Learning Reactive Motion Policies in Multiple Task Spaces from Human Demonstrations

M. Asif Rana*[1,3] **Anqi Li***[1,3] **Harish Ravichandar**[1] **Mustafa Mukadam**[1]
Sonia Chernova[1] **Dieter Fox**[2,3] **Byron Boots**[1,3] **Nathan Ratliff**[3]
[1]Georgia Institute of Technology [2]University of Washington [3]NVIDIA

Abstract: Complex manipulation tasks often require non-trivial and coordinated movements of different parts of a robot. In this work, we address the challenges associated with learning and reproducing the skills required to execute such complex tasks. Specifically, we decompose a task into multiple subtasks and learn to reproduce the subtasks by learning stable policies from demonstrations. By leveraging the RMPflow framework for motion generation, our approach finds a stable global policy in the configuration space that enables simultaneous execution of various learned subtasks. The resulting global policy is a weighted combination of the learned policies such that the motions are coordinated and feasible under the robot's kinematic and environmental constraints. We demonstrate the necessity and efficacy of the proposed approach in the context of multiple constrained manipulation tasks performed by a Franka Emika robot.

Keywords: Learning from demonstration

1 Introduction

Learning from demonstration (LfD) [1] provides a paradigm for introducing robots to skills required for executing complex tasks. Hand-specifying such skills can otherwise be hard or infeasible, especially for inexperienced end-users. A given task may require *specific*, *coordinated*, and *adaptive* movements of different parts of the robot. For example, consider a wiping task. In addition to requiring the end-effector to maintain contact, it may be desirable to maintain a specific elbow orientation in order to effectively wipe the surface. Movements of the elbow and the end-effector are also interdependent, requiring coordination. All the robot parts must also adapt to environmental changes, such as displacement of the surface to be wiped, or introduction of a new obstacle. Thus, each part of the robot can be viewed as executing multiple individual subtasks, dictating desired behaviors.

While previous work has detailed how to learn goal-directed reactive policies from demonstrations [2, 3, 4], these approaches cannot combine multiple policies, each dictating the behavior in a certain subtask space, while maintaining stability properties. A recently introduced motion generation approach, RMPflow [5], enables geometrically-consistent combination of multiple reactive motion policies [5, 6]. The policies employed in RMPflow however are hand-designed, potentially limiting its use to scenarios in which the individual policies can be easily specified and are known *a priori*. In this paper, we learn stable reactive subtask policies from human demonstrations and utilize RMPflow to combine them. As a result, we contribute a new LfD method capable of learning and reproducing desired behaviors simultaneously in multiple subtask spaces.

We view human demonstrations as motions in potentially non-Euclidean subtask spaces, governed by *Riemannian Motion Policies* (RMPs) [6]. An RMP is a mathematical object composed of an acceleration policy along with a Riemannian metric, which defines the underlying non-Euclidean geometry of the subtask space. Subtask spaces are often not independent and thus must be coordinated while satisfying constraints, such as those enforced by the robot kinematics. Toward this end, we utilize RMPflow [5], a method to combine policies associated with different subtask spaces. Specifically, for each subtask space, we independently learn a stable RMP. RMPflow is then utilized to enforce correlations and constraints among the subtask policies as dictated by the kinematics of the robot. The combined policy from RMPflow preserves stability, while the learned subtask Rie-

* Indicates equal contribution. Email: asif.rana@gatech.edu, anqil4@cs.washington.edu

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

mannian metrics assist in policy resolution by acting as state-dependent weights associated with subtask acceleration policies.

In summary, we contribute an LfD approach that: (*i*) learns stable time-invariant policies from demonstrations defined in multiple subtask spaces; and (*ii*) combines the learned policies by explicitly taking into account the robot kinematics, while ensuring that the combined overall policy is Lyapunov stable. We demonstrate the effectiveness of the proposed approach on *door reaching* and *drawer closing* tasks and illustrate the necessity of learning policies in multiple subtasks spaces.

2 Related Work

Motion generation techniques for articulated robots can be grouped into motion planning [7, 8, 9] and reactive policy synthesis [5, 10]. While motion planning methods, particularly trajectory optimization-based methods [11, 12, 13], seek to minimize a cost functional over a given time-horizon, reactive policy methods generate actions based on the instantaneous state. Typically, the cost functional or the policy are hand-specified to generate smooth and collision-free motions. This is sometimes infeasible for complex manipulation tasks. Learning from demonstration seeks to overcome this challenge by learning either a cost functional [14, 15, 16, 17, 18], a time-dependent motion representation [19, 20, 21], or a time-invariant (reactive) policy [2, 3, 4, 21].

In practice, stable reactive methods are well suited for dynamic environments since they allow instantaneous adaptation to environmental changes while also providing convergence guarantees to a target. In this work, we focus on learning reactive motion policies from demonstrations. Existing methods learn reactive policies for the robot end-effector only, without considering the desired behaviors for other parts of the robot. Perhaps, a potential approach to simultaneously encoding the role of different parts of the robot could learn motions in the robot's joint space. Prior work has explored learning policies in either robot's joint space only or in conjunction with its workspace [19, 20, 22, 23]. These methods, however, provide time-dependent policies, and are thus prone to failure when execution timing drifts away from demonstrated timing. Furthermore, learning policies in joint-space can sometimes be over-constraining, especially for redundant manipulators, whereby the particular joint values are not as critical as the movement in the relevant subtask spaces. Lastly, reactive policy learning methods generally do not account for the robot's kinematic constraints and/or joint limits.

To the best of our knowledge, no existing work learns stable time-invariant policies from demonstrations capable of simultaneously reproducing the desired behaviors in multiple subtask spaces.

