Towards Coordinated Robot Motions: End-to-End Learning of Motion Policies on Transform Trees

M. Asif Rana*[1,3], Anqi Li*[2,3], Dieter Fox[2,3], Sonia Chernova[1], Byron Boots[2,3], and Nathan Ratliff[3]

***Abstract*— Generating robot motion that fulfills multiple tasks simultaneously is challenging due to the geometric constraints imposed on the robot. In this paper, we propose to solve multi-task problems through learning structured policies from human demonstrations. Our structured policy is inspired by RMPflow, a framework for combining subtask policies on different spaces. The policy structure provides the user an interface to 1) specifying the spaces that are directly relevant to the completion of the tasks, and 2) designing policies for certain tasks that do not need to be learned. We derive an end-to-end learning objective that is suitable for the multi-task problem, emphasizing the distance between generated motions and demonstrations measured on task spaces. Furthermore, the motion generated from the learned policy class is guaranteed to be stable. We validate the effectiveness of our proposed learning framework through qualitative and quantitative evaluations on three robotic tasks on a 7-DOF Rethink Sawyer robot.**

I. Introduction

Robotic systems often need to consider multiple tasks simultaneously to achieve their overall missions. Consider the task of placing an object on a shelf. The end-effector of the robot needs to reach a goal location, while the whole body of the robot is required to avoid collisions with the shelf. Generating motions that fulfills all tasks simultaneously is challenging, as the execution of each task is not independent due to the geometric constraints of the robot.

Recently, a framework, called RMPflow [1], has been proposed for solving the aforementioned multi-task problem. RMPflow generates robot motion by combining task policies defined on different (and potentially correlated) spaces. It provides each task with an acceleration policy and a state-dependent importance weight matrix. The tuple containing the acceleration policy and the importance weight is called a Riemannian motion policy (RMP) [2]. These RMPs are then combined into a configuration space policy through an algorithm called RMPflow that solves a weighted least-squares problem [3]. It has been shown in [1] that, when the RMPs satisfy certain geometric conditions, the motion generated by RMPflow is Lyapunov stable. Due to its stability properties and computational efficiency, RMPflow has be applied to a variety of robotic systems for generating complex motions in multi-task setting, e.g. [4]–[8].

Despite its rich expressivity, it is in general hard to design an RMP: it requires designing the state-dependent importance weight matrix, which, for the stability properties to hold, also adds complications to policy design. One way to overcome this design difficulty is through learning RMPs from human demonstrations, in particular, kinesthetic teaching, as it provides the user an intuitive way communicate desired behaviors with the robot.

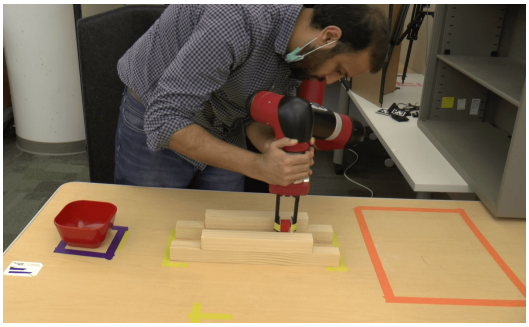


Fig. 1: A human providing demonstrations to the robot for a manipulation task through kinesthetic teaching.

Using the RMPflow structure for learning from demonstrations (LfD) has the following benefits. First, the learned motions are guaranteed to be stable as long as all task RMPs are properly designed or parameterized [5]. Second, it provides the user a convenient interface to specifying which spaces are relevant to the tasks. For instance, if the end-effector position is what matters to the tasks, the human may provide joint space trajectories that seem conflicting with one another (although they are consistent when viewed in the workspace). Directly regressing on the joint space trajectories will produce large errors both in the joint space and in the workspace. Lastly, it also allows policies to be hand-designed for certain tasks, e.g. joint damping, redundancy resolution, collision avoidance, while the other policies are learned from data. This provides the user with freedom in deciding which task policies should be learned.

Existing work [9] has explored learning RMPs from human demonstrations on complex robotic tasks. In [9], each RMP is *independently* learned to reproduce the demonstrated trajectories mapped to the corresponding task space. After learning, these independently learned RMPs, as well as hand-designed ones, are combined together by the RMPflow algorithm [1] to produce the configuration space policy. It empirically demonstrates that the learned RMPs can be combined with hand-designed RMPs, such as collision avoidance and joint limit RMPs, after learning to satisfy task constraints and generalize to new obstacle configurations. The major limitation of this work is that the learned importance weight matrices are not learned to provide the proper trade-offs between policies, as each policy is learned independently. In addition, the geometric constraints (e.g. induced by the

* Indicates equal contribution. [1] Georgia Institute of Technology. [2] University of Washington. [3] NVIDIA Research

kinematics of the robot) between tasks are not considered during learning. Despite its empirical success, the approach introduced in [9] is not able to fully exploit the benefits provided by the RMPflow structure. We will further demonstrate these limitations in the experiments.

In this paper, we propose a velocity-control[1] motion generation algorithm similar to RMPflow. We show that this new velocity-control formulation enjoys all the benefits of RMPflow mentioned above, while providing a simpler structure for parameterizing stable policies. Furthermore, we provide a principled approach to learning structured policies from human-demonstrations in an end-to-end fashion. In contrast to [9], during learning, we jointly consider all RMPs being learned as well as the hand-specified ones. A new objective function is proposed to measure the distance between the human demonstrations and the learned motions in all task spaces. We differentiate through the weighted least-squares optimization procedure induced by the proposed RMPflow-type algorithm so that the geometric constraints between task spaces are accounted during learning. Finally, we incorporate an expressive parameterization of RMPs through learning a latent space policy, which is inspired a recent work in learning diffeomorphisms [10].

II. Related Work

Motion Generation for Multi-Task Problems: A general strategy for solving this multi-task motion generation problem is to generate (through either designing or learning) controllers or policies for each task independently, and then provide a high-level rule to combine them. Null-space or hierarchical operational control assigns priorities to the tasks, and only allows the lower-priority policies to act on the null space of high-priority tasks [11]–[14]. However, these approaches could suffer from algorithmic singularities, due to multiple projections, that may arise when there are a large number of tasks. If this occurs, the system can easily become unstable [15]. Instead of assigning priorities, RMPflow provide each task with a state-dependent importance weight matrix, and the motion is generated through solving a weighted least-squares problem defined by the importance weight matrix [3]. As is discussed in Section I, the motion generated by RMPflow is Lyapunov stable as long as all RMPs follow certain geometric structure.

