2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Differentiable Motion Manifold Primitives for Reactive Motion Generation under Kinodynamic Constraints

Yonghyeon Lee

***Abstract*— Real-time motion generation – which is essential for achieving reactive and adaptive behavior – under kinodynamic constraints for high-dimensional systems is a crucial yet challenging problem. We address this with a two-step approach: *offline* learning of a lower-dimensional trajectory manifold of task-relevant, constraint-satisfying trajectories, followed by rapid *online* search within this manifold. Extending the discrete-time Motion Manifold Primitives (MMP) framework, we propose *Differentiable Motion Manifold Primitives (DMMP)*, a novel neural network architecture that encodes and generates continuous-time, differentiable trajectories, trained using data collected offline through trajectory optimizations, with a strategy that ensures constraint satisfaction – absent in existing methods. Experiments on dynamic throwing with a 7-DoF robot arm demonstrate that DMMP outperforms prior methods in planning speed, task success, and constraint satisfaction. Project page: https://diffmmp.github.io/.**

I. Introduction

Motion Manifold Primitives (MMP) is a generative movement primitive framework based on an autoencoder architecture [1], [2], [3]. It encodes a diverse set of trajectories – collected primarily from human demonstrations – into a lower-dimensional manifold *offline*, thereby enabling *online* real-time motion generation and supporting reactive, adaptive behavior in dynamically changing environments [4], [5], [6], [7]. In this paper, our main objective is to extend this framework to motion generation under stringent kinodynamic constraints, where the generated motions are required to push against the limits of these constraints to successfully accomplish the task (e.g., see Fig. 1).

Throughout, we assume that the objective function and constraints are given – unlike previous approaches that rely on demonstration data – so, in principle, a solution could be obtained by solving a trajectory optimization problem. However, we focus on problems whose solutions cannot be computed in real time – specifically, not within sub-second latency – due to their complexity and the high dimensionality of the system, despite many advances in sampling, search, and optimization techniques [8], [9], [10], [11], [12], [13], [14], [15].

A natural approach is to adapt the MMP framework to our setting by collecting demonstration data through offline trajectory optimization. However, existing MMP methods do not explicitly incorporate kinodynamic constraints during training, which often results in trajectories that substantially violate these constraints.

Y. Lee is with the Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: yhl@mit.edu).

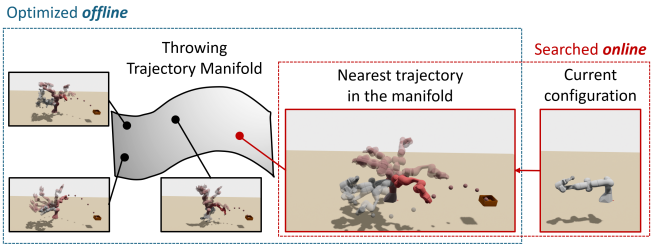

Fig. 1. Throwing trajectory manifold – which consists of task-completing trajectories that satisfy the kinodynamic constraints – is optimized *offline*, and given a current configuration, the nearest throwing trajectory can be quickly searched *online* within the manifold. The moment of the throw is depicted in deep red, preceded by gray and followed by transparent red.

To train an MMP model that satisfies the imposed constraints, we present two key contributions. The first is *Differentiable Motion Manifold Primitives (DMMP)*, a novel neural network architecture designed to encode and generate a manifold of continuous-time trajectories that are differentiable in time. Unlike prior discrete-time MMP formulations, this differentiability enables us to naturally enforce constraint satisfaction during training by incorporating the constraints directly into the loss function.

The second contribution is a practical four-step training strategy for DMMP (see Fig. 2; details are provided in Section II). First, we collect multiple trajectories for each task parameter by solving trajectory optimization problems with diverse random initializations and seeds. Second, we fit these trajectories to a differentiable motion manifold. Third, we train a task-conditioned latent flow model, following [7]. Finally, we freeze the latent flow and fine-tune the manifold to ensure that the generated trajectories both accomplish the tasks and satisfy the kinodynamic constraints.

We present a case study on a dynamic throwing task using a 7-DoF robot arm, which serves as a representative example of challenging kinodynamic motion-planning problems. To throw beyond the nominal workspace limits, the robot must adopt a preparatory posture – such as pulling its arm back – and fully exploit its velocity limits. This requires optimizing the entire joint trajectory and the throwing time, rather than only the throwing phase, while simultaneously satisfying kinodynamic constraints, thereby making the problem highly complex.

We compare our DMMP with conventional trajectory optimization methods [16], [17], [18], as well as existing MMPs [5], [7], evaluating task success rates, constraint satisfaction rates, and planning times. Our findings demonstrate that our method generates trajectories much more quickly than traditional trajectory optimization, with significantly

higher success and constraint satisfaction rates compared to existing MMPs.