3 Background: Riemannian Motion Policies

Consider a robot with its configuration space $\mathcal{C}$ given by a smooth manifold, admitting a global generalized coordinate $\mathbf{q} \in \mathbb{R}^d$. Oftentimes, it is more convenient to describe the specifications of the robot motion on another manifold called the *task space*, denoted $\mathcal{T}$, where there exists a smooth task map $\psi : \mathcal{C} \to \mathcal{T}$ that maps the configuration space to the task space. The RMP framework [5, 6] assumes that the overall task can be decomposed into a set of *subtasks* defined on different *subtask spaces*. Examples of subtasks include goal reaching or trajectory following for the end-effector, collision avoidance for a certian part of the robot, etc. The task space $\mathcal{T}$ can thereby be represented as the collection of multiple *subtask* spaces. The goal of RMPs and RMPflow [6, 5] is to generate an acceleration policy $\mathbf{a_r} = \pi(\mathbf{q}, \dot{\mathbf{q}})$ on the configuration space $\mathcal{C}$ such that the transformed policies exhibit the desired behaviors in each of the subtask spaces.

3.1 Riemannian Motion Policies (RMPs)

A Riemannian Motion Policy (RMP) [5, 6] is a mathematical object describing motions on manifolds. Consider an m-dimensional manifold $\mathcal{M}$ with generalized coordinates $\mathbf{x} \in \mathbb{R}^m$. An RMP on the manifold $\mathcal{M}$ can be succinctly represented by its *canonical form* as a pair $(\mathbf{a}, \mathbf{M})^{\mathcal{M}}$. Here, the first component, $\mathbf{a} \in \mathbb{R}^m$ is an acceleration policy governing motions on the manifold, while the second component, $\mathbf{M} \in \mathbb{R}^{m \times m}_{+}$ is a Riemannian metric, that is a positive definite matrix defining the structure of the underlying manifold. An alternative parameterization of RMP which will become useful later is given by the *natural form* $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$ where $\mathbf{f} := \mathbf{M}\,\mathbf{a}$ is the desired *force map*.

One realization of an RMP is a *virtual* dynamical system [5],

$$\mathbf{a} = \mathbf{M}(\mathbf{x})^{-1}\big(-\nabla\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x})\,\dot{\mathbf{x}} - \boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x}, \dot{\mathbf{x}})\big), \tag{1}$$

where, as in Geometric Mechanics [24], the Riemannian metric $\mathbf{M}(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}_+^{m\times m}$ serves as the *inertia matrix*, $\mathbf{B}(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}_+^{m\times m}$ is the *damping matrix* and $\Phi(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}_+$ the *potential function*. The system in (1) dictates the motion on a manifold under the influence of a damped virtual potential field. An important property of this system is that it is inherently *Lyapunov stable*.

The Riemannian metric $\mathbf{M}$ also induces the curvature term $\boldsymbol{\xi}_{\mathbf{M}}$. The ith component of the curvature term, denoted $(\xi_{\mathbf{M}})_i$, is given by

$$(\xi_{\mathbf{M}})_i = \sum_{j=1}^{n}\sum_{k=1}^{n} \frac{1}{2}\left(\frac{\partial M_{ij}(\mathbf{x})}{\partial x_k} + \frac{\partial M_{ki}(\mathbf{x})}{\partial x_j} - \frac{\partial M_{jk}(\mathbf{x})}{\partial x_i}\right)\dot{x}_j\,\dot{x}_k, \tag{2}$$

where we use M_{ij} to denote an entry of a matrix $\mathbf{M}$ and x_i to denote a component of a vector $\mathbf{x}$. Intuitively, the curvature term $\boldsymbol{\xi}_{\mathbf{M}}$ bends the trajectories to follow geodesics on the manifold in the absence of the potential and damping terms. It should be noted that the Riemannian metric $\mathbf{M}(\mathbf{x})$ employed in this paper is only position dependent. In the case when the Riemannian metric is both position and velocity-dependent, a generalization of the system (1), called Geometric Dynamical System (GDS) [5], can be used instead.

For simple tasks like reaching a goal without any additional specifications, a system that generates the RMP can be hand-specified as is done in [6]. However, for more complex behaviors, it can be hard or sometimes infeasible for an end-user to design RMPs. To mitigate this problem, in Section 4 we detail a method which can instead learn RMPs of the form (1) from human demonstrations.

3.2 RMPflow

RMPflow [5] is a computational framework to generate reactive policies by combining RMPs. Given multiple individually specified (or learned in the subsequent sections) RMPs for different subtasks, RMPflow combines these policies into one global configuration space policy.

The core data structure of RMPflow is the RMP-tree, a directed tree encoding the structure of the task map. Specifically, each node $\mathtt{v}$ along the RMP-tree is made up of a state $(\mathbf{x}, \dot{\mathbf{x}})$ on a manifold along with an associated RMP $(\mathbf{a}_{\mathtt{v}}, \mathbf{M}_{\mathtt{v}})^{\mathcal{M}}$. Each edge $\mathtt{e}$ in the RMP-tree corresponds to a smooth map $\psi_{\mathtt{e}}$ from the given parent node manifold to the child node manifold. The root node in the RMP-tree, $\mathtt{r}$, is associated with the state $(\mathbf{q}, \dot{\mathbf{q}})$ in the configuration space $\mathcal{C}$ and its policy $(\mathbf{a}_{\mathtt{r}}, \mathbf{M}_{\mathtt{r}})^{\mathcal{C}}$. Let K be the number of leaf nodes in the RMP-tree. The leaf nodes $\{\mathtt{l}_k\}_{k=1}^K$ are associated with subtask RMPs $\{(\mathbf{a}_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})^{\mathcal{T}_{\mathtt{l}_k}}\}_{k=1}^K$. Each subtask RMP encodes a desired subtask acceleration policy, while the associated Riemannian metric assigns a state-dependent importance weight to the policy when combined with other policies. An example RMP-tree is shown in Figure 1a. Recursive application of RMP-algebra (cf. Appendix A) along the RMP-tree enables a weighted combination of the leaf node RMPs to generate a global configuration space policy $\mathbf{a}_{\mathtt{r}} = \pi(\mathbf{q}, \dot{\mathbf{q}})$.

