \overset{\text{x}}{\mathbf{N}}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}} \\
&\quad - \frac{1}{2}\nabla_{\mathbf{x}}(\dot{\mathbf{y}}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}}) \\
&= \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}} + \mathbf{J}(\mathbf{x})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \mathbf{J}(\mathbf{x})^\top \overset{\text{x}}{\mathbf{N}}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}} \\
&\quad - \frac{1}{2}\mathbf{J}(\mathbf{x})^\top \nabla_{\mathbf{y}}(\dot{\mathbf{y}}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}}) - \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}} - \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y},\dot{\mathbf{y}})^\top \mathbf{J}(\mathbf{x})\dot{\mathbf{x}} \\
&= \mathbf{J}(\mathbf{x})^\top (\mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})\dot{\mathbf{x}} + \overset{\text{x}}{\mathbf{N}}(\mathbf{y},\dot{\mathbf{y}})\mathbf{J}(\mathbf{x})\dot{\mathbf{x}} - \frac{1}{2}\nabla_{\mathbf{y}}(\dot{\mathbf{y}}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}})) \\
&\quad - \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y},\dot{\mathbf{y}})^\top \mathbf{J}(\mathbf{x})\dot{\mathbf{x}}
\end{aligned}
$$

In the second to the last equality above, we use $\partial_{\mathbf{x}}\dot{\mathbf{y}} = \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})$ and derive

$$
\begin{aligned}
\frac{1}{2}\nabla_{\mathbf{x}}(\dot{\mathbf{y}}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}}) &= \frac{1}{2}\mathbf{J}^\top \nabla_{\mathbf{y}}(\dot{\mathbf{y}}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}}) + \frac{1}{2}\nabla_{\mathbf{x}}(\dot{\mathbf{y}})\nabla_{\dot{\mathbf{y}}}(\dot{\mathbf{y}}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}}) \\
&= \frac{1}{2}\mathbf{J}^\top \nabla_{\mathbf{y}}(\dot{\mathbf{y}}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}}) + \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}} \\
&\quad + \frac{1}{2}\dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \nabla_{\dot{\mathbf{y}}}(\mathbf{z}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{z}})|_{\mathbf{z}=\dot{\mathbf{y}}} \\
&= \frac{1}{2}\mathbf{J}^\top \nabla_{\mathbf{y}}(\dot{\mathbf{y}}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}}) + \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{y}} \\
&\quad + \dot{\mathbf{J}}(\mathbf{x},\dot{\mathbf{x}})^\top \boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y},\dot{\mathbf{y}})^\top \mathbf{J}(\mathbf{x})\dot{\mathbf{x}}
\end{aligned}
$$

as $\frac{1}{2}\partial_{\dot{\mathbf{y}}}(\mathbf{z}^\top \mathbf{N}(\mathbf{y},\dot{\mathbf{y}})\dot{\mathbf{z}})|_{\mathbf{z}=\dot{\mathbf{y}}} = \frac{1}{2}\dot{\mathbf{y}}^\top \left(\sum_{i=1}^{n} \dot{y}_i \partial_{\dot{\mathbf{y}}}\mathbf{n}_i(\mathbf{y},\dot{\mathbf{y}})\right) = \dot{\mathbf{y}}^\top \boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y},\dot{\mathbf{y}})$, where $\mathbf{n}_i$ is the ith column of $\mathbf{N}$.

To further simplify the expression, we note that by $\partial_{\mathbf{x}}\dot{\mathbf{y}} = \dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})$ we have

$$\begin{aligned}
\overset{\mathbf{x}}{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{y}} &= \sum_{i=1}^{n} \dot{y}_i \partial_{\mathbf{x}} \mathbf{n}_i(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{x}} \\
&= \sum_{i=1}^{n} \dot{y}_i (\partial_{\mathbf{y}} \mathbf{n}_i(\mathbf{y}, \dot{\mathbf{y}})\mathbf{J}(\mathbf{x})\dot{\mathbf{x}} + \partial_{\dot{\mathbf{y}}} \mathbf{n}_i(\mathbf{y}, \dot{\mathbf{y}})\partial_{\mathbf{x}}(\dot{\mathbf{y}})\dot{\mathbf{x}}) \\
&= \sum_{i=1}^{n} \dot{y}_i \partial_{\mathbf{y}} \mathbf{n}_i(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{y}} + \dot{y}_i \partial_{\dot{\mathbf{y}}} \mathbf{n}_i(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \\
&= \left(\sum_{i=1}^{n} \dot{y}_i \partial_{\mathbf{y}} \mathbf{n}_i(\mathbf{y}, \dot{\mathbf{y}}) \right) \dot{\mathbf{y}} + \left(\sum_{i=1}^{n} \dot{y}_i \partial_{\dot{\mathbf{y}}} \mathbf{n}_i(\mathbf{y}, \dot{\mathbf{y}}) \right) \dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \\
&= \overset{\mathbf{y}}{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{y}} + 2\boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}
\end{aligned}$$

Combining these two equalities, we can write

$$\begin{aligned}
&\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x}, \dot{\mathbf{x}}) = \\
&\quad \mathbf{J}(\mathbf{x})^{\top} \left(\overset{\mathbf{y}}{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{y}} - \frac{1}{2}\nabla_{\mathbf{y}}(\dot{\mathbf{y}}^{\top}\mathbf{N}(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{y}}) + (\mathbf{N}(\mathbf{y}, \dot{\mathbf{y}}) + 2\boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}}))\dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \right) \\
&\quad - \dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})^{\top}\boldsymbol{\Xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}})^{\top}\mathbf{J}(\mathbf{x})\dot{\mathbf{x}}
\end{aligned}$$

Substituting the definition of $\boldsymbol{\xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}}) = \overset{\mathbf{y}}{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{y}} - \frac{1}{2}\nabla_{\mathbf{y}}(\dot{\mathbf{y}}^{\top}\mathbf{N}(\mathbf{y}, \dot{\mathbf{y}})\dot{\mathbf{y}})$ proves the general statement.