Learning from Human Demonstrations: Several approaches seek to learn policies from human demonstrations. These methods are typically grouped into two categories: 1) time-dependent policy learning [16], [17], and 2) time-invariant policy learning [10], [18]–[20]. As elaborated in [18], time-dependent methods, including the well-known dynamic movement primitives [17], [21], are susceptible to fail when either the environment or the time-horizon of motions is dynamic. On the other hand, time-invariant policies, in the absence of stability guarantees, are likely to suffer from the compounding error problem [22]. Most previous approaches for learning stable time-invariant policies [10], [18]–[20], however, are limited to learning motions associated with a single task assigned to a given robot body part (e.g. center of the end-effector).

Learning Riemannian Motion Policies: Recent works have explored learning RMPs from data [7], [9], [23], [24]. Meng et al. [7] learn to map perception input to RMPs through imitating hand-designed RMPs in an autonomous navigation setting. However, the learned RMPs does not satisfy the geometric condition for generating stable motions. To fine-tune the motion generated by fixed hand-designed policies, Mukadam et al. [23] add learnable scalar weights to the RMPflow algorithm. The expressively of this policy class, however, is limited by the fixed, hand-designed policies. Aljalbout et al. [24] propose to learn collision avoidance RMPs through reinforcement learning, although the learned policy is not guaranteed to be stable.

The work most relevant to this paper is [9], where it also learns RMPs from human demonstrations. To provide stability guarantees to the learned policy, it incorporates a neural network architecture to ensure the positive-definiteness of the importance weight matrix. However, as is mentioned in Section I, this work has the limitation due to the fact that the policies are learned independently, and also that the policy parameterization has limited capacity.

III. Motion Generation with Transform Trees

In this section, we propose a new motion generation for velocity-based motion control inspired by RMPflow [1]. In Section III.B, we introduce the optimization problem for the velocity-based control problem. We then introduce our proposed algorithm in Section III.C and analyze the stability property of the algorithm in Section III.D.

A. Motion Generation for Multi-task Problems

The goal of motion generation is to provide a configuration space trajectory given the desired behaviors on the task space. We consider multi-task problems, where the robot can be tasked with multiple specifications, which we call *subtasks*, simultaneously. For example, consider the task of placing an object on a shelf. The end-effector of the robot needs to reach a goal location, while the whole body of the robot is required to avoid collisions with the shelf. A subtask can sometimes be more easily specified on its individual space, rather than the joint configuration space. For example, collision avoidance can be described as a behavior on the 1-dimensional distance field. This yields a motion generation problem with subtasks defined on different *subtask spaces*.

It should be noted that, the substask spaces are often not independent but intertwined together as the image of the common configuration space. Therefore, solving the multi-task problem requires coordination between multiple robot body parts in a complex way.

B. Optimization Problem for Velocity-based Control

Consider a robot with its configuration space, denoted $\mathcal{C}$, given by a smooth d-dimensional manifold. We assume

[1]This is practical as modern robots often have a good low-level tracking control, allowing position and velocity-based control interfaces.

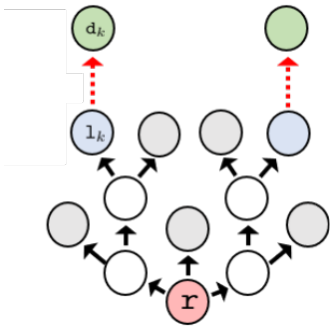


Fig. 2: A transform tree with root in the configuration space alongside hand-specified subtask/leaf nodes (grey) and learned subtask nodes (blue). Each learned subtask node is linked to a latent subtask node (green) under a chained map $\psi_{\mathtt{l}_k \to \mathtt{d}_k} = \psi_1 \circ \cdots \circ \psi_M$.

that the configuration space $\mathcal{C}$ admits global coordinates, called *generalized coordinates*, denoted $\mathbf{q} \in \mathbb{R}^d$. An example of generalized coordinates is the joint angles for a robot manipulator. In contrast to RMPflow [1], which considers acceleration policies, we are interested instead in encoding robot motion as a feedback velocity policy, i.e. $\dot{\mathbf{q}} = \pi(\mathbf{q})$. Such velocity-control problem usually occurs when there is a low-level tracking controller [25] applied in conjunction with the policy π.

We assume that the overall task can be decomposed as a set of K *subtasks* defined on different *subtask spaces*, denoted $\{\mathcal{T}_k\}_{k=1}^K$. Let $\psi_k : \mathcal{C} \to \mathcal{T}_k$ be the subtask map for the k-th subtask, and let $\mathbf{z}_k \in \mathbb{R}^n$ be the generalized coordinates on the subtask space $\mathcal{T}_k$, i.e., $\mathbf{z}_k = \psi_k(\mathbf{q})$. We describe the k-th *subtask policy* as a tuple $(\mathbf{v}_k, \mathbf{M}_k)$, consisting of a nominal velocity policy $\mathbf{v}_k : \mathbb{R}^n \to \mathbb{R}^n$ along with a state-dependent matrix-valued importance weight matrix $\mathbf{M}_k : \mathbb{R}^n \to \mathbb{R}^{n\times n}_{++}$. The importance weight matrix $\mathbf{M}_k(\mathbf{z}_k)$ denotes the directional importance of the velocity policy $\mathbf{v}_k(\mathbf{z}_k)$ at point $\mathbf{z}_k$.

Given a collection of subtask policies $\{(\mathbf{v}_k, \mathbf{M}_k)\}_{k=1}^K$, our goal is to generate a structured configuration space velocity policy π which trades off the errors to the velocity policies $\mathbf{v}_k$ viewed on each subtask space with an importance weight defined by $\mathbf{M}_k$. Formally, the policy is given by the solution to the following weighted least-squares problem:

$$\pi(\mathbf{q}) \coloneqq \arg\min_{\mathbf{u}} \sum_{k=1}^{K} \left\| \mathbf{v}_k(\psi_k(\mathbf{q})) - \mathbf{J}_k(\mathbf{q})\,\mathbf{u} \right\|^2_{\mathbf{M}_k(\psi_k(\mathbf{q}))} \tag{1}$$

where $\mathbf{J}_k = \partial_{\mathbf{q}} \psi_k$ is the Jacobian of the subtask map ψ_k. To look deeper into the objective (1), the term $\mathbf{v}_k(\psi_k(\mathbf{q})) = \mathbf{v}_k(\mathbf{z}_k)$ is the desired velocity in the subtask space $\mathcal{T}_k$, and the term $\mathbf{J}_k(\mathbf{q})\mathbf{u}$ is the velocity in the substask space $\mathcal{T}_k$ if we apply configuration space velocity $\mathbf{u}$. Therefore, the objective function (1) seeks to minimize the sum of deviation in each subtask space weighted by the importance weight $\mathbf{M}_k(\psi_k(\mathbf{q})) = \mathbf{M}_k(\mathbf{z}_k)$.