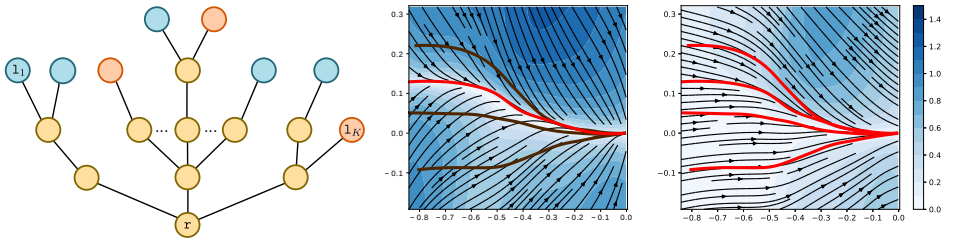
One important feature of RMPflow is that it preserves the stability property of leaf node policies: if all subtask RMPs are generated by systems in the form of (1), the combined policy in the configuration space is also in the form of (1) and hence Lyapunov stable. In the following section, we will make full use of the stability property of RMPflow to learn RMPs that can be combined (with both hand-specified RMPs and learned RMPs) into a stable policy.

4 Skill Reproduction via RMPflow

For robot manipulators, a skill may involve coordinated and constrained motion of different parts of a robot. To encode a skill, we first construct an RMP-tree with root node in the configuration space, specifically the joint space of the robot. The relevant robot body parts are added as child nodes of the root in the RMP-tree with edges given by the forward kinematics of the robot. Branching out further from these nodes are leaf nodes, corresponding to various subtask spaces. We propose to learn leaf node RMPs from human demonstrations. A human can choose to provide demonstrations for each subtask space either independently or simultaneously. Additional hand-specified leaf RMPs, for example obstacle avoidance and joint-limit RMPs [5] can be added along the RMP-tree to ensure feasibility of robot motions. The overall configuration space policy $\mathbf{a}_{\mathtt{r}}$ is found using RMPflow as described previously. It should be noted here that in order to ensure stability of the configuration space policy, all RMPs employed in this work are of the form (1).

4.1 Human-Guided Riemannian Motion Policies

In the kth subtask space $\mathcal{T}_{\mathtt{l}_k}$, corresponding to leaf node $\mathtt{l}_k$, our aim is to learn an RMP of the form (1). In lieu of this, we assume the availability of N human demonstrations $\{\boldsymbol{\zeta}_i^{(k)}\}_{i=1}^N$. Here the



(a) An example RMP-tree (b) Potential function (c) Potential w/ learned metric

Figure 1: (a) An example RMP-tree: both learned leaf node policies (blue) and hand specified policies can be combined to generate a global (root node) policy at the configuration space. (b) Vector field introduced by the learned potential function: every point in the trajectory are attracted to the single demonstration (red) with which the potential function is learned. (c) Vector field generated by the learned potential and metric: the space is warped by the learned metric so that the demonstrations can be reproduced.

ith trajectory demonstration is composed of T_i datapoints $\boldsymbol{\zeta}_i^{(k)} = \{\boldsymbol{\zeta}_{i,t}^{(k)}\}_{t=0}^{T_i}$, and all subtask space trajectories converge and come to rest at a common target position $\boldsymbol{\zeta}_{i,T_i}^{(k)} = \boldsymbol{\zeta}_T^{(k)}$. For notational clarity, we will drop the subtask space subscript k in the remainder of this section. However we will always refer to a particular subtask unless otherwise stated.

We observe here that the desired motion generated by the system (1) is mainly governed by the component $-\mathbf{M}(\mathbf{x})^{-1}\nabla\Phi(\mathbf{x})$, which can be seen as setting $\ddot{\mathbf{x}}^d$ along the negative *natural gradient* of the potential function on the manifold. The remaining components: the *damping acceleration* $-\mathbf{M}(\mathbf{x})^{-1}\,\mathbf{B}(\mathbf{x})\,\dot{\mathbf{x}}$ and the *curvature acceleration* $-\mathbf{M}(\mathbf{x})^{-1}\,\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}})$, simply ensure stable and geometrically consistent behavior. Hence, we view the learning from demonstration problem as equivalent to learning the aforementioned natural gradient descent subsystem. The damping component is pre-specified as $\mathbf{B}(\mathbf{x}) = \gamma_d\,\mathbf{M}(\mathbf{x})$, such that the damping acceleration $\mathbf{M}(\mathbf{x})^{-1}\,\mathbf{B}(\mathbf{x}) \equiv \gamma_d\,\mathbf{I}$ is always independent of the choice of metric.

The problem of learning the natural gradient descent subsystem is highly under-constrained, that is, there can be many different potential and metric combinations that can result in the same desired acceleration policy. We choose to bias the solution of this learning problem via a two-step process. Specifically, we first learn a less expressive potential function and then learn a Riemannian metric which warps the gradient of the potential such that the overall policy is expressive enough to accurately reproduce the demonstrations. According to [5], for $\mathbf{M}(\mathbf{x}), \mathbf{B}(\mathbf{x}) \in \mathbb{R}_+^{m\times m}$, the system in (1) converges to the forward invariant set $\mathcal{C}_\infty := \{(\mathbf{x},\dot{\mathbf{x}}) : \nabla\Phi(\mathbf{x}) = 0, \dot{\mathbf{x}} = 0\}$ [5]. Hence, we require our learned potential to have a minima at the target and our learned Riemannian metric to be globally positive definite. The remainder of this section details our learning procedure.

4.2 Learning Nominal Potential from Demonstrations

In order to learn a subtask space potential function, we first select a nominal demonstration $\boldsymbol{\zeta}^*$ which is the least dissimilar from the other demonstrations in terms of geometric features. One such metric of dissimilarity is Dynamic Time Warping (DTW) distance. The nominal demonstration is therefore given by the demonstration with the least mean DTW distance from other demonstrations.