In the special case, $\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{M}(\mathbf{x})$ (which implies $\boldsymbol{\Xi}_{\mathbf{M}} = 0$),

$$\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x}, \dot{\mathbf{x}}) = \mathbf{J}(\mathbf{x})^{\top} \left(\boldsymbol{\xi}_{\mathbf{N}}(\mathbf{y}, \dot{\mathbf{y}}) + \mathbf{N}(\mathbf{y})\dot{\mathbf{J}}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \right)$$

We show this expression is equal to $\boldsymbol{\eta}_{\mathbf{M};\mathcal{S}}$ regardless of the structure $\mathcal{S}$. This can be seen from the follows: If further $\mathbf{N}(\mathbf{y}) = \mathbf{L}(\mathbf{y})^{\top}\mathbf{C}(\mathbf{z})\mathbf{L}(\mathbf{y})$ and $\mathbf{M}$ is structured as $(\mathbf{LJ})^{\top}\mathbf{C}(\mathbf{LJ})$

from some Jacobian matrix $\mathbf{L}(\mathbf{y}) = \partial_{\mathbf{y}}\mathbf{z}$, we can write

$$\begin{aligned}
\boldsymbol{\eta}_{\mathbf{M};\mathcal{S}} &= \mathbf{J}^\top \mathbf{L}^\top (\boldsymbol{\xi}_{\bar{\mathbf{C}}} + \mathbf{C}\frac{d(\mathbf{LJ})}{dt}\dot{\mathbf{x}}) \\
&= \mathbf{J}^\top (\mathbf{L}^\top \boldsymbol{\xi}_{\bar{\mathbf{C}}} + \mathbf{L}^\top \mathbf{C}(\dot{\mathbf{L}}\mathbf{J} + \mathbf{L}\dot{\mathbf{J}})\dot{\mathbf{x}}) \\
&= \mathbf{J}^\top \left(\mathbf{L}^\top (\boldsymbol{\xi}_{\bar{\mathbf{C}}} + \mathbf{C}\dot{\mathbf{L}}\dot{\mathbf{y}}) + \mathbf{L}^\top \mathbf{C}\mathbf{L}\dot{\mathbf{J}}\dot{\mathbf{x}} \right) \\
&= \mathbf{J}^\top \left(\boldsymbol{\xi}_{\mathbf{N}} + \mathbf{N}\dot{\mathbf{J}}\dot{\mathbf{x}} \right) = \boldsymbol{\xi}_{\mathbf{M}}
\end{aligned}$$ ■

C.2 Proof of Proposition 1

Proposition 1. *For* $(\mathcal{C}, \mathbf{G}, \mathbf{B}, \Phi)_{\mathcal{S}}$, $\dot{\mathbf{V}}(\mathbf{q}, \dot{\mathbf{q}}) = -\dot{\mathbf{q}}^\top \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$.

Proof of Proposition 1. Let $K(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\dot{\mathbf{q}}^\top \mathbf{G}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$. Its time derivative can be written as

$$\begin{aligned}
\frac{d}{dt}K(\mathbf{q}, \dot{\mathbf{q}}) &= \dot{\mathbf{q}}^\top \left(\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}})\ddot{\mathbf{q}} + \frac{1}{2}(\frac{d}{dt}\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}))\dot{\mathbf{q}} \right) \\
&= \dot{\mathbf{q}}^\top \left(\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}})\ddot{\mathbf{q}} + \frac{1}{2}\sum_{i=1}^{d} \dot{q}_i \frac{d}{dt}\mathbf{g}_i(\mathbf{q}, \dot{\mathbf{q}}) \right) \\
&= \dot{\mathbf{q}}^\top \left(\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}})\ddot{\mathbf{q}} + \frac{1}{2}\sum_{i=1}^{d} \dot{q}_i \partial_{\mathbf{q}}\mathbf{g}_i(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \frac{1}{2}\sum_{i=1}^{d} \dot{q}_i \partial_{\dot{\mathbf{q}}}\mathbf{g}_i(\mathbf{q}, \dot{\mathbf{q}})\ddot{\mathbf{q}} \right) \\
&= \dot{\mathbf{q}}^\top \left((\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}}))\ddot{\mathbf{q}} + \frac{1}{2}\overset{\mathbf{q}}{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \right)
\end{aligned}$$

where we recall $\mathbf{G}$ is symmetric and $\overset{\mathbf{q}}{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}}) \coloneqq [\partial_{\mathbf{q}}\mathbf{g}_i(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}]_{i=1}^{d}$. Therefore, by definition $(\mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}}))\ddot{\mathbf{q}} = (-\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{q}, \dot{\mathbf{q}}) - \nabla_{\mathbf{q}}\Phi(\mathbf{q}) - \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}(\mathbf{q}, \dot{\mathbf{q}}))$, we can derive

$$\begin{aligned}
\frac{d}{dt}\mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) &= \frac{d}{dt}K(\mathbf{q}, \dot{\mathbf{q}}) + \dot{\mathbf{q}}^\top \nabla_{\mathbf{q}}\Phi(\mathbf{q}) \\
&= \dot{\mathbf{q}}^\top \left(-\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{q}, \dot{\mathbf{q}}) - \nabla_{\mathbf{q}}\Phi(\mathbf{q}) - \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{x}} + \frac{1}{2}\overset{\mathbf{q}}{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \nabla_{\mathbf{q}}\Phi(\mathbf{q}) \right) \\
&= -\dot{\mathbf{q}}^\top \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \dot{\mathbf{q}}^\top \left(-\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{q}, \dot{\mathbf{q}}) + \frac{1}{2}\overset{\mathbf{q}}{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} \right)
\end{aligned}$$

To finish the proof, we use two lemmas below.

Lemma 4. $\frac{1}{2}\dot{\mathbf{q}}^\top \overset{\mathbf{q}}{\dot{\mathbf{G}}}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} = \dot{\mathbf{q}}^\top \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{q},\dot{\mathbf{q}})$.

Proof of Lemma 4. This can be shown by definition:

$$\begin{aligned}
\dot{\mathbf{q}}^\top \boldsymbol{\xi}_{\mathbf{G}}(\mathbf{q},\dot{\mathbf{q}}) &= \dot{\mathbf{q}}^\top \left(\overset{\mathbf{q}}{\dot{\mathbf{G}}}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} - \frac{1}{2}\nabla_{\mathbf{q}}(\dot{\mathbf{q}}^\top \mathbf{G}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}) \right) \\
&= \sum_{k=1}^{d} \dot{q}_k \left(\sum_{i,j=1}^{d} \dot{q}_i \dot{q}_j \partial_{q_j} G_{k,i} - \frac{1}{2} \sum_{i,j=1}^{d} \dot{q}_i \dot{q}_j \partial_{q_k} G_{i,j} \right) \\
&= \sum_{i,j,k=1}^{d} \dot{q}_i \dot{q}_j \dot{q}_k \partial_{q_j} G_{k,i} - \frac{1}{2} \sum_{i,j,k=1}^{d} \dot{q}_i \dot{q}_j \dot{q}_k \partial_{q_k} G_{i,j} \\
&= \sum_{i,j,k=1}^{d} \dot{q}_i \dot{q}_j \dot{q}_k \partial_{q_k} G_{j,i} - \frac{1}{2} \sum_{i,j,k=1}^{d} \dot{q}_i \dot{q}_j \dot{q}_k \partial_{q_k} G_{i,j} \\
&= \frac{1}{2} \sum_{i,j,k=1}^{d} \dot{q}_i \dot{q}_j \dot{q}_k \partial_{q_j} G_{k,i} = \frac{1}{2} \dot{\mathbf{q}}^\top \overset{\mathbf{q}}{\dot{\mathbf{G}}}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}
\end{aligned}$$

where for the second to the last equality we use the symmetry $G_{i,j} = G_{j,i}$. ■

Using Lemma 4, we can show another equality.