C. The Algorithm for Policy Composition

Although the subtask maps $\{\psi_k\}_{k=1}^K$ can be viewed as independent when solving (1), the evaluation of $\{\psi_k\}_{k=1}^K$, and similarly, their Jacobians $\{J_k\}_{k=1}^K$, can benefit from *reusing computation*. As an example, for robots with kinematic chain structure, the poses of the earlier links (closer to the base) are implicitly computed while evaluating the poses of the end effector. Such structure in the subtask maps can therefore lend itself amenable to computationally efficient algorithms.

Similar to RMPflow [1], we use a *transform tree* to describe a tree-structured map from the configuration space to subtask spaces. Each node $\mathtt{u}$ along the transform tree is associated with a manifold $\mathcal{M}$, each edge $\mathtt{e}_j$ corresponds to a smooth map $\psi_{\mathtt{e}_j} \coloneqq \psi_{\mathtt{v}_j;\mathtt{u}}$ from the parent node manifold to the manifold associated with child node $\mathtt{v}_j$. The root node in the transform tree, $\mathtt{r}$, corresponds to the configuration space $\mathcal{C}$, and the leaf nodes $\{\mathtt{l}_k\}_{k=1}^K$ are associated with subtask spaces $\{\mathcal{T}_k\}_{k=1}^K$. The subtask map is then computed as $\psi_k = \psi_{\mathtt{l}_k;\mathtt{r}}$, i.e., through aggregating the maps from the root node all the way to the leaf node $\mathtt{l}_k$.

We propose a computational framework for solving (1) through propagating information along the transform tree. The algorithm consists of the following four stages:

1) *Forward pass:* From the root node to the leaf nodes, the coordinate associated with each intermediate node is calculated based on the coordinate of its parent node: $\mathbf{y}_j = \psi_{\mathtt{e}_j}(\mathbf{x})$, where $\mathbf{x}$ and $\mathbf{y}_j$ are the coordinates for the parent and the child node, respectively, and $\psi_{\mathtt{e}_j}$ is the map associated with the edge. The Jacobian matrix associated with each edge, $\mathbf{J}_{\mathtt{e}_j}$, is also evaluated.
2) *Leaf evaluation:* For each leaf node, evaluate the subtask velocity policy $\mathbf{v}_k(\mathbf{z}_k)$ and $\mathbf{M}_k(\mathbf{z}_k)$. Then compute their product $\mathbf{p}_k(\mathbf{z}_k) = \mathbf{M}_k(\mathbf{z}_k)\mathbf{v}_k(\mathbf{z}_k)$.
3) *Backward pass:* From the leaf nodes to the root node, recursively compute the polices at each node based the policies at the child nodes: Consider a node $\mathtt{u}$ with N child nodes $\{\mathtt{v}_j\}_{j=1}^N$. The policy at $\mathtt{u}$ is calculated as,
$$\mathbf{p}_{\mathtt{u}} = \textstyle\sum_{j=1}^N \mathbf{J}_{\mathtt{e}_j}^\top \mathbf{p}_{\mathtt{v}_j}, \quad \mathbf{M}_{\mathtt{u}} = \textstyle\sum_{j=1}^N \mathbf{J}_{\mathtt{e}_j}^\top \mathbf{M}_{\mathtt{v}_j} \mathbf{J}_{\mathtt{e}_j}. \tag{2}$$
where $\mathtt{e}_j$ is the edge from $\mathtt{u}$ to $\mathtt{v}_j$.
4) *Resolve:* At the root node, the velocity policy is solved as $\pi(\mathbf{q}) = \mathbf{M}_{\mathtt{r}}^{-1}\,\mathbf{p}_{\mathtt{r}}$.

D. Stability Properties of the Proposed Algorithm

The configuration space motions governed by (1) exhibit several desirable properties if the leaf node velocity policies on the transform tree take the form:

$$\mathbf{v}_k(\mathbf{z}_k) = -\mathbf{M}_k^{-1}(\mathbf{z}_k)\,\nabla_{\mathbf{z}_k}\Phi_k(\mathbf{z}_k), \tag{3}$$

where $\Phi_k : \mathbb{R}^n \to \mathbb{R}$ is called the potential function. We call (3) the natural gradient flow dynamics, which can be viewed as a continuous-time version of natural gradient descent [26]. It evolves along steepest descent direction of Φ_k on a Riemannian manifold defined by the Riemannian metric $\mathbf{M}_k$. Under the assumption that each leaf node policy is

given by a natural gradient flow dynamics (3), the following properties hold for the generated root node velocity policy:

- *Closure:* the motion follows natural gradient flow with metric $\mathbf{M}_\mathtt{r} = \sum_{k=1}^{K} \mathbf{J}_k^\top \mathbf{M}_k \mathbf{J}_k$, and potential function $\Phi_\mathtt{r} = \sum_{k=1}^{K} \Phi_k \circ \psi_k$, where $\circ$ denotes map composition;
- *Stability:* the system converges to the stationary points of the potential function $\Phi_\mathtt{r}$.

Formally, the above properties are stated in the following theorem:

Theorem III.1. *Assume that the importance weight matrix at the root node is non-singular, i.e.* $\mathbf{M}_\mathtt{r} \succ 0$. *If each subtask policy is given by natural gradient flow* (3)*, the root node policy is given by natural gradient flow* $\dot{\mathbf{q}} = -\mathbf{M}_\mathtt{r}^{-1} \nabla_\mathbf{q} \Phi_\mathtt{r}$*, where* $\Phi_\mathtt{r} = \sum_{k=1}^{K} \Phi_k \circ \psi_k$. *Further, if* $\Phi_\mathtt{r}$ *is proper, continuously differentiable and lower bounded, the system* $\dot{\mathbf{q}} = \pi(\mathbf{q})$ *converges to a forward invariant set* $\mathcal{C}_\infty \coloneqq \{\mathbf{q} : \nabla_\mathbf{q} \Phi_\mathtt{r} = 0\}$.