Given the nominal trajectory demonstration $\boldsymbol{\zeta}^*$, we learn a potential $\Phi(\mathbf{x})$, which generates a dissipative field $-\nabla\Phi(\mathbf{x})$ that produces motions which: *(i)* converge smoothly towards the nominal trajectory, and *(ii)* follow the remaining trajectory after convergence. Furthermore, to ensure stability we also enforce

$$\nabla\Phi(\boldsymbol{\zeta}_T) = 0, \qquad \Phi(\mathbf{x}) \to \infty \text{ as } \mathbf{x} \to \infty, \tag{3}$$

so that the trajectories can always be bounded following the dynamics (1). Figure 1b shows a vector field generated by the negative gradient of an example potential field with aforementioned properties. We use an approach similar to that used by [25]. Specifically, the overall potential is given by a convex combination of potential elements centered at each trajectory point $\boldsymbol{\zeta}_t^*$,

$$\Phi(\mathbf{x}) = \sum_{t=1}^{t=T} w_t(\mathbf{x})\phi_t(\mathbf{x}), \quad \text{where } w_t(\mathbf{x}) = \frac{k(\mathbf{x},\boldsymbol{\zeta}_t^*)}{\sum_{t'=1}^{t'=T} k(\mathbf{x},\boldsymbol{\zeta}_{t'}^*)}, \quad k(\mathbf{x},\boldsymbol{\zeta}_t^*) = e^{-\frac{\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|^2}{2\sigma^2}} \tag{4}$$

Furthermore, each contributing potential element here is a summation of two components: $\phi_t(\mathbf{x}) = \phi_t^{\perp}(\mathbf{x}) + \phi_t^0$, whereby the component $\phi_t^{\perp} : \mathbb{R}^m \mapsto \mathbb{R}_+$ is a strictly convex function with a global minima at ζ_t^* and $\phi_t^0 \in \mathbb{R}_+$ is a bias term. As a consequence of the aforementioned decomposition, the gradient of the overall warped potential in (4) can also be decomposed as,

$$\nabla\Phi(\mathbf{x}) = \nabla\Phi^{\perp}(\mathbf{x}) + \nabla\Phi^{/\!/}(\mathbf{x}) \tag{5}$$

where the gradient component $\nabla\Phi^{\perp}(\mathbf{x})$ causes attractive pull towards the demonstration while the component $\nabla\Phi^{/\!/}(\mathbf{x})$ produces accelerations along the direction of motion of the demonstration. The motion along the trajectory is direct consequence of monotonically decreasing bias terms ϕ_t^0, such that the negative of the potential gradient aligns with the demonstrated motion. Due to this decomposition, we independently hand-design the function $\phi_t^{\perp}(\mathbf{x})$ as per our desired attractive accelerations towards the demonstration. Next, we learn the bias terms ϕ_t^0 such that the direction of motion governed by the potential at each data-point $\boldsymbol{\zeta}_t^*$ matches the demonstration.

As a first step in this procedure, we choose to go with the following attractive potential element [5],

$$\phi_t^{\perp}(\mathbf{x}) = \frac{1}{\eta}\log\left(e^{\eta\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|} + e^{-\eta\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|}\right), \quad \nabla\phi_t^{\perp}(\mathbf{x}) = s_\eta\big(\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|\big)\frac{\mathbf{x}-\boldsymbol{\zeta}_t^*}{\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|} \tag{6}$$

where $\eta > 0$ defines the effective smoothing radius of the function at the origin and $s_\eta(0) = 0$ 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_t^* where it smoothly decreases to zero. A trivial alternative to this function is a quadratic as used by Khansarizadeh et al [25]. However, the gradient of a quadratic function increases linearly with distance which can cause undesirably large accelerations far away from the demonstrations.

Towards the second step in the procedure, we learn the bias terms $\{\phi_t^0\}$. As mentioned before, we require negative natural gradient to match the accelerations from zero velocity. Also, we are only concerned with the direction of motion. Hence the potential learning problem becomes,

$$\begin{aligned} \min_{\{\boldsymbol{\phi}_t^0\}} \ & \frac{1}{T}\sum_{t=1}^{t=T} \|\hat{\dot{\boldsymbol{\zeta}}}_t^* + \nabla\Phi\big(\boldsymbol{\zeta}_t^*; \{\phi_t^0\}\big)\|^2 + \lambda\|\phi_t^0\|^2 \\ \text{s.t. } & 0 \le \phi_T^0 \le \phi_{t+1}^0 \le \phi_t^0 \quad \forall t = 1, \ldots, (T-1) \\ & \nabla\Phi(\boldsymbol{\zeta}_T^*) = 0 \end{aligned} \tag{7}$$

where $\hat{\dot{\boldsymbol{\zeta}}}_t^* = \frac{\dot{\boldsymbol{\zeta}}_t^*}{\|\dot{\boldsymbol{\zeta}}_t^*\|}$ is the direction of motion (with unit magnitude) and λ is the regularization parameter. Furthermore, the optimization constraints enforce the potential to decrease monotonically along the demonstration with a stationary point at the goal location $\boldsymbol{\zeta}_T$. The aforementioned optimization problem can be solved efficiently with off-the-shelf solvers, e.g. CVX [26].

4.3 Learning Riemannian Metric from Multiple Demonstrations

While a single demonstration can imply certain traits of motions, multiple demonstrations can further capture certain properties of the skill that can not be encoded by a single trajectory. For example, if all the demonstrations stay close to each other in a particular region of the state space, it informs that the motion is highly constrained in the region. In such part of the state space, the reproduced trajectories should follow the demonstrations closely so that the skill specifications are satisfied. Therefore, we choose to learn a Riemannian metric $\mathbf{M}(\mathbf{x})$ on the subtask space (manifold) to warp the learned potential function (4). The metric expands or contracts the space so that the attractive component is no longer uniform along the trajectories. Figure 1c shows an example vector field given by negative gradient of a nominal potential warped by a learned Riemannian metric.

To ensure the stability of the learned system, the Riemannian metric $\mathbf{M}(\mathbf{x})$ needs to be positive definite. Hence, we parameterize the metric by its Cholesky decomposition, $\mathbf{M}(\mathbf{x}) = \mathbf{L}(\mathbf{x})\,\mathbf{L}(\mathbf{x})^{\top}$, where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a lower-triangular matrix with positive diagonal entries. Let $\mathbf{l}_d(\mathbf{x}) \in \mathbb{R}^n$ and $\mathbf{l}_o(\mathbf{x}) \in \mathbb{R}^{\frac{1}{2}(n^2-n)}$ denote the vector given by collecting the diagonal and off-diagonal entries of $\mathbf{L}(\mathbf{x})$, respectively. In order for $\mathbf{L}(\mathbf{x})$ to be a Cholesky decomposition of the positive definite matrix $\mathbf{M}(\mathbf{x})$, we require each entries of $\mathbf{l}_d(\mathbf{x})$ to be strictly positive.