Lemma 5. *For all structure $\mathcal{S}$,* $\dot{\mathbf{q}}^\top \left(-\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{q},\dot{\mathbf{q}}) + \frac{1}{2}\overset{\mathbf{q}}{\dot{\mathbf{G}}}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} \right) = 0$

Proof of Lemma 5. This can be seen from Lemma 3. Suppose $\mathcal{S}$ factorizes $\mathbf{G}(\mathbf{q},\dot{\mathbf{q}}) = \mathbf{J}(\mathbf{q})^\top \mathbf{H}(\mathbf{x},\dot{\mathbf{x}})\mathbf{J}(\mathbf{q})$ where $\mathbf{J}(\mathbf{q}) = \partial_{\mathbf{q}}\mathbf{x}$. By Lemma 3, we know

$$\boldsymbol{\xi}_{\mathbf{G}} = \mathbf{J}^\top \left(\boldsymbol{\xi}_{\mathbf{H}} + (\mathbf{H} + 2\boldsymbol{\Xi}_{\mathbf{H}})\dot{\mathbf{J}}\dot{\mathbf{x}} \right) - \dot{\mathbf{J}}^\top \boldsymbol{\Xi}_{\mathbf{H}}^\top \mathbf{J}\dot{\mathbf{x}}$$

On the other hand, by definition, we have $\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}} \coloneqq \mathbf{J}^\top(\boldsymbol{\xi}_{\mathbf{H}} + (\mathbf{H} + \boldsymbol{\Xi}_{\mathbf{H}})\dot{\mathbf{J}}\dot{\mathbf{x}})$. Therefore, by comparing the two, we can derive,

$$\dot{\mathbf{q}}^\top \boldsymbol{\xi}_{\mathbf{G}} = \dot{\mathbf{q}}^\top \left(\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}} + \mathbf{J}^\top \boldsymbol{\Xi}_{\mathbf{H}} \dot{\mathbf{J}}\dot{\mathbf{q}} - \dot{\mathbf{J}}^\top \boldsymbol{\Xi}_{\mathbf{H}}^\top \mathbf{J}\dot{\mathbf{q}} \right) = \dot{\mathbf{q}}^\top \boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}$$

Combing the above equality and Lemma 4 proves the equality. ■

Finally, we use Lemma 5 and the previous result and conclude

$$\frac{d}{dt}\mathbf{V}(\mathbf{q},\dot{\mathbf{q}}) = -\dot{\mathbf{q}}^\top \mathbf{B}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \dot{\mathbf{q}}^\top \left(-\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{q},\dot{\mathbf{q}}) + \frac{1}{2}\dot{\mathbf{G}}^{\mathbf{q}}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}}\right) = -\dot{\mathbf{q}}^\top \mathbf{B}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} \blacksquare$$

C.3 Proof of Theorem 2

Theorem 2. *Suppose every leaf node is a GDS with a metric matrix in the form* $\mathbf{R}(\mathbf{x}) + \mathbf{L}(\mathbf{x})^\top \mathbf{D}(\mathbf{x},\dot{\mathbf{x}})\mathbf{L}(\mathbf{x})$ *for differentiable functions* $\mathbf{R}$*,* $\mathbf{L}$*, and* $\mathbf{D}$ *satisfying*

$$\mathbf{R}(\mathbf{x}) \succeq 0, \qquad \mathbf{D}(\mathbf{x},\dot{\mathbf{x}}) = \operatorname{diag}((d_i(\mathbf{x},\dot{y}_i))_{i=1}^n) \succeq 0, \qquad \dot{y}_i \partial_{\dot{y}_i} d_i(\mathbf{x},\dot{y}_i) \geq 0$$

where $\mathbf{x}$ *is the coordinate of the leaf-node manifold and* $\dot{\mathbf{y}} = \mathbf{L}\dot{\mathbf{x}} \in \mathbb{R}^n$*. It holds* $\boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{q},\dot{\mathbf{q}}) \succeq 0$*. If further* $\mathbf{G}(\mathbf{q},\dot{\mathbf{q}}), \mathbf{B}(\mathbf{q},\dot{\mathbf{q}}) \succ 0$*, then* $\mathbf{M} \in \mathbb{R}^{d\times d}_{++}$*, and the global RMP generated by RMPflow converges to the forward invariant set* $\mathcal{C}_\infty$ *in Corollary 2.*

Proof. Let $\mathbf{A}(\mathbf{x},\dot{\mathbf{x}}) = \mathbf{R}(\mathbf{x}) + \mathbf{L}(\mathbf{x})^\top \mathbf{D}(\mathbf{x},\dot{\mathbf{x}})\mathbf{L}(\mathbf{x})$. The proof of the theorem is straightforward, if we show that $\boldsymbol{\Xi}_{\mathbf{A}}(\mathbf{x},\dot{\mathbf{x}}) \succeq 0$. To see this, suppose $\mathbf{L} = \mathbb{R}^{n\times m}$. Let $\boldsymbol{\omega}_j^\top$ be the jth row $\mathbf{L}$, respectively. By definition of $\boldsymbol{\Xi}_{\mathbf{A}}(\mathbf{x},\dot{\mathbf{x}})$ we can write

$$\begin{aligned}
\boldsymbol{\Xi}_{\mathbf{A}}(\mathbf{x},\dot{\mathbf{x}}) &= \frac{1}{2}\sum_{i=1}^m \dot{x}_i \partial_{\dot{\mathbf{x}}} \mathbf{a}_i(\mathbf{x},\dot{\mathbf{x}}) \\
&= \frac{1}{2}\mathbf{L}(\mathbf{x})^\top \sum_{i=1}^m \dot{x}_i \partial_{\dot{\mathbf{x}}} (\mathbf{D}(\mathbf{x},\dot{\mathbf{x}})\mathbf{l}_i(\mathbf{x})) \\
&= \frac{1}{2}\mathbf{L}(\mathbf{x})^\top \sum_{i=1}^m \sum_{j=1}^n \dot{x}_i \partial_{\dot{\mathbf{x}}} (d_j(\mathbf{x},\dot{y}_j) L_{ji}(\mathbf{x})\mathbf{e}_j) \\
&= \frac{1}{2}\mathbf{L}(\mathbf{x})^\top \sum_{j=1}^n \left(\sum_{i=1}^m L_{ji}(\mathbf{x})\dot{x}_i\right) \partial_{\dot{y}_j} d_j(\mathbf{x},\dot{y}_j)\mathbf{e}_j\boldsymbol{\omega}_j^\top \\
&= \frac{1}{2}\mathbf{L}(\mathbf{x})^\top \sum_{j=1}^n \dot{y}_j \partial_{\dot{y}_j} d_j(\mathbf{x},\dot{y}_j)\mathbf{e}_j\boldsymbol{\omega}_j^\top \\
&= \mathbf{L}(\mathbf{x})^\top \boldsymbol{\Xi}_{\mathbf{D}}(\mathbf{x},\dot{\mathbf{x}})\mathbf{L}(\mathbf{x})
\end{aligned}$$

where $\mathbf{e}_j$ the jth canonical basis and $\boldsymbol{\Xi}_{\mathbf{D}}(\mathbf{x}, \dot{\mathbf{x}}) = \frac{1}{2}\mathrm{diag}((\partial_{\dot{y}_i} d_i(\mathbf{x}, \dot{y}_i))_{i=1}^n)$. Therefore, under the assumption that $\partial_{\dot{y}_i} d_i(\mathbf{x}, \dot{y}_i) \geq 0$, $\boldsymbol{\Xi}_{\mathbf{A}}(\mathbf{x}, \dot{\mathbf{x}}) \succeq 0$. This further implies $\boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}}) \succeq 0$ by Theorem 1.