Proof sketch: Assume each subtask policy is given by natural gradient flow, $\mathbf{p}_k = \mathbf{M}_k \mathbf{v}_k = -\nabla_{\mathbf{z}_k} \Phi_k$, for all $k \in \{1, \dots, K\}$. We now prove that each node follows natural gradient flow: Consider any non-leaf node $\mathtt{u}$. Let $\{\mathtt{v}_j\}_{j=1}^N$ be the child nodes of $\mathtt{u}$. Suppose each child node $\mathtt{v}_j$ follows natural gradient flow with potential $\Phi_{\mathtt{v}_j}$ and metric $\mathbf{M}_{\mathtt{v}_j}$. At node $\mathtt{u}$, by (2),

$$\mathbf{p}_\mathtt{u} = \textstyle\sum_{j=1}^{N} \mathbf{J}_{\mathtt{e}_j}^\top \mathbf{p}_{\mathtt{v}_j} = \sum_{j=1}^{N} \mathbf{J}_{\mathtt{e}_j}^\top \nabla_{\mathbf{y}_j} \Phi_{\mathtt{v}_j} = \nabla_\mathbf{x} \Phi_\mathtt{u}, \quad \text{where } \Phi_\mathtt{u} \coloneqq \sum_{j=1}^{N} \Phi_{\mathtt{v}_j} \circ \psi_{\mathtt{e}_j} \tag{4}$$

Therefore, by recursively applying the analysis from the leaf nodes to the root node, we have that the root node also follows natural gradient flow $\mathbf{p}_\mathtt{r} = \nabla_\mathbf{q} \Phi_\mathtt{r}$. Hence, we have,

$$\frac{d}{dt} \Phi_\mathtt{r} = \dot{\mathbf{q}}^\top \nabla_\mathbf{q} \Phi_\mathtt{r} = -\big(\nabla_\mathbf{q} \Phi_\mathtt{r}\big)^\top \mathbf{M}_\mathtt{r}^{-1} \nabla_\mathbf{q} \Phi_\mathtt{r} \tag{5}$$

Under the assumption $\mathbf{M}_\mathtt{r} \succ 0$, by LaSalle's invariance principle [27], the system converges to the forward invariant set $\mathcal{C}_\infty = \{\mathbf{q} : \nabla_\mathbf{q} \Phi_\mathtt{r} = 0\}$. □

The Benefit of Velocity-based Motion Control: The main benefit of our proposed velocity-based motion control framework, compared to RMPflow [1], is its simplicity. The forward pass and backward pass of RMPflow requires computing additional curvature terms $\dot{\mathbf{J}}_k \dot{\mathbf{q}}$ resulting from pushing forward accelerations. More importantly, to ensure the stability of the generated motion, it also requires the leaf policies to take a more complicated form, known as geometrical dynamical systems [1], which involves curvature terms of the importance weight matrices $\{\mathbf{M}_k\}_{k=1}^K$. This creates a huge amount of computational overhead for learning when the importance weight matrices are parameterized, and also makes the optimization problem more complicated.

IV. Learning Structured Motion Policies for Human Demonstrations

In this section, we provide details of our approach to learning motion policies (1) from human demonstrations.

A. Problem Statement

Consider the problem of kinesthetic teaching, where the human provide demonstrations by moving the robot, providing a number of trajectory demonstrations in the joint configuration space. Additionally, we allow the human to specify a number of *subtask spaces*[2], where the motion on the these subtask spaces is relevant to the achieving the overall task. For example, for a goal-reaching subtask, the user may specify, as a subtask space, the 3-dimensional Euclidean workspace of the end-effector position. The goal for our learning problem is to learn a parameterized policy π^θ which can generate motion similar to the human demonstrations when *viewed in these subtask spaces*.

Consider N trajectory demonstrations in the configuration space of the robot, each composed of T_i position-velocity pairs, denoted by $\{\{(\mathbf{q}_{i,t}^d, \dot{\mathbf{q}}_{i,t}^d)\}_{t=1}^{T_i}\}_{i=1}^N$. On possible way of learning is to directly regress joint space velocity so that it matches the joint velocity of the demonstrations, i.e.,

$$\theta_\mathcal{C}^\star = \arg\min_\theta \underbrace{\sum_{i=1}^{N} \sum_{t=1}^{T_i} \left\| \dot{\mathbf{q}}_{i,t}^d - \pi^\theta\big(\mathbf{q}_{i,t}^d\big) \right\|^2}_{\mathcal{L}_\mathcal{C}(\theta)}. \tag{6}$$

Presumably, if the learned joint velocity policy perfectly matches the demonstrated joint velocities, it should also perfectly matches the demonstrations in the subtask spaces. However, in practice, there usually does not exist a policy such that $\mathcal{L}_\mathcal{C}(\theta) = 0$. This is because, when providing demonstrations, the human primarily cares about the motion on spaces that are directly relevant to achieving the task (i.e. the subtask spaces). As a result, the demonstrations can be conflicting in the joint space (providing vastly different velocities at the same joint position), even though they can be consistent in the subtask spaces. Therefore, directly regressing on the joint space trajectories (6) will produce large error both in the joint space and in the workspace.

Given the observation that the trajectory demonstrations are the most informative when considered in the subtask spaces, we proposed an alternative objective function that direct penalize the deviation in the subtask spaces:

$$\begin{aligned} \theta^\star &= \arg\min_\theta \sum_{i=1}^{N} \sum_{t=1}^{T_i} \sum_{k=1}^{K} \lambda_k \left\| \mathbf{J}_k \dot{\mathbf{q}}_{i,t}^d - \mathbf{J}_k \pi^\theta(\mathbf{q}_{i,t}^d) \right\|^2 \\ &= \arg\min_\theta \underbrace{\sum_{i=1}^{N} \sum_{t=1}^{T_i} \sum_{k=1}^{K} \lambda_k \left\| \dot{\mathbf{q}}_{i,t}^d - \pi^\theta(\mathbf{q}_{i,t}^d) \right\|_{\mathbf{J}_k^\top \mathbf{J}_k}^2}_{\mathcal{L}(\theta)}, \end{aligned} \tag{7}$$

where $\lambda_k > 0$ is the user-specified weight for the k-th subtask. In contrast to (6), our proposed objective only penalizes the deviation of velocities in the subtask spaces. The velocities in each subtask spaces are given by the *pushforward* operator $\mathbf{q} \mapsto \mathbf{J}_k(\mathbf{q})$.

[2]In most existing learning from demonstrations literature [10], [16]–[20], only a single (subtask) space is considered, and it is usually either the configuration space, or the (3-d or 6-d) end-effector workspace.