We represent the metric matrix as a neural-network with parameter θ. The neural network takes in the coordinate $\mathbf{x} \in \mathbb{R}^n$ and outputs $\mathbf{l}_d(\mathbf{x};\theta)$ and $\mathbf{l}_o(\mathbf{x};\theta)$ (Figure 2). To ensure that $\mathbf{l}_d(\mathbf{x};\theta)$ is strictly

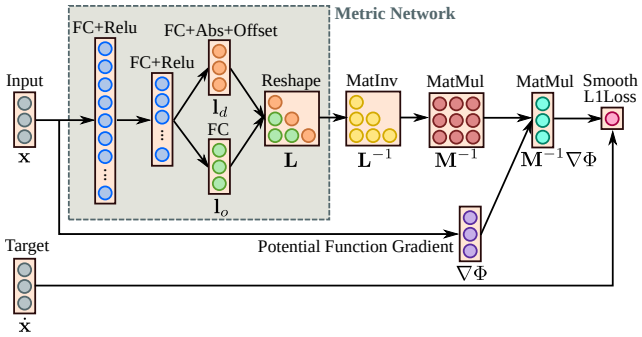


Figure 2: The structure of the neural network for metric learning. The first two layers are fully connected layers with Relu activation functions. The diagonal and off-diagonal elements of the lower triangular matrix $\mathbf{L}$ is then predicted through another fully connected layer. In order to ensure that the diagonal elements are strictly positive, the absolute value of the output of the layer is taken and a positive offset is added. Then, the inverse of the metric matrix computed to calculate the loss for training the network.

positive for all $\mathbf{x} \in \mathbb{R}^n$, we take absolute value of the output of the linear layer and add it with a small positive offset $\epsilon > 0$. Hence, lower-triangular matrix $\mathbf{L}(\mathbf{x};\theta)$ is always invertible and the Riemannian metric $\mathbf{M}(\mathbf{x};\theta)$ is guaranteed to be positive definite. The Riemannian metric learning problem seeks to find the parameters of the neural network such that final natural gradient descent subsystem (negative warped potential gradient) reproduces the demonstrated normalized velocities along the demonstrations,

$$\begin{aligned} \arg\min_{\theta} \quad & \sum_{i=1}^{N}\sum_{t=1}^{T_i} \mathcal{L}\big(-\mathbf{M}(\boldsymbol{\zeta}_{i,t};\theta)^{-1}\nabla\Phi(\boldsymbol{\zeta}_{i,t}),\ \hat{\dot{\boldsymbol{\zeta}}}_{i,t}\big) \\ \text{s.t.} \quad & \mathbf{M}(\boldsymbol{\zeta}_{i,t};\theta) = \mathbf{L}(\boldsymbol{\zeta}_{i,t};\theta)\,\mathbf{L}(\boldsymbol{\zeta}_{i,t};\theta)^{\top}, \end{aligned} \tag{8}$$

where $\mathcal{L}(\cdot,\cdot)$ can be any regression loss function, e.g. L2 loss, smooth L1 loss, and their regularized version. It is noteworthy that $\mathbf{M}(\cdot;\theta)$ need not be explicitly computed during training since $\mathbf{M}(\mathbf{x};\theta)^{-1} = \mathbf{L}(\mathbf{x};\theta)^{-\top}\mathbf{L}(\mathbf{x};\theta)^{-1}$, and $\mathbf{L}(\mathbf{x};\theta)^{-1}$ can be efficiently computed through forward-substitution due to the fact that $\mathbf{L}(\mathbf{x};\theta)$ is a lower-triangular matrix. Although the training is done on the first-order system. While executing the learned policy, the curvature term $\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}};\theta)$ needs to be computed as well. By definition of the curvature term (2), its calculation involves computing the partial derivatives $\frac{\partial M_{ij}(\mathbf{x};\theta)}{\partial x_k}$, which can be computed through back-propagation using standard deep learning frameworks such as PyTorch [27].

Limitations: First, we assume in this work that only the geometric features (i.e. positions and direction of motion/shape) of the demonstrated motions are important for the task. Reproducing the speed profiles of demonstrations would perhaps require learning the entire second-order system in (1) with a loss defined on the weighted-combination of policies given by RMPflow. Second, the Riemannian metric can not rotate the attractive potential field by more than 90 degrees. This limitation will manifest itself when demonstrations have large variations in their geometric features.

5 Experimental Evaluation

We evaluated the proposed approach on two manipulation tasks, including *door reaching* and *drawer closing*[2]. Both tasks were demonstrated on a 7-DOF Franka Emika robot with configuration space coordinate $\mathbf{q} \in \mathbb{R}^7$. For each skill, a human subject provided 6 demonstrations via kinesthetic teaching, starting from various joint angles of the robot. The demonstrations were recorded in the joint space such that the ith demonstration is defined as $\boldsymbol{\zeta}_i^{\boldsymbol{q}} := \{\boldsymbol{\zeta}_{i,t}^{\boldsymbol{q}}\}_{t=0}^{T_i}$, where $\boldsymbol{\zeta}_{i,t}^{\boldsymbol{q}} \in \mathbb{R}^7$.

The desired tasks require coordinated motion of the entire hand of the robot with respect to the target objects. To encode the motion of the entire wrist, we pick $k = 3$ control points on the hand; one defined at the center of the gripper and one each at the tips of the two-fingered gripper. We setup an RMP-tree with root in the joint space of the robot and leaf nodes representing 3-dimensional subtasks, one per control point, under the mappings $\{\psi^{(k)}\}_{k=1}^{3}$. Specifically, a subtask map is defined as a composition $\psi^{(k)} := g^{(k)} \circ f^{(k)}$. Here $f^{(k)} : \mathbb{R}^7 \mapsto \mathbb{R}^3$ maps the joint angles to the kth

[2]Video available at: https://youtu.be/9jvz5fE_1dM.

control point position under the robot's forward kinematics. On the other hand, $g^{(k)}(\mathbf{x}) = \mathbf{x} - \mathbf{p}^{(k)}$ defines a translation such that the target location $\mathbf{p}^{(k)} \in \mathbb{R}^3$ coincides with the origin of the subtask space [3]. For each subtask, we independently learn a *human-guided* RMP of the form (1) under the transformed subtask space demonstrations $\{\boldsymbol{\zeta}_i^{(k)}\}_{i=1}^N$. Appendix B provides additional details on the learning pipeline and the integrated system.