The stability of the entire system follows naturally from the rule of `pullback`, which ensures that $\mathbf{M}_r(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{M}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{G}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{q}, \dot{\mathbf{q}}) \succ 0$ given that the leaf-node condition is satisfied. Consequently, the condition in Corollary 2 holds and the convergence to $\mathcal{C}_\infty$ is guaranteed. ■

APPENDIX D

RMPFLOW AND RECURSIVE NEWTON-EULER

The policy generation procedure of RMPflow is closely related to the algorithms [140] for computing forward dynamics (i.e. computing accelerations given forces) based on recursive Newton-Euler algorithm. In a summary, these algorithms computes the forward dynamics in following steps:

1. It propagates positions and velocities from the base to the end-effector.
2. It computes the Coriollis force by backward propagating the inverse dynamics of each link under the condition that the acceleration is zero.
3. It computes the (full/upper-triangular/lower-triangular) joint inertia matrix.
4. It solves a linear system of equations to obtain the joint acceleration.

In [140], they assume a recursive Newton-Euler algorithm (RNE) for inverse dynamics is given, and realize Step 1 and Step 2 above by calling the RNE subroutine. The computation of Step 3 depends on which part of the inertia matrix is computed. In particular, their Method 3 (also called the Composite-Rigid-Body Algorithm in [157, Chapter 6]) computes the upper triangle part of the inertia matrix by a backward propagation from the end-effector to the base.

RMPflow can also be used to compute forward dynamics, when we set the leaf-node GDS as the constant inertia system on the body frame of each link and we set the transformation in the RMP-tree as the change of coordinates across of robot links. This works because we show GDSs cover SMSs as a special case, and at root node the effective dynamics is the pullback GDS, which in this case is the effective robot dynamics defined by the inertia matrix of each link.

We can use this special case to compare RMPflow with the above procedure. We see that the forward pass of RMPflow is equivalent to Step 1, and the backward pass of RMPflow is equivalent of Step 2 and Step 3, and the final `resolve` operation is equivalent to Step 4.

Despite similarity, the main difference is that RMPflow computes the force and the inertia matrix in a *single* backward pass to exploit shared computations. This change is important, especially, the number of subtasks are large, e.g., in avoiding multiples obstacles. In addition, the design of RMPflow generalizes these classical computational procedures (e.g. designed only for rigid bodies, rotational/prismatic joints) to handle abstract and even non-Euclidean task spaces that have velocity-dependent metrics/inertias. This extension provides a unified framework of different algorithms and results in an expressive class of motion policies.

APPENDIX E

PROOFS OF RMPFUSION ANALYSIS

Here we provide the proof of Theorem 3. We use Theorem 1 as the main lemma in this proof. Using the recursive property, it is sufficient to show that `pullback*` preserves a family of structured GDSs, which are specified by the weight functions. Then the statement of Theorem 3 follows directly as in Appendix C.

We proceed by first decoupling the `pullback*` into two steps. Let u be a parent node on manifold $\mathcal{M}$ and $\{v_k\}_{k=1}^K$ be its K child nodes on manifold $\{\mathcal{N}_k\}_{k=1}^K$ in an RMP-tree*. Between u and each v_k, we add an extra node $\tilde{v}_k$ on manifold $\mathcal{M}$ to create a new graph. In this new graph, u has K child nodes $\{\tilde{v}_k\}_{k=1}^K$ *with identity transformation and the original weight function* w_k, and $\tilde{v}_k$ has a single child which is v_k *with the original transformation from* u *to* v_k *and an identity weight function*. Under this construction, the `pullback*` operator in the original graph can then be realized in the new graph as (i) a `pullback*` operator from v_k to $\tilde{v}_k$ for each k; (ii) a `pullback*` operator from $\{\tilde{v}_k\}_{k=1}^K$ to u. To verify this we rewrite Eq. (6.1) as

$$\begin{aligned}
\mathbf{f} &= \textstyle\sum_{i=1}^K w_i \mathbf{J}_i^\top (\mathbf{f}_i - \mathbf{M}_i \dot{\mathbf{J}}_i \dot{\mathbf{x}}) + \mathbf{h}_i =: \sum_{i=1}^K w_i \tilde{\mathbf{f}}_i + \tilde{\mathbf{h}}_i \\
\mathbf{M} &= \textstyle\sum_{i=1}^K w_i \mathbf{J}_i^\top \mathbf{M}_i \mathbf{J}_i =: \sum_{i=1}^K w_i \tilde{\mathbf{M}}_i \\
\mathbf{G} &= \textstyle\sum_{i=1}^K w_i \mathbf{J}_i^\top \mathbf{G}_i \mathbf{J}_i =: \sum_{i=1}^K w_i \tilde{\mathbf{G}}_i \\
L &= \textstyle\sum_{i=1}^K w_i L_i =: \sum_{i=1}^K w_i \tilde{L}_i
\end{aligned}$$

where we also has $\mathbf{h}_i = \tilde{\mathbf{h}}_i = \tilde{L}_i \nabla_{\mathbf{x}} w_i - (\dot{\mathbf{x}}^\top \nabla_{\mathbf{x}} w_i) \tilde{\mathbf{G}}_i \dot{\mathbf{x}}$. That is, node $\tilde{v}_k$ has the RMP $(\tilde{\mathbf{f}}_i, \tilde{\mathbf{M}}_i)_C^{\mathcal{M}}$, the metric matrix $\tilde{\mathbf{G}}_i$, and the Lagrangian $\tilde{L}_i$. From the equalities above, we verify the two-step decomposition of `pullback*` is valid.

Next we show that each step in the two-step decomposition yields a structured GDS like

Lemma 1, which is sufficient condition we need to prove Theorem 3. In the first step from v_i to $\tilde{v}_i$, because the weight is constant identity, `pullback*` is the same as `pullback`. We apply Lemma 1 and conclude that $\tilde{v}_i$ follows $(\mathcal{M}, \tilde{\mathbf{G}}_i, \tilde{\mathbf{B}}_i, \tilde{\Phi}_i)_{\tilde{\mathcal{S}}_i}$, where $\tilde{\mathcal{S}}_i$ preserves $\mathcal{S}_i$.

Then we show the second step from $\{\tilde{v}_i\}_{i=1}^K$ to u has similar properties. This is summarized as Lemma 6 below.