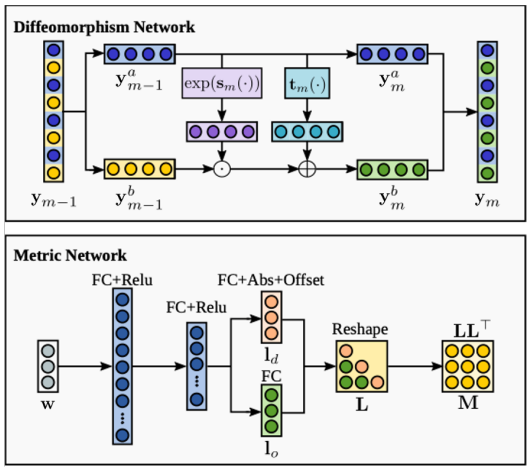


Fig. 3: *Top:* Structure of the network defining a single map ψ_m in the diffeomorphism chain [10]. *Bottom:* Structure of the network for defining latent subtask metric $\mathbf{M}_{\mathtt{d}_k}$ [9].

Let us now consider policy learning with the structured policy class introduced in Section III. As the subtask spaces are provided by the demonstrator, we can represent the joint space velocity as the solution to the motion generation problem described in Section III, where the subtask policies are parameterized, i.e.,

$$\pi^\theta(\mathbf{q}) = \arg\min_{\mathbf{u}} \sum_{k=1}^{K} \left\| \mathbf{v}_k^{\theta_k}(\psi_k(\mathbf{q})) - \mathbf{J}_k\,\mathbf{u} \right\|^2_{\mathbf{M}_k^{\theta_k}}. \tag{8}$$

We can then optimize for the objective function (7) through, e.g. gradient descent:

$$\begin{aligned} \theta &\leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\mathcal{C}) \\ &\leftarrow \theta - \alpha \sum_{i=1}^{N} \sum_{t=1}^{T_i} \frac{\partial \pi^\theta(\mathbf{q}^d_{i,t})}{\partial \theta} \nabla_{\pi^\theta(\mathbf{q}^d_{i,t})} \mathcal{L}(\theta), \end{aligned} \tag{9}$$

where the term $\frac{\partial \pi^\theta(\mathbf{q}^d_{i,t})}{\partial \theta}$ can be computed by back-propagating through the motion generation algorithm described in Section III-C.

Note that the policy parameterization π^θ in (8) also allows for certain subtask policies to be *fixed* during learning, i.e., $\theta_k = \emptyset$. This provides the user with the freedom to manually design subtask policies, for, e.g., joint damping, respecting joint limit, collision avoidance, etc. In this case, the parameterized subtask policies also learn to trade off against the hand-designed policies through the importance weight matrices. For the remainder of this section, we present an expressive class of stable learnable subtask policies which result in a stable configuration space velocity policy under Theorem III.1.

B. A Class of Stable Subtask Policies

We seek to paramterize subtask policies $\{(\mathbf{v}_k^{\theta_k}, \mathbf{M}_k^{\theta_k})\}_{k=1}^K$ so that the resulting configuration space policy is stable. According to Theorem III.1, the resulting motion is stable as long as the subtask policy follows natural gradient flow (3). Therefore, we choose to parameterize the subtask policy $(\mathbf{v}_k, \mathbf{M}_k)$ through the tuple $(\Phi_k^{\theta_k}, \mathbf{M}_k^{\theta_k})$, where Φ_k is the potential function. The velocity policy is then given by $\mathbf{v}_k^{\theta_k} = (\mathbf{M}_k^{\theta_k})^{-1}\nabla\Phi_k^{\theta_k}$. To ensure stability of the combined policy in the configuration space, we need the importance weight matrix $\mathbf{M}_k^{\theta_k}$ to be always positive definite. Additionally, we require Φ_k to have a unique minimum at a desired goal location $\mathbf{z}_k^*$ as we care primarily about goal-directed motions.

1) From subtasks to latent subtasks: The main challenge for representing the subtask policy is finding a expressive parameterization of the potential function without introducing spurious attractive points. While direct parameterization with such property is in general challenging, recent work [10] has demonstrate success through parameterizing diffeomorphisms to *latent* spaces, where a simple potential function is defined there. We adopt this approach as it shows expressiveness in representing complex motions without introducing undesired local minima.

Conveniently, in our transform tree formulation (Section III-C), this is equivalent to adding a child node, $\mathtt{d}_k$, to the original "leaf" node $\mathtt{l}_k$ (see Fig.2). The map between $\mathtt{d}_k$ and $\mathtt{l}_k$ is a learnable map, $\phi_k^{\theta_k} : \mathbf{z}_k \mapsto \mathbf{w}_k$.

Then, in the latent space, we can use a simple pre-specified potential function, e.g. $\Phi_{\mathtt{d}_k}(\mathbf{w}_k) = 0.5\|\mathbf{w}_k - \phi_k^{\theta_k}(\mathbf{z}_k^*)\|^2$, and any positive-definite parameterization of the importance weight matrix[3] $\mathbf{M}_{\mathtt{d}_k}^{\theta^k}$. Then, by properties of diffeomorphisms [10], the generated motion is guaranteed stable. Next, we introduce the parameterizations we choose for the diffeomorphism and importance weight matrix, respectively.

2) Diffeomorphisms parameterized by flow networks: To realize a diffeomorphism, we rely on the formulation in [10] (see Fig.3). Specifically, we view $\phi_k^{\theta_k}$ as a chain of M simpler maps, i.e. $\phi_k^{\theta_k} = \psi_1 \circ \cdots \circ \psi_M$. Assuming coordinates $\mathbf{y}_m \in \mathbb{R}^n$ for the co-domain of ψ_m i.e. $\mathbf{y}_m = \psi_m(\mathbf{y}_{m-1})$, $\mathbf{y}_0 = \mathbf{z}_k$, and $\mathbf{y}_M = \mathbf{w}_k$, we define,

$$\mathbf{y}_m = \begin{bmatrix} \mathbf{y}^a_m \\ \mathbf{y}^b_m \end{bmatrix} = \begin{bmatrix} \mathbf{y}^a_{m-1} \\ \mathbf{y}^b_{m-1} \odot \exp\left(s_m(\mathbf{y}^a_{m-1})\right) + t_m(\mathbf{y}^a_{m-1}) \end{bmatrix}, \tag{10}$$