To execute the skill in a new environment, additional hand-specified joint limit RMPs, and obstacle avoidance RMPs are also added as leaf RMPs to the RMP-tree. These hand-specified RMPs dictate the kinematic and environmental constraints as detailed in [5]. All the policies are resolved in real-time under the learned and hand-specified Riemannian metrics by recursively running RMPflow on the RMP-tree. It should be noted that the kinematic constraints enforced along the RMP-tree ensures the coordination of motion in various subtask spaces.

The *drawer closing* task required the robot to reach the handle of a drawer from a given initial position and push it closed. Figure 4a–4b shows the demonstrations transformed in the 3 aforementioned subtask spaces. It should be noted that the closing motion not only requires a straight line motion by the end-effector after making contact with the drawer handle, but also requires the wrist to align with the face of the handle. Figure 4c–4d shows the reproductions from the same initial positions as the demonstrations while Figure 5 shows an instance of reproduction on the real robot. Among all the 6 trials, the robot is able to successfully reproduce the demonstrations and close the drawer.

The *door reaching* task required the robot to start from inside a cabinet, going around the cabinet door and reaching for the door handle outside the door. The robot is required to stop at a standoff position a small distance away from the drawer handle. This task requires a highly constrained motion that results in the end-effector moving along a C-shaped arc. Furthermore, as before, the entire wrist trajectory is important here since the robot is required to reach the handle at a certain relative angle. Figure 6a–6b shows the demonstrated trajectories while Figure 6c–6d shows the reproductions from the same set of initial positions. Figure 7 shows snapshots of a task reproduction on the real robot. Once again we notice all the reproductions to successfully achieve the task. Furthermore, to test the reactive behavior of our policy, we displace the door during execution. As noticeable in Figure 8, the robot successfully react to the the change in goal location and hence complete the task.

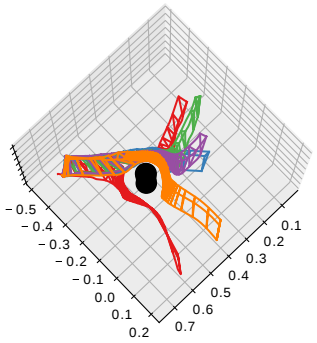


Figure 3: Reproductions of the *drawer closing* task with a cylindrical obstacle (in black) in the scene. The positions of the 3 control points on robot hand are shown as vertices of a triangle.

Another important feature our approach inherits from RMPflow is obstacle avoidance. We desire the generated motions to be collision-free in order to be feasible in any new environment. To test the obstacle avoidance behavior, we place a cylindrical obstacle in the environment such that it hinders the robot's motion towards the drawer for the *drawer closing* task. We notice successful completion of the task as the robot is able to go around the obstacles in all 6 trials (Figure 3).

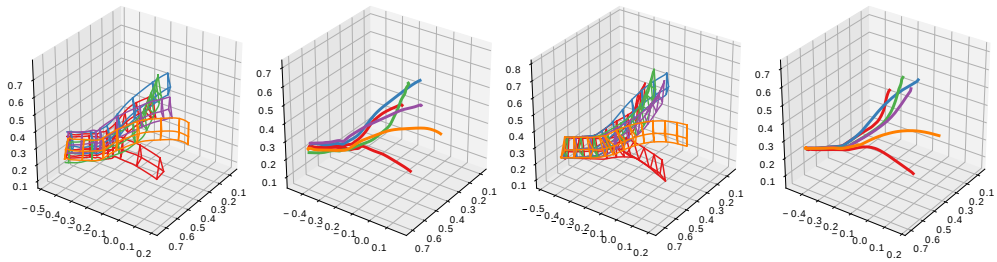
6 Conclusion

We introduced an approach for learning and reproducing complex tasks, composed of multiple interrelated subtasks. Our approach is capable of learning inherently-stable reactive polices in these subtask spaces directly from human demonstrations. For task reproduction, our method utilizes RMPflow to carry out policy resolution and generate a stable joint-space policy that enables simultaneous execution of various learned subtasks. Furthermore, the motions generated by the combined policy adhere to the robot's kinematics and environmental constraints. Experimental results demonstrate that the proposed approach can capture the desired behaviors of multiple robot links in order to accomplish constrained manipulation tasks.

Acknowledgments

This research was supported in part by NVIDIA Research, NSF CAREER award 1750483, and NSF IIS award 1637562.

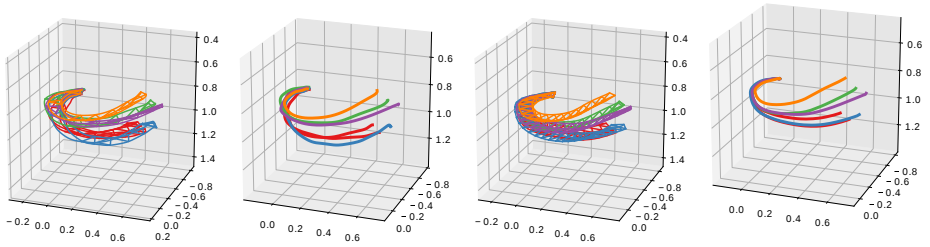
[3] Target location $\mathbf{p}^{(k)}$ is attached to the target object. This enables the policy to react to object displacement.



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

Figure 4: Trajectories for the *drawer closing* task. (a), (c): The motion of 3 control points on the robot hand, shown as vertices of a triangle, along the demonstrated and reproduced trajectories respectively. (b), (d): The demonstrated and reproduced end-effector (center of hand) trajectories.

Figure 5: The *drawer closing* task. The robot successfully closes the drawer from a new initial configuration.



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

Figure 6: Trajectories for the *cabinet door reaching* task. (a), (c): The motion of 3 control points on the robot hand, shown as vertices of a triangle, along the demonstrated and reproduced trajectories respectively. (b), (d): The demonstrated and reproduced end-effector (center of hand) trajectories.