Lemma 6. *If $\tilde{v}_i$ follows $(\mathcal{M}, \tilde{\mathbf{G}}_i, \tilde{\mathbf{B}}_i, \tilde{\Phi}_i)_{\tilde{S}_i}$, then u follows $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)_S$, where S preserves $\tilde{S}_i$, $\mathbf{G} = \sum_{i=1}^K w_i \tilde{\mathbf{G}}_i$, $\mathbf{B} = \sum_{i=1}^K w_i \tilde{\mathbf{B}}_i$, and $\Phi = \sum_{i=1}^K w_i \tilde{\Phi}_i$.*

Proof of Lemma 6. This can be shown by algebraically comparing the dynamics of $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)_S$ and the result of Eq. (6.1). Let $\mathbf{x}$ be a coordinate of $\mathcal{M}$ and, without loss of generality, let us consider w_k to be a function of only $\mathbf{x}$ (we ignore the dependency on the auxiliary state). By Eq. (5.23), the dynamics of $(\mathcal{M}, \mathbf{G}, \mathbf{B}, \Phi)_S$ satisfies

$$\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}})\ddot{\mathbf{x}} + \boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{x}, \dot{\mathbf{x}}) = -\nabla_{\mathbf{x}}\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \tag{E.1}$$

We first show the recursion of $\mathbf{f}$ of `pullback*` satisfies Eq. (E.1). To this end, we rewrite $\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}$ by Eq. (5.23) as

$$\begin{aligned}
\boldsymbol{\eta}_{\mathbf{G};\mathcal{S}}(\mathbf{x}, \dot{\mathbf{x}}) &= \textstyle\sum_{i=1}^K \boldsymbol{\xi}_{w_i \tilde{\mathbf{G}}_i}(\mathbf{x}, \dot{\mathbf{x}}) \\
&= \textstyle\sum_{i=1}^K w_i(\mathbf{x}) \boldsymbol{\eta}_{\tilde{\mathbf{G}}_i}(\mathbf{x}, \dot{\mathbf{x}}) + (\dot{\mathbf{x}}^\top \nabla_{\mathbf{x}} w_i(\mathbf{x}))\tilde{\mathbf{G}}_i(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}} \\
&\quad - \tfrac{1}{2}\nabla_{\mathbf{x}} w_i(\mathbf{x})\dot{\mathbf{x}}^\top \tilde{\mathbf{G}}_i(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}
\end{aligned}$$

where in the first equality we use the trick we made that the transformation from u to $\tilde{v}_k$ is identity and we use the fact $\tilde{S}_i$ preserves S_i, so the structure S that preserves $\tilde{S}_i$ has a clean

structure

$$\mathbf{G} = \begin{bmatrix} I & \dots & I \end{bmatrix} \begin{bmatrix} w_1\tilde{\mathbf{G}}_1 & & \\ & \ddots & \\ & & w_K\tilde{\mathbf{G}}_K \end{bmatrix} \begin{bmatrix} I \\ \vdots \\ I \end{bmatrix}$$

Similarly, we rewrite $\nabla_{\mathbf{x}}\Phi(\mathbf{x}) = \sum_{i=1}^{K} w_i(\mathbf{x})\nabla_{\mathbf{x}}\tilde{\Phi}_i(\mathbf{x}) + \tilde{\Phi}_i\nabla_{\mathbf{x}}w_i(\mathbf{x})$. Substituting these two equalities into Eq. (E.1), we can write (with input dependency omitted)

$$\begin{aligned}
\mathbf{M}\ddot{\mathbf{x}} &= -\nabla_{\mathbf{x}}\Phi - \mathbf{B}\dot{\mathbf{x}} - \boldsymbol{\eta}_{\mathbf{G};\mathcal{S}} \\
&= \textstyle\sum_{i=1}^{K} -w_i\nabla_{\mathbf{x}}\tilde{\Phi}_i - \tilde{\Phi}_i\nabla_{\mathbf{x}}w_i - w_i\mathbf{B}_i\dot{\mathbf{x}} \\
&\quad + \textstyle\sum_{i=1}^{K} -w_i\boldsymbol{\eta}_{\tilde{\mathbf{G}}_i} - (\dot{\mathbf{x}}^\top\nabla_{\mathbf{x}}w_i)\tilde{\mathbf{G}}_i\dot{\mathbf{x}} + \tfrac{1}{2}\nabla_{\mathbf{x}}w_i\dot{\mathbf{x}}^\top\tilde{\mathbf{G}}_i\dot{\mathbf{x}} \\
&= \textstyle\sum_{i=1}^{K} w_i\tilde{\mathbf{f}}_i + \tfrac{1}{2}\nabla_{\mathbf{x}}w_i\dot{\mathbf{x}}^\top\tilde{\mathbf{G}}_i\dot{\mathbf{x}} - \tilde{\Phi}_i\nabla_{\mathbf{x}}w_i - (\dot{\mathbf{x}}^\top\nabla_{\mathbf{x}}w_i)\tilde{\mathbf{G}}_i\dot{\mathbf{x}} \\
&= \textstyle\sum_{i=1}^{K} w_i\tilde{\mathbf{f}}_i + \mathbf{h}_i
\end{aligned}$$

where we use the fact that $\tilde{\mathbf{f}}_i = -\nabla_{\mathbf{x}}\tilde{\Phi}_i - \tilde{\mathbf{B}}_i\dot{\mathbf{x}} - \boldsymbol{\eta}_{\tilde{\mathbf{G}}_i;\tilde{\mathcal{S}}_i}$ as $\tilde{v}_i$ follows $(\mathcal{M}, \tilde{\mathbf{G}}_i, \tilde{\mathbf{B}}_i, \tilde{\Phi}_i)_{\tilde{\mathcal{S}}_i}$ with $\tilde{\mathcal{S}}_i$ preserving $\mathcal{S}_i$. This is exactly the recursion of $\mathbf{f}$ when `pullback`* is applied between $\tilde{v}_i$ and u, i.e. $\mathbf{f} = \mathbf{M}\ddot{\mathbf{x}} = \sum_{i=1}^{K} w_i\tilde{\mathbf{f}}_i + \mathbf{h}_i$.

To establish the equivalence of the other recursions, we next rewrite $\mathbf{M}$ as

$$\begin{aligned}
\mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) &= \mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) + \boldsymbol{\Xi}_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) \\
&= \textstyle\sum_{i=1}^{K} w_i(\mathbf{x}) \left(\tilde{\mathbf{G}}_i(\mathbf{x}, \dot{\mathbf{x}}) + \boldsymbol{\Xi}_{\tilde{\mathbf{G}}_i}(\mathbf{x}, \dot{\mathbf{x}}) \right) \\
&= \textstyle\sum_{i=1}^{K} w_i(\mathbf{x})\tilde{\mathbf{M}}_i(\mathbf{x}, \dot{\mathbf{x}})
\end{aligned}$$

where we use the fact that w_i does not on the velocity $\dot{\mathbf{x}}$. The recursion for $\mathbf{G}$ and L can be derived similarly, so we omit them here. ■

So far we have shown that `pullback`* of RMPfusion retains the closure of structured

GDSs as `pullback` in RMPflow. In addition, we show that the structured GDS created by `pullback`* has a linearly weighted metric matrix, damping matrix, and potential function (cf. Lemma 6). By recursively applying the two-step decomposition above, from the leaf nodes to the root node, we conclude that the root node policy will be a structured GDS with an energy given by the recursion in Eq. (6.2). The rest of the statement of Theorem 3 follows from the properties of structured GDSs.