where $\odot$ and exp denote pointwise product and exponential respectively. $s_m : \mathbb{R}^{\lfloor n/2 \rfloor} \to \mathbb{R}^{\lceil n/2 \rceil}$ and $t_m : \mathbb{R}^{\lfloor n/2 \rfloor} \to \mathbb{R}^{\lceil n/2 \rceil}$ are learnable scaling and translation functions. The components $\mathbf{y}^a_{m-1} \in \mathbb{R}^{\lfloor n/2 \rfloor}$ and $\mathbf{y}^b_{m-1} \in \mathbb{R}^{\lceil n/2 \rceil}$ constitute alternate input dimensions, with the pattern of alternation reversed after each mapping in the chain. We parameterize the scaling and translation functions as linear combinations of random Fourier features (10) i.e. $s_m(\cdot) \coloneqq s_m(\cdot\,; \theta_{s_m}) = \varphi(\cdot)^\top \theta_{s_m}$, and $t_m(\cdot) \coloneqq t_m(\cdot\,; \theta_{t_m}) = \varphi(\cdot)^\top \theta_{t_m}$. The feature

$$\varphi(\cdot) = \sqrt{\frac{2}{D}} \left[\cos(\boldsymbol{\alpha}_1^\top(\cdot) + \boldsymbol{\beta}_1), \ldots, \cos(\boldsymbol{\alpha}_D^\top(\cdot) + \boldsymbol{\beta}_D) \right]^\top \otimes \mathbf{I}, \tag{11}$$

is a D-dimensional Fourier feature approximation of a matrix-valued Gaussian separable kernel [28], [29], $K(\mathbf{y}, \mathbf{y}') = \exp(-\frac{\|\mathbf{y}-\mathbf{y}'\|^2}{2l^2})\mathbf{I}$ with length-scale l. Due to the choice of parameterization in (10)-(11), ψ_m is a smooth and affine bijective map, and thus a diffeomorphism. Consequently, the chain $\phi_k^{\theta_k}$ is a diffeormorphism.

[3]One can also choose to parameterize the matrix in the subtask space instead of the latent space.

3) Importance weight matrix via Cholesky decomposition: Similar to [9], we represent a latent subtask inertia matrix $\mathbf{M}_{\mathtt{d}_k}$ by its Cholesky decomposition parameterized by a matrix-valued neural network (see Fig.3). This parameterization has been previously introduced in [9]. Concretely, we construct $\mathbf{M}_{\mathtt{d}_k} \coloneqq \mathbf{L}_{\mathtt{d}_k}\mathbf{L}_{\mathtt{d}_k}^{\top}$, where $\mathbf{L}_{\mathtt{d}_k}(\mathbf{w}_k) \in \mathbb{R}^{n\times n}$ is a lower-triangular matrix. We parameterize the vectorized diagonal and off-diagonal entries of $\mathbf{L}_{\mathtt{d}_k}$, i.e. $\mathbf{l}_d(\mathbf{w}_k;\theta_{\mathbf{l}_d}) \in \mathbb{R}^n$ and $\mathbf{l}_o(\mathbf{w}_k;\theta_{\mathbf{l}_o}) \in \mathbb{R}^{\frac{1}{2}(n^2-n)}$, as fully-connected neural networks with RELU activations. Furthermore, the networks for $\mathbf{l}_o$ and $\mathbf{l}_d$ share parameters for all the layers except their output layers. To ensure $\mathbf{L}_{\mathtt{d}_k}$ is a valid Cholesky decomposition, and consequently $\mathbf{M}_{\mathtt{d}_k}$ is positive definite, we require the entries of $\mathbf{l}_d$ to be strictly positive. In lieu of this, we take the absolute value of the output linear layer of $\mathbf{l}_d$ and add a small positive bias $\epsilon > 0$.

C. Discussion

At first glance, our approach may seems very similar to [9] as we both learn multiple subtask policies and, during execution, combine them together. However, mathematically, they are fundamentally different.

In [9], each subtask policy is independently learned to imitate the demonstrated trajectories mapped the the corresponding space, i.e.,

$$\theta_k^{\star} = \arg\min_{\theta_k} \sum_{i=1}^{N}\sum_{t=1}^{T_i} \left\| \mathbf{J}_k\dot{\mathbf{q}}_{i,t}^{d} - (\mathbf{M}_k^{\theta_k})^{-1}\nabla\Phi_k^{\theta_k} \right\|^2. \quad (12)$$

The combination of these individually-learned policy only happens after learning, i.e., during the execution of the policy. This strategy has the following limitations. First, the learned importance weight matrices are not learned to provide the proper trade-offs between policies, as each policy is learned independently. Due to this, additional manual scaling of the importance weight matrices is needed especially when combined with hand-designed policies. Second, the geometric constraints (e.g. induced by the kinematics of the robot) between tasks are not considered during learning, which contributions to errors during execution. In summary, despite its empirical success, the approach introduced in [9] is not able to fully exploit the benefits provided by the policy structure. Whereas our approach, by properly formulating the learning problem (7) and differentiable through the structured policy (8), is able to take full advantage over the policy structure during learning. We will further demonstrate this through experiments in upcoming section.

V. Experimental Results

We evaluated our approach on three manipulations tasks[4] on a 7-DOF Rethink Sawyer robot with configuration space coordinates $\mathbf{q} \in \mathbb{R}^7$. We consider 3 tasks including *inspection*, *placing-1*, and *placing-2*. For details about the task specifications, the reader is referred to Figs. 6–8. For each task, a human subject provided multiple configuration space demonstrations via kinesthetic teaching: 14 demonstrations for *inspection*, 9 for *placing-1*, and 12 for *placing-2*.

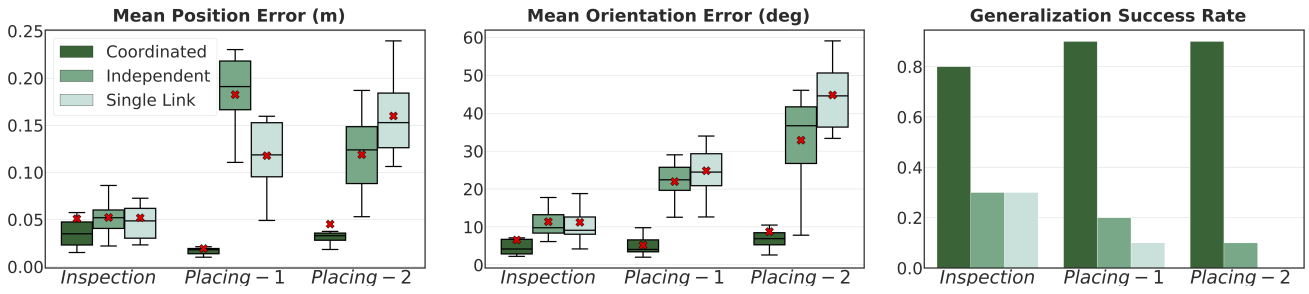
Subtasks: Each of the tasks is decomposed into learnable subtasks assigned to 3 robot body parts, whereby each body part is represented by a unique *control point* (see Fig. 5 for details). Given our choice of control points, the subtask policies effectively control the end-effector pose (i.e. position and orientation). However, we stress that our learning approach is not only limited to learning policies for robot poses. In fact, one may instead, for instance, choose to learn motion policies dictating a partial pose (by removing a control point), or pose alongside the robot elbow (by adding an additional control point). Furthermore, there is a hand-specified default subtask policy pulling the end-effector in straight-line towards a desired goal pose. It is governed by a convex potential and a constant inertia matrix $\mathbf{M} = 10\mathbf{I}$. Additionally, to ensure the root importance weight matrix $\mathbf{M}_{\mathtt{r}}$ is always non-singular and well-conditioned, we add a small offset $\epsilon_{\mathtt{r}} = 0.02$ to its diagonal entries.