Figure 7: The cabinet *door reaching task*. The robot manages to reach the cabinet door handle despite of the new initial configuration and new door configuration.

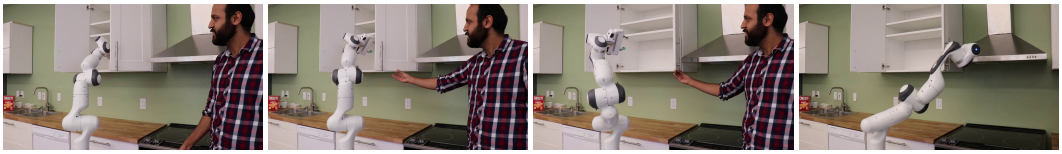


Figure 8: Reactivity. The door handle location is displaced during execution and the robot can be seen to adapting to the new target.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[2] S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[3] K. Neumann and J. J. Steil. Learning robot motions with stable dynamical systems under diffeomorphic transformations. *Robotics and Autonomous Systems*, 70:1–15, 2015.

[4] H. C. Ravichandar and A. Dani. Learning position and orientation dynamics from demonstrations via contraction analysis. *Autonomous Robots*, 43(4):897–912, 2019.

[5] 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. *Proceedings of the 13th Annual Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2018.

[6] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

[7] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[8] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[9] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[10] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[11] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 489–494. IEEE, 2009.

[12] 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*, 33(9):1251–1270, 2014.

[13] 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)*, 2018.

[14] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[15] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. 2008.

[16] A. D. Dragan, K. Muelling, J. A. Bagnell, and S. S. Srinivasa. Movement primitives via optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2339–2346, 2015.

[17] 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*, 2(2):819–826, 2017.

[18] 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.

[19] S. Calinon and A. Billard. Statistical learning by imitation of competing constraints in joint space and task space. *Advanced Robotics*, 23(15):2059–2076, 2009.

[20] A. Paraschos, R. Lioutikov, J. Peters, and G. Neumann. Probabilistic prioritization of movement primitives. *IEEE Robotics and Automation Letters*, 2(4):2294–2301, 2017.

[21] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[22] 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.

[23] J. Silvério, S. Calinon, L. Rozo, and D. G. Caldwell. Learning task priorities from demonstrations. *IEEE Transactions on Robotics*, 35(1):78–94, 2019.

[24] F. Bullo and A. D. Lewis. *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*. Springer New York, 2005.

[25] S. M. Khansari-Zadeh and O. Khatib. Learning potential functions from human demonstrations with encapsulated dynamic and compliant behaviors. *Autonomous Robots*, 41(1):45–69, 2017.

[26] M. Grant, S. Boyd, and Y. Ye. CVX: Matlab software for disciplined convex programming, 2009.

[27] 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. In *NIPS Autodiff Workshop*, 2017.

[28] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] T. Schmidt, R. A. Newcombe, and D. Fox. Dart: Dense articulated real-time tracking. In *Robotics: Science and Systems*, volume 2. Berkeley, CA, 2014.

Appendices

A RMPflow

In this appendix, we provide a brief introduction of RMPflow [5], the computational framework for policy generation with RMPs. We refer the readers to [5] for detailed introduction and theoretical analysis of RMPflow.

Two components that form the basis of RMPflow include: *1*) RMP-tree: a directed tree encoding the structure of the task map, and *2*) RMP-algebra: a set of operations to propagate information across the RMP-tree. An RMP-tree is a directed tree initiating at the root node, branching out and culminating at the leaf nodes, with edges connecting the parent-child node pairs. Specifically, each node $\mathtt{v}$ along the RMP-tree is made up of a state $(\mathbf{x}, \dot{\mathbf{x}})$ on a manifold along with an associated RMP $(\mathbf{a}_\mathtt{v}, \mathbf{M}_\mathtt{v})^{\mathcal{M}}$. Each edge $\mathtt{e}$ in the RMP-tree corresponds to a smooth map $\psi_\mathtt{e}$ from the given parent node manifold to the child node manifold. The root node in the RMP-tree, $\mathtt{r}$, is associated with the state $(\mathbf{q}, \dot{\mathbf{q}})$ in the configuration space $\mathcal{C}$ and its policy $(\mathbf{a}_\mathtt{r}, \mathbf{M}_\mathtt{r})^{\mathcal{C}}$. Let K be the number of leaf nodes in the RMP-tree. The leaf nodes $\{\mathtt{l}_k\}_{k=1}^K$ are associated with subtask policies $\{(\mathbf{a}_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})^{\mathcal{T}_{\mathtt{l}_k}}\}_{k=1}^K$.

The subtask policies along the RMP-tree are combined using RMP-algebra. To illustrate how RMP-algebra operates, consider a node $\mathtt{u}$ in the RMP-tree with N child nodes $\{\mathtt{v}_j\}_{j=1}^N$ (see Figure 9). Let $\{\mathtt{e}_j\}_{j=1}^N$ denote the edge between the parent node $\mathtt{u}$ and the child nodes $\{\mathtt{v}_j\}_{j=1}^N$. RMP-algebra consists of three operators:

(*i*) `pushforward` propagates the state of a node in the RMP-tree to update the states of its child nodes. Let $(\mathbf{x}, \dot{\mathbf{x}})$ and $\{\mathbf{y}_j, \dot{\mathbf{y}}_j\}_{j=1}^N$ be the state associated with the parent node and the child nodes, respectively. The state of its jth child node $\mathtt{v}_j$ is computed as $(\mathbf{y}_j, \dot{\mathbf{y}}_j) = (\psi_{\mathtt{e}_j}(\mathbf{x}), \mathbf{J}_{\mathtt{e}_j}(\mathbf{x})\,\dot{\mathbf{x}})$, where $\psi_{\mathtt{e}_j}$ is the smooth map associated with the edge $\mathtt{e}_j$ and $\mathbf{J}_{\mathtt{e}_j} = \partial_\mathbf{x}\psi_{\mathtt{e}_j}$ is the Jacobian matrix.