REFERENCES

[1] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *ICRA*, vol. 94, 1994, pp. 3310–3317.

[2] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm.," in *ICAPS*, 2005, pp. 262–271.

[3] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[5] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, IEEE, vol. 2, 2000, pp. 995–1001.

[6] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[7] D. H. Jacobson and D. Q. Mayne, "Differential dynamic programming," 1970.

[8] D. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd. Athena Scientific, Belmont, MA, 2000.

[9] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, IEEE, 2009, pp. 489–494.

[10] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *In proceedings of the American Control Conference*, vol. 1, 2005, pp. 300–306.

[11] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization.," in *Robotics: Science and Systems*, Citeseer, vol. 9, 2013, pp. 1–10.

[12] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[13] H. Attias, "Planning by probabilistic inference," in *AISTATS*, 2003.

[14] M. Toussaint and A. Storkey, "Probabilistic inference for solving discrete and continuous state Markov decision processes," in *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, pp. 945–952.

[15] M. Toussaint, "Robot trajectory optimization using approximate inference," in *Proceedings of the 26th annual international conference on machine learning*, ACM, 2009, pp. 1049–1056.

[16] H. J. Kappen, V. Gómez, and M. Opper, "Optimal control as a graphical model inference problem," *Machine learning*, vol. 87, no. 2, pp. 159–182, 2012.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[19] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *arXiv preprint arXiv:1511.03791*, 2015.

[20] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 3357–3364.

[21] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[22] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[23] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 512–519.

[24] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[25] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 23–30.

[26] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Conference on Robot Learning*, 2017, pp. 262–270.

[27] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[28] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 3981–3989.

[29] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 136–145.

[30] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems*, 2018, pp. 6572–6583.

[31] A. Byravan and D. Fox, "Se3-nets: Learning rigid body motion using deep neural networks," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 173–180.

[32] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," *arXiv preprint arXiv:1811.08027*, 2018.

[33] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects," *arXiv preprint arXiv:1710.04102*, 2017.

[34] Z. Lv, F. Dellaert, J. M. Rehg, and A. Geiger, "Taking a deeper look at the inverse compositional algorithm," *arXiv preprint arXiv:1812.06861*, 2018.

[35] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.

[36] M. Mukadam, X. Yan, and B. Boots, "Gaussian process motion planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 9–15.

[37] M. Toussaint and C. Goerick, "A Bayesian view on motor control and planning," in *From Motor Learning to Interaction Learning in Robots*, Springer, 2010, pp. 227–252.

[38] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[39] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[40] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using Gaussian processes and factor graphs," in *Proceedings of Robotics: Science and Systems (RSS-2016)*, 2016.

[41] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time Gaussian process motion planning via probabilistic inference," *The International Journal of Robotics Research (IJRR)*, vol. 37, no. 11, pp. 1319–1340, 2018.

[42] M. Kaess, A. Ranganathan, and F. Dellaert, "ISAM: Incremental smoothing and mapping," *Robotics, IEEE Transactions on*, vol. 24, no. 6, pp. 1365–1378, 2008.

[43] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "ISAM2: Incremental smoothing and mapping using the Bayes tree," *The International Journal of Robotics Research*, p. 0 278 364 911 430 419, 2011.

[44] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[45] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[46] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *Algorithmic Foundation of Robotics VIII*, Springer, 2009, pp. 449–464.

[47] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[48] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, 2010.

[49] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 3067–3074.

[50] K. He, E. Martin, and M. Zucker, "Multigrid CHOMP with local smoothing," in *Proc. of 13th IEEE-RAS Int. Conference on Humanoid Robots (Humanoids)*, 2013.

[51] A. Byravan, B. Boots, S. S. Srinivasa, and D. Fox, "Space-time functional gradient optimization for motion planning," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 6499–6506.

[52] Z. Marinho, A. Dragan, A. Byravan, B. Boots, G. J. Gordon, and S. Srinivasa, "Functional gradient motion planning in reproducing kernel Hilbert spaces," in *Proceedings of Robotics: Science and Systems (RSS-2016)*, 2016.

[53] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 4569–4574.

[54] M. Bosse and R. Zlot, "Continuous 3D scan-matching with a spinning 2D laser," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, IEEE, 2009, pp. 4312–4319.

[55] M. Li, B. H. Kim, and A. I. Mourikis, "Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, IEEE, 2013.

[56] H. Dong and T. D. Barfoot, "Lighting-invariant visual odometry using lidar intensity imagery and pose interpolation," in *Field and Service Robotics (FSR)*, Springer, 2014, pp. 327–342.

[57] C. Bibby and I. Reid, "A hybrid SLAM representation for dynamic marine environments," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, IEEE, 2010, pp. 257–264.

[58] S. Anderson and T. D. Barfoot, "Towards relative continuous-time SLAM," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, IEEE, 2013.

[59] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, 2013.

[60] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *Intl. J. of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[61] A. Patron-Perez, S. Lovegrove, and G. Sibley, "A spline-based trajectory representation for sensor fusion and rolling shutter cameras," *International Journal of Computer Vision*, vol. 113, no. 3, pp. 208–219, 2015.

[62] P. Furgale, C. H. Tong, T. D. Barfoot, and G. Sibley, "Continuous-time batch trajectory estimation using temporal basis functions," *Intl. J. of Robotics Research*, vol. 34, no. 14, pp. 1688–1710, 2015.

[63] S. Anderson, F. Dellaert, and T. Barfoot, "A hierarchical wavelet decomposition for continuous-time SLAM," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014.

[64] M. Elbanhawi, M. Simic, and R. Jazar, "Randomized bidirectional B-Spline parameterization motion planning," *Intelligent Transportation Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[65] S. Vijayakumar, A. D'souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[66] K. Kersting, C. Plagemann, P. Pfaff, and W. Burgard, "Most likely heteroscedastic Gaussian process regression," in *Proceedings of the 24th international conference on Machine learning*, ACM, 2007, pp. 393–400.

[67] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schölkopf, "Learning inverse dynamics: A comparison," in *European Symposium on Artificial Neural Networks*, 2008.

[68] J. Sturm, C. Plagemann, and W. Burgard, "Body schema learning for robotic manipulators from visual self-perception," *Journal of Physiology-Paris*, vol. 103, no. 3, pp. 220–231, 2009.

[69] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[70] M. K. C. Tay and C. Laugier, "Modelling smooth paths using Gaussian processes," in *Field and Service Robotics*, Springer, 2008, pp. 381–390.

[71] T. Barfoot, C. H. Tong, and S. Sarkka, "Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression," *Proceedings of Robotics: Science and Systems, Berkeley, USA*, 2014.