Baselines: To evaluate the performance of our approach, we establish two baselines: *(i)* an *independent* learning version whereby the subtask policies are learned independently, which reproduces the setup of [9], and *(ii)* a *single link* learning version where just a single control point (i.e. end-effector) is chosen and the associated subtask policy is again learned independently.

Learning Details: As is introduced in Section IV, the subtask policies are defined by a set of diffeomorphisms $\{\phi_k^{\theta_k}\}_{k=1}^3$ and a set of latent importance weight matrices $\{\mathbf{M}_{\mathtt{d}_k}^{\theta_k}\}_{k=1}^3$. In our parameterization, each diffeomorphism is composed of $M = 10$ chained diffeomorphisms, each parameterized by $D = 200$ random Fourier features with length-scale $l = 0.45$. On the other hand, each latent importance weight matrix $\mathbf{M}_{\mathtt{d}_k}$ has two hidden layers with 128 and 64 dimensions respectively. The optimization problem in (7) was solved with Adam optimizer [30] with a learning rate of 1×10^{-4} and weight decay 1×10^{-8}.

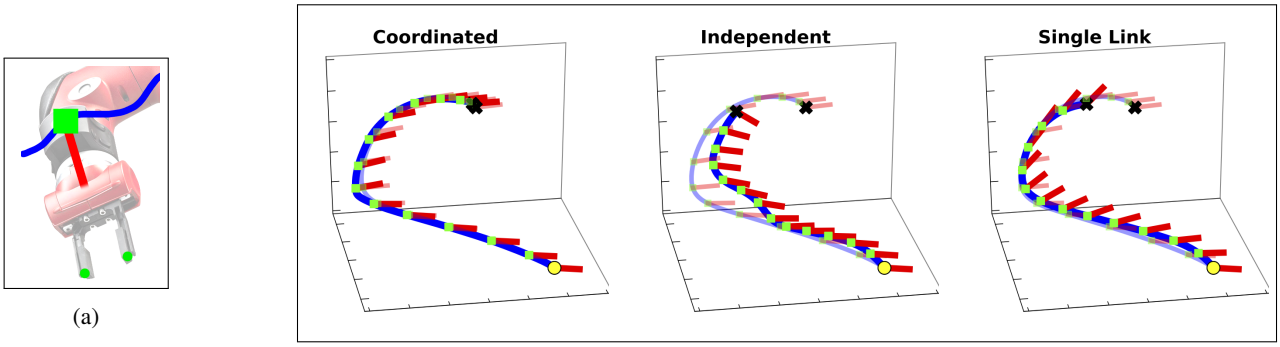
Results Fig. 5 shows example reproductions of end-effector pose trajectories under the aforementioned variants of our algorithm. Our coordinated learning approach is observed to successfully reproduce the demonstrated motions. However, the baselines either fail to reproduce the position profile or the orientations. To quantitatively evaluate the capacity of our approach to reproduce demonstrations, we employ two error metrics, mean position error and mean orientation error. We evaluate position errors in terms of the Euclidean distance i.e. $error(\boldsymbol{p}_1(t), \boldsymbol{p}_2(t)) = \|\boldsymbol{p}_1(t) - \boldsymbol{p}_2(t)\|_2$, where $\mathbf{p}_1(t)$ and $\mathbf{p}_2(t)$ are end-effector positions on the demonstrated and reproduced trajectory at time stamp t, respectively. On the other hand, for orientation errors we employ $error(\boldsymbol{o}_1(t), \boldsymbol{o}_2(t)) = \arccos(|\boldsymbol{o}_1(t) \cdot \boldsymbol{o}_2(t)|)$, where $\boldsymbol{o}_1(t)$ and $\boldsymbol{o}_2(t)$ are unit quaternions representing end-effector orientations. For each comparison metric, we take the mean of the errors accumulated over the duration of a trajectory. Fig. 4 reports these comparisons as box plots. For the two *placing* tasks, our approach outperforms the baselines by a significant margin. A major contributor

[4]Accompanying video is available at: `https://youtu.be/hwcxzLnxZPQ`.



(a) (b) (c)

Fig. 4: Comparison of our approach against baselines based on (a) mean position error, (b) mean orientation error, and (c) generalization success rate over 10 executions.



(a) (b)

Fig. 5: (a) Visualization of the 3 control points (in green), with the end-effector control point denoted by a square while the two control points for gripper tips are given by dots. Overlaid is an end-effector position trajectory (in blue), and a line directed from the end-effector to the center of the gripper (in red) denoting instantaneous end-effector orientation. (b) Plots showing pose trajectories starting from an initial end-effector pose (yellow circle) governed by our approach (Coordinated) and baselines (Independent [9] and Single Link). Also shown in the background, is the demonstration starting from the same initial pose. The final positions are denoted by black crosses.

towards this difference in performance is the existence of default subtask policies. When learned without accounting for the existing policies, the learned policies may not be able to sufficiently bias against the default behavior. Furthermore, we also observe that the independent learning version occasionally performs worse than the single link learning variant. This is perhaps because the independently learned subtask policies may conflict with each other. This does not manifest as much in the single link case, since there is only one learned subtask policy.

Lastly, we test the generalization performance of our approach. For this evaluation, we roll out our motion policy from 10 new initial configurations. A rollout is considered successful if all the goals of the task are met without any collisions. Fig. 4 (c) reports the success rates. Once again, our end-to-end learning approach outperforms the baselines. We also observe that, while the difference in terms of quantitative errors between our approach and the baselines is small on the *inspection* task, there are vast differences in performances given by generalization success rates. This is perhaps because, even when not trained end-to-end, the robot's kinematic constraints may enforce certain level of coordination between subtasks, thus resulting in low reproduction errors. However, for highly constrained tasks like the ones we explore in this paper, even small errors can result in task execution failures. A subset of rollouts from our learned policies, starting from the same configurations as demonstrations, are visualized in Figs. 6–8 (*bottom*).