(*ii*) `pullback` propagates the natural form of RMPs $\{[\mathbf{f}_{\mathtt{v}_j}, \mathbf{M}_{\mathtt{v}_j}]^{\mathcal{N}_j}\}_{j=1}^N$ from the child nodes to the parent node. The RMP associated with the parent node $[\mathbf{f}_\mathtt{u}, \mathbf{M}_\mathtt{u}]^{\mathcal{M}}$ is computed as,

$$\mathbf{f}_\mathtt{u} = \sum_{j=1}^N \mathbf{J}_{\mathtt{e}_j}^\top (\mathbf{f}_{\mathtt{v}_j} - \mathbf{M}_{\mathtt{v}_j}\dot{\mathbf{J}}_{\mathtt{e}_j}\dot{\mathbf{x}}), \qquad \mathbf{M}_\mathtt{u} = \sum_{j=1}^N \mathbf{J}_{\mathtt{e}_j}^\top \mathbf{M}_{\mathtt{v}_j}\mathbf{J}_{\mathtt{e}_j}. \tag{9}$$

The natural form of RMPs are used since they more efficient to combine.

(*iii*) `resolve` maps an RMP from its natural form $[\mathbf{f}_\mathtt{u}, \mathbf{M}_\mathtt{u}]^{\mathcal{M}}$ to its canonical form $(\mathbf{a}_\mathtt{u}, \mathbf{M}_\mathtt{u})^{\mathcal{M}}$ with $\mathbf{a}_\mathtt{u} = \mathbf{M}_\mathtt{u}^\dagger \mathbf{f}_\mathtt{u}$, where $\dagger$ denotes Moore-Penrose inverse.

Given configurations state $(\mathbf{q}(t), \dot{\mathbf{q}}(t))$ at time t, RMPflow computes the global policy $\pi(\mathbf{q}(t), \dot{\mathbf{q}}(t)) = \mathbf{a}_\mathtt{r}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$ through the following procedure. The `pushforward` operator is first recursively applied to the RMP-tree to propagate the state associated with each node in the RMP-tree. Then, the subtask policies $\{(\mathbf{f}_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})\}_{k=1}^K$ are evaluated by the leaf nodes and combined recursively along the RMP-tree by the `pullback` operator. the `resolve` operator is finally applied on the root node to compute the acceleration policy $\mathbf{a}_\mathtt{r}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$.

B Details of the Experiments

Here we provide some details of the experiments, including the learning pipeline and the integrated perception and control system.

B.1 The Learning Pipeline

Data Collection For both the drawer closing experiment and the cabinet door reaching experiment, we collect 6 human demonstrations starting from different initial configurations of the robot through kinesthetic teaching. The initial configuration of the environment, i.e., the cabinet door and the drawer, respectively, remains fixed across different demonstrations. For the drawer closing experiment, in particular, the initial configuration of the drawer is always fixed across the data collection

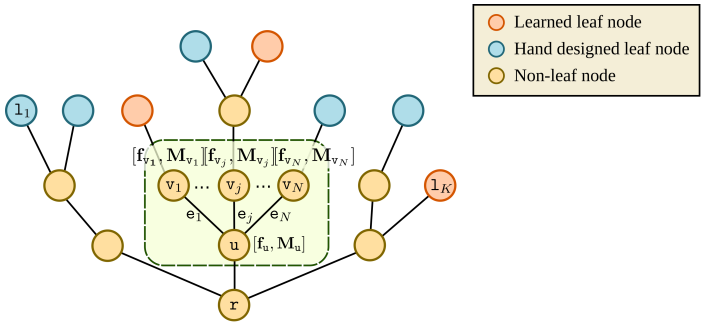


Figure 9: An example RMP-tree. The root of RMP-tree is associated with the configuration space while the leaf nodes are associated with subtasks. The policies for the subtasks can be either hand-designed or learned. The RMP-algebra propagates information along the tree. The green block contains a segment of the RMP-tree to illustrate the RMP-algebra.

phase and the testing phase. In the cabinet door reaching experiment, however, the initial configuration of the cabinet door is recorded by the perception system (see Appendix B.2). In the testing phase, we vary the initial configuration of the cabinet door across different trials. The demonstrations are recorded in the joint space.

Data Preprocessing We define the subtask spaces in the reference frame of the object, i.e., the cabinet door handle and the center of the drawer front, respectively. In particular, we consider 3 control points on the wrist of the robot making up 3 subtask spaces of $\mathbb{R}^3$. We also translate the origin of the subtask spaces such that all demonstrations converge to the origin in each subtask spaces. Hence, given the demonstrations in the joint space, we transfer the demonstrations into the subtask spaces using the forward kinematics of the robot composited with a rigid body transformation.

To preprocess the demonstrated trajectories, we first remove the static segments in the trajectories. The trajectories are then smoothed with a Savitzky-Golay filter. We resample the trajectories with 200 sample points evenly spaced in time using cubic interpolation. The velocity at the sample points are computed through finite-difference. Since we assume that the magnitude of the velocities are not important to the skill, we rescale the velocity at each sample point so that the trajectory has unit velocity for each sample point.

Metric Learning The number of nodes in the two fully connected layers are 128 and 64, respectively. The neural network is trained by the Adam optimizer [28] with a learning rate of 0.005 and weight decay of 3×10^{-5}. Since the size of the training set is relatively small, we use batch update and terminate training after 1000 updates. The average training time for metric learning on a subtask space is 15.76s. The reported training time is an average over 10 runs on a CPU machine with an Intel Core i7 processor.

B.2 The Integrated System

Drawer Closing Experiment No perception systems are used for the drawer closing experiment. The initial configuration of the drawer is fixed throughout data collection and testing. The robot is considered moving in an empty space except for the trial where a simulated cylinder-shaped obstacle is added to the environment.

Cabinet Door Reaching Experiment In the cabinet door reaching experiment, the configuration of the robot and the environment, e.g., cabinet door, is tracked by an overhead RGB-D kinect camera using DART [29]. The perception system updates the state of the robot and the environment at 30Hz.

Hand-crafted RMPs In addition to the learned RMPs, obstacle avoidance RMPs and joint limit RMPs detailed in [5] are also active during the testing phase to avoid collision with obstacles and exceeding the joint limits.

Control Frequency The learned RMPs operate at 50Hz during execution while the other hand-crafted RMPs operate at 1000Hz.