[72] X. Yan, V. Indelman, and B. Boots, "Incremental sparse GP regression for continuous-time trajectory estimation and mapping," in *Robotics and Autonomous Systems*, vol. 87, 2017, pp. 120–132.

[73] J. Ko and D. Fox, "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models," *Autonomous Robots*, vol. 27, no. 1, pp. 75–90, 2009.

[74] C. H. Tong, P. Furgale, and T. D. Barfoot, "Gaussian process Gauss-Newton for non-parametric simultaneous localization and mapping," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 507–525, 2013.

[75] E. Theodorou, Y. Tassa, and E. Todorov, "Stochastic differential dynamic programming," in *American Control Conference (ACC), 2010*, IEEE, 2010, pp. 1125–1132.

[76] S. Levine and V. Koltun, "Variational policy search via trajectory optimization," in *Advances in Neural Information Processing Systems*, 2013, pp. 207–215.

[77] K. Rawlik, M. Toussaint, and S. Vijayakumar, "On stochastic optimal control and reinforcement learning by approximate inference," *Proceedings of Robotics: Science and Systems VIII*, 2012.

[78] S. Koenig, C. Tovey, and Y. Smirnov, "Performance bounds for planning in unknown terrain," *Artificial Intelligence*, vol. 147, no. 1, pp. 253–279, 2003.

[79] C. Park, J. Pan, and D. Manocha, "ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments.," in *ICAPS*, 2012.

[80] ——, "Real-time optimization-based planning in dynamic environments using GPUs," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, IEEE, 2013, pp. 4090–4097.

[81] S. Sarkka, A. Solin, and J. Hartikainen, "Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering," *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 51–61, 2013.

[82] S. Anderson, T. D. Barfoot, C. H. Tong, and S. Särkkä, "Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression," *Autonomous Robots*, vol. 39, no. 3, pp. 221–238, 2015.

[83] R. Courant and D. Hilbert, *Methods of mathematical physics*. CUP Archive, 1966, vol. 1.

[84] S. Quinlan, "Real-time modification of collision-free paths," PhD thesis, Stanford University, 1994.

[85] M. Mukadam, C. A. Cheng, X. Yan, and B. Boots, "Approximately optimal continuous-time motion planning and control via probabilistic inference.," in *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[86] M. Mukadam, J. Dong, F. Dellaert, and B. Boots, "Simultaneous trajectory estimation and planning via probabilistic inference.," in *Proceedings of Robotics: Science and Systems (RSS)*, 2017.

[87] ——, "STEAP: Simultaneous trajectory estimation and planning," in *Autonomous Robots (AURO)*, vol. 43, 2018, pp. 415–434.

[88] M. A. Rana, M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots., "Towards robust skill generalization: Unifying learning from demonstration and motion planning," in *Proceedings of the 2017 Conference on Robot Learning (CoRL)*, 2017.

[89] M. A. Rana, M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots, "Learning generalizable robot skills from demonstrations in cluttered environments," in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[90] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[91] M. Kaess, V. Ila, R. Roberts, and F. Dellaert, "The Bayes tree: An algorithmic foundation for probabilistic robot mapping," in *Algorithmic Foundations of Robotics IX*, Springer, 2011, pp. 157–173.

[92] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Tech Technical Report, GT-RIM-CP&R-2012-002, Tech. Rep., 2012.

[93] E. Huang, M. Mukadam, Z. Liu, and B. Boots, "Motion planning with graph-based trajectories and Gaussian process inference.," in *Proceedings of the 2017 IEEE Conference on Robotics and Automation (ICRA)*, 2017.

[94] J. Dong, M. Mukadam, B. Boots, and F. Dellaert, "Sparse gaussian processes on matrix lie groups: A unified framework for optimizing continuous-time trajectories.," in *Proceedings of the 2018 IEEE Conference on Robotics and Automation (ICRA)*, 2018.

[95] A. Lambert, M. Mukadam, B. Sundaralingam, N. Ratliff, B. Boots, and D. Fox., "Joint inference of kinematic and force trajectories with visuo-tactile sensing," in *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, 2019.

[96] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[97] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in neural information processing systems*, 2003, pp. 1547–1554.

[98] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, 2007.

[99] B. Reiner, W. Ertel, H. Posenauer, and M. Schneider, "Lat: A simple learning from demonstration method," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2014, pp. 4436–4441.

[100] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.

[101] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with Gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[102] S. Calinon, I. Sardellitti, and D. G. Caldwell, "Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2010, pp. 249–254.

[103] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

[104] D. B. Grimes, R. Chalodhorn, and R. P. Rao, "Dynamic imitation in a humanoid robot through nonparametric probabilistic inference," in *Robotics: Science and Systems (RSS)*, 2006, pp. 199–206.

[105] M. Schneider and W. Ertel, "Robot learning by demonstration with local Gaussian process regression," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2010, pp. 255–260.

[106] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," in *Advances in neural information processing systems*, 2013, pp. 2616–2624.

[107] A. D. Dragan, K. Muelling, J. A. Bagnell, and S. S. Srinivasa, "Movement primitives via optimization," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 2339–2346.

[108] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *2009 IEEE International Conference on Robotics and Automation(ICRA)*, IEEE, 2009, pp. 763–768.

[109] G. Ye and R. Alterovitz, "Demonstration-guided motion planning," in *International symposium on robotics research (ISRR)*, vol. 5, 2011.

[110] T. Osa, A. M. G. Esfahani, R. Stolkin, R. Lioutikov, J. Peters, and G. Neumann, "Guiding trajectory optimization by demonstrated distributions," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 819–826, 2017.

[111] D. Koert, G. Maeda, R. Lioutikov, G. Neumann, and J. Peters, "Demonstration based trajectory optimization for generalizable robot motions," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2016, pp. 515–522.

[112] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[113] M. K. Titsias, "Variational learning of inducing variables in sparse Gaussian processes.," in *AISTATS*, vol. 5, 2009, pp. 567–574.

[114] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[115] M. Bhardwaj, S. Choudhury, and S. Scherer, "Learning heuristic search via imitation," in *CoRL*, 2017.

[116] A. Faust, K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson, "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 5113–5120.

[117] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.

[118] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks: Learning generalizable representations for visuomotor control," in *International Conference on Machine Learning*, 2018, pp. 4739–4748.

[119] R. Clark, M. Bloesch, J. Czarnowski, S. Leutenegger, and A. J. Davison, "Learning to solve nonlinear least squares for monocular stereo," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 284–299.

[120] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable mpc for end-to-end planning and control," in *Advances in Neural Information Processing Systems*, 2018, pp. 8299–8310.

[121] M. Bhardwaj, B. Boots, and M. Mukadam, "Differentiable Gaussian process motion planning," *arXiv preprint arXiv:1907.09591*, 2019.

[122] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[123] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[124] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[125] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural computation*, vol. 2, no. 4, pp. 490–501, 1990.

[126] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[127] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[128] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[129] E. Rimon and D. Koditschek, "The construction of analytic diffeomorphisms for exact robot navigation on star worlds," *Transactions of the American Mathematical Society'*, vol. 327, no. 1, pp. 71–116, 1991.