References

[1] 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.

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

[3] C. A. Cheng, *Efficient and principled robot learning: theory and algorithms*. PhD thesis, Georgia Institute of Technology, 2020.

[4] 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," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1864–1871, 2018.

[5] A. Li, C.-A. Cheng, B. Boots, and M. Egerstedt, "Stable, concurrent controller composition for multi-objective robotic tasks," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 1144–1151, IEEE, 2019.

[6] A. Li, M. Mukadam, M. Egerstedt, and B. Boots, "Multi-objective policy generation for multi-robot systems using Riemannian motion policies," in *International Symposium on Robotics Research*, 2019.

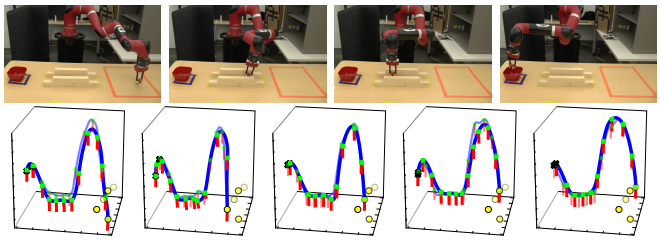


Fig. 6: The *inspection* task required the robot to pick an object from one side of the table and place it in a bowl on the other side. In the middle, the robot was required to pass a constrained pathway. *Top:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlaid on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts.

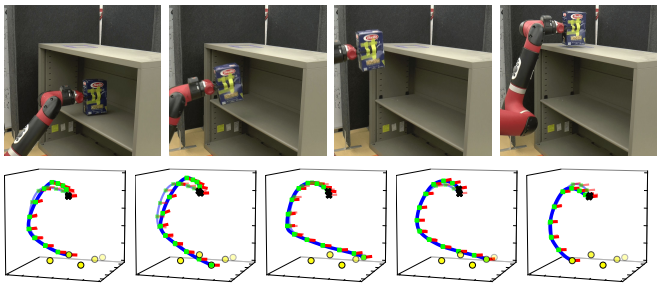


Fig. 7: The *placing-1* task required the robot pick an object from a lower shelf and place it on at a goal location on the top-most shelf at a certain orientation. *Top:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlaid on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts.

[7] X. Meng, N. Ratliff, Y. Xiang, and D. Fox, "Neural autonomous navigation with riemannian motion policy," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8860–8866, IEEE, 2019.

[8] B. Wingo, C. Cheng, M. Murtaza, M. Zafar, and S. Hutchinson, "Extending riemmanian motion policies to a class of underactuated wheeled-inverted-pendulum robots," in *IEEE International Conference on Robotics and Automation*, 2020.

[9] M. A. Rana, A. Li, H. Ravichandar, M. Mukadam, S. Chernova, D. Fox, B. Boots, and N. Ratliff, "Learning reactive motion policies in multiple task spaces from human demonstrations," in *Conference on Robot Learning*, pp. 1457–1468, 2020.

[10] M. A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, and N. Ratliff, "Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems," 2020.

[11] 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.

[12] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[13] A. Dietrich, A. Albu-Schäffer, and G. Hirzinger, "On continuous null space projections for torque-based, hierarchical, multi-objective manipulation," in *IEEE International Conference on Robotics and Automation*, pp. 2978–2985, IEEE, 2012.

[14] J. Lee, N. Mansard, and J. Park, "Intermediate desired value approach for task transition of robots in kinematic control," *IEEE Transactions on Robotics*, vol. 28, no. 6, pp. 1260–1277, 2012.

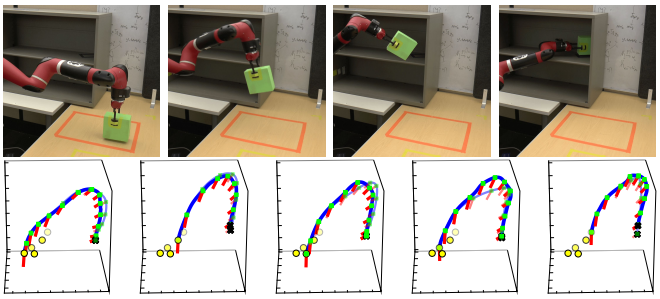


Fig. 8: The *placing-2* task required the robot pick an object from a table, significantly rotate its end-effector, and place the object on a shelf. *Top:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlaid on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts. Note that the viewing angle in the plots is different from that in the robot execution snapshots.

[15] A. Dietrich, C. Ott, and J. Park, "The hierarchical operational space formulation: stability analysis for the regulation case," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1120–1127, 2018.

[16] A. Paraschos, R. Lioutikov, J. Peters, and G. Neumann, "Probabilistic prioritization of movement primitives," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2294–2301, 2017.

[17] 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)*, pp. 763–768, IEEE, 2009.

[18] 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.

[19] 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.

[20] H. chaandar Ravichandar and A. Dani, "Learning position and orientation dynamics from demonstrations via contraction analysis," *Autonomous Robots*, vol. 43, no. 4, pp. 897–912, 2019.

[21] 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.

[22] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of statistical planning and inference*, vol. 90, no. 2, pp. 227–244, 2000.

[23] M. Mukadam, C.-A. Cheng, D. Fox, B. Boots, and N. Ratliff, "Riemannian motion policy fusion through learnable lyapunov function reshaping," in *Conference on Robot Learning*, pp. 204–219, 2020.

[24] E. Aljalbout, J. Chen, K. Ritt, M. Ulmer, and S. Haddadin, "Learning vision-based reactive policies for obstacle avoidance," *arXiv preprint arXiv:2010.16298*, 2020.

[25] J. J. Craig, *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India, 2009.

[26] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[27] H. K. Khalil and J. W. Grizzle, *Nonlinear systems*, vol. 3. Prentice hall Upper Saddle River, NJ, 2002.

[28] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, "Kernels for vector-valued functions: A review," *arXiv preprint arXiv:1106.6251*, 2011.

[29] V. Sindhwani, M. H. Quang, and A. C. Lozano, "Scalable matrix-valued kernel learning for high-dimensional nonlinear multivariate regression and granger causality," *arXiv preprint arXiv:1210.4792*, 2012.

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