[130] N. Ratliff, M. Toussaint, and S. Schaal, "Understanding the geometry of workspace obstacles in motion optimization," in *IEEE ICRA*, 2015.

[131] V. Ivan, D. Zarubin, M. Toussaint, T. Komura, and S. Vijayakumar, "Topology-based representations for motion planning and generalization in dynamic environments with interactions," *IJRR*, vol. 32, no. 9-10, pp. 1151–1163, 2013.

[132] M. Watterson, S. Liu, K. Sun, T. Smith, and V. Kumar, "Trajectory optimization on manifolds with applications to SO(3) and R3XS2," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, 2018.

[133] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[134] J. Peters, M. Mistry, F. Udwadia, J. Nakanishi, and S. Schaal, "A unifying framework for robot control with redundant dofs," *Autonomous Robots*, vol. 24, no. 1, pp. 1–12, 2008.

[135] F. E. Udwadia, "A new perspective on the tracking control of nonlinear structural and mechanical systems," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 459, no. 2035, pp. 1783–1800, 2003. eprint: `http://rspa.royalsocietypublishing.org/content/459/2035/1783.full.pdf`.

[136] D. Kappler, F. Meier, J. Issac, J. Mainprice, C. Garcia Cifuentes, M. Wüthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg, "Real-time perception meets reactive motion generation," *https://arxiv.org/abs/1703.03512*, 2017.

[137] F. Bullo and A. D. Lewis, *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*. Springer Science & Business Media, 2004, vol. 49.

[138] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "RMPflow: A computational graph for automatic motion policy generation," in *The 13th International Workshop on the Algorithmic Foundations of Robotics*, 2018.

[139] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," *arXiv preprint arXiv:1801.02854*, 2018.

[140] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *Journal of Dynamic Systems, Measurement, and Control*, vol. 104, no. 3, pp. 205–211, 1982.

[141] A. Albu-Schaffer and G. Hirzinger, "Cartesian impedance control techniques for torque controlled light-weight robots," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, IEEE, vol. 1, 2002, pp. 657–663.

[142] L. Sentis and O. Khatib, "A whole-body control framework for humanoids operating in human environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, IEEE, 2006, pp. 2641–2648.

[143] S.-Y. Lo, C.-A. Cheng, and H.-P. Huang, "Virtual impedance control for safe human-robot interaction," *Journal of Intelligent & Robotic Systems*, vol. 82, no. 1, pp. 3–19, 2016.

[144] H. K. Khalil, "Noninear systems," *Prentice-Hall, New Jersey*, vol. 2, no. 5, pp. 5–1, 1996.

[145] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model-predictive control of humanoid robots," in *IEEE/RAS International Conference on Humanoid Robots*, 2013.

[146] E Todorov, "Optimal control theory," *In Bayesian Brain: Probabilistic Approaches to Neural Coding*, D. K. et al, Ed., pp. 269–298, 2006.

[147] A. Liegeois, "Automatic supervisory control of the configuration and behaviour of multibody mechanisms," *IEEE Transactions on systems, man and cybernetics*, vol. 7, no. 12, pp. 868–871, 1977.

[148] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: A theoretical and empirical comparison," *IJRR*, vol. 6, pp. 737–757, 2008.

[149] R. Platt, M. E. Abdallah, and C. W. Wampler, "Multiple-priority impedance control.," in *ICRA*, Citeseer, 2011, pp. 6033–6038.

[150] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[151] F. E. Udwadia and R. E. Kalaba, *Analytical Dynamics: A New Approach*. Cambridge University Press, 1996.

[152] J. Mainprice, N. Ratliff, and S. Schaal, "Warping the workspace geometry with electric potentials for motion optimization of manipulation tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[153] J. R. Taylor, *Classical Mechanics*. University Science Books, 2005.

[154] J. Nash, "The imbedding problem for Riemannian manifolds," *Ann. Math.*, vol. 63, pp. 20–63, 1956.

[155] T. Schmidt, R. Newcombe, and D. Fox, "DART: Dense articulated real-time tracking with consumer depth cameras," *Autonomous Robots*, vol. 39, no. 3, pp. 239–258, 2015.

[156] A. Li, M. Mukadam, M. Egerstedt, and B. Boots, "Multi-objective policy generation for multi-robot systems using riemannian motion policies," *arXiv preprint arXiv:1902.05177*, 2019.

[157] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer, 2008.

[158] S. B. Slotine, "A general framework for managing multiple tasks in highly redundant robotic systems," in *proceeding of 5th International Conference on Advanced Robotics*, vol. 2, 1991, pp. 1211–1216.

[159] R. C. Arkin, "Governing lethal behavior: Embedding ethics in a hybrid deliberative/reactive robot architecture," in *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*, ACM, 2008, pp. 121–128.

[160] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning.," in *OSDI*, vol. 16, 2016, pp. 265–283.

[161] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[162] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[163] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *International conference on artificial intelligence and statistics*, 2011, pp. 627–635.

[164] S. Ross and J. A. Bagnell, "Reinforcement and imitation learning via interactive no-regret learning," *arXiv preprint arXiv:1406.5979*, 2014.

[165] C.-A. Cheng, X. Yan, N. Wagener, and B. Boots, "Fast policy learning through imitation and reinforcement," *arXiv preprint arXiv:1805.10413*, 2018.

[166] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[167] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[168] H. Ravichandar, S. R. Ahmadzadeh, M. A. Rana, and S. Chernova, "Skill acquisition via automated multi-coordinate cost balancing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[169] K. Neumann and J. J. Steil, "Learning robot motions with stable dynamical systems under diffeomorphic transformations," *Robotics and Autonomous Systems*, vol. 70, pp. 1–15, 2015.

[170] H. C. Ravichandar and A. Dani, "Learning position and orientation dynamics from demonstrations via contraction analysis," *Autonomous Robots*, pp. 1–16, 2018.

[171] S. Calinon and A. Billard, "Statistical learning by imitation of competing constraints in joint space and task space," *Advanced Robotics*, vol. 23, no. 15, pp. 2059–2076, 2009.

[172] A. Paraschos, R. Lioutikov, J. Peters, and G. Neumann, "Probabilistic prioritization of movement primitives," 2017.

[173] J. Silvério, S. Calinon, L. Rozo, and D. G. Caldwell, "Learning task priorities from demonstrations," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 78–94, 2019.

[174] Y. Shavit, N. Figueroa, S. S. M. Salehian, and A. Billard, "Learning augmented joint-space task-oriented dynamical systems: A linear parameter varying and synergetic control approach," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2718–2725, 2018.

[175] S. M. Khansari-Zadeh and O. Khatib, "Learning potential functions from human demonstrations with encapsulated dynamic and compliant behaviors," *Autonomous Robots*, vol. 41, no. 1, pp. 45–69, 2017.

[176] M. Grant, S. Boyd, and Y. Ye, *Cvx: Matlab software for disciplined convex programming*, 2009.