II. Differentiable Motion Manifold Primitives

We begin with assumptions, notations, and problem definitions. Let the configuration be $q \in Q$ and the task parameter $\tau \in \mathcal{T}$ (e.g., the target box position for a throwing task). For a fixed terminal time T, a smooth trajectory is denoted by $q(t)$ for $t \in [0, T]$, and an additional variable η may be needed to fully specify a motion – for example, in a throwing task, the release time $\eta \in (0, T)$ determines when the object is released at $q(\eta) \in Q$. The task-dependent objective function is $J(q(\cdot), \eta; \tau)$, and kinodynamic constraints such as self-collisions, joint limits, and bounds on velocity, acceleration, jerk, and torque are expressed as $C(q, \dot{q}, \ddot{q}, \dddot{q}) \in \mathbb{R}^k \leq 0$ with C differentiable. We assume that, for each τ, multiple globally or locally optimal motions $(q(t), \eta)$ exist. Our goal is to learn a model, given any $\tau \in \mathcal{T}$, generates multiple feasible motions $(q(t), \eta)$ that satisfy the constraints and optimize the objective.

Our framework consists of four steps. First, we collect multiple optimal trajectories by solving trajectory optimizations for each τ in section II-A. Using these trajectories, we fit a differentiable motion manifold in section II-B. In section II-C, we train a task-conditioned motion distribution in the latent space. Lastly, in section II-D, we fine-tune the motion manifold. The overall pipeline is visualized in Fig. 2.

A. Data Collection via Trajectory Optimizations

We collect multiple trajectories for each τ to enable subsequent manifold learning. Since sampling all $\tau \in \mathcal{T}$ is infeasible, we select a finite subset $\mathcal{T}_s = \{\tau_i\}_{i=1}^M$ that approximates $\mathcal{T}$. For each τ_i, we solve

$$\min_{q(t),\eta} J(q(\cdot), \eta; \tau_i) \quad \text{s.t. } C(q, \dot{q}, \ddot{q}, \dddot{q}) \leq 0, \ \forall t \in [0, T]. \quad (1)$$

Specifically, we parameterize the trajectory as

$$q(t) = q_0 + (q_T - q_0)(3 - 2s)s^2 + s^2(s-1)^2\Phi(s)w, \quad (2)$$

where $s = \frac{t}{T}$ and $q_0, q_T \in Q \subset \mathbb{R}^n$. The basis matrix is $\Phi(s) = [\phi_1(s), \ldots, \phi_B(s)] \in \mathbb{R}^{1\times B}$, where $\phi_i(s) = \exp(-B^2(s - \frac{i-1}{B-1})^2)$ and B is the number of basis functions. The coefficient matrix $w \in \mathbb{R}^{B\times n}$. This parameterization satisfies the boundary conditions $q(0) = q_0$, $q(T) = q_T$, and $\dot{q}(0) = \dot{q}(T) = 0$. We optimize over (q_0, q_T, w) with diverse random seeds and initializations to collect diverse solutions $\{(q_{ij}(t), \eta_{ij})\}_{j=1}^{N_i}$.

B. Learning Differentiable Motion Manifold

In this section, we propose a method to train a manifold of continuous-time, differentiable trajectories from the dataset $\{(q_{ij}(t), \eta_{ij})\}_{j=1}^{N_i}$. Following autoencoder-based manifold learning approaches [19], [20], [21], [22] and discrete-time MMPs [5], [7], we introduce an encoder g and decoder f with latent space $Z = \mathbb{R}^m$. The encoder maps a discretized trajectory $(q_1, \ldots, q_L) \in Q^L$ and η to a latent variable $z \in Z$, i.e., $g((q_1, \ldots, q_L), \eta) = z$. Unlike discrete MMP decoders that take only z and output a vectorized trajectory $(\hat{q}_1, \ldots, \hat{q}_L) \in Q^L$, our decoder takes (z, t) and outputs $\hat{q}(z, t) \in Q$ and $\eta(z)$, with η depending solely on z. Because $f(z, t)$ is differentiable in t, we refer to it as a *differentiable decoder* (see Fig. 3).

The encoder and decoder are approximated with deep neural networks, denoted by g_α and $f_\beta = (\hat{q}_\beta, \eta_\beta)$, respectively, with parameters α, β, and trained to minimize the following loss function:

$$\mathcal{L}_{\text{recon}}(\alpha, \beta) := \frac{1}{M}\sum_i \frac{1}{N_i}\sum_j \|\hat{q}_\beta(z_{ij}, t) - q_{ij}(t)\|^2_{c(t)} + \|\eta_\beta(z_{ij}) - \eta_{ij}\|^2, \quad (3)$$

where $z_{ij} = g_\alpha((q_{ij}(t_1), \ldots, q_{ij}(t_L)), \eta_{ij})$, $t_l = \frac{l-1}{L-1}T$ for $l = 1, \ldots, L$, and $\|\delta(t)\|^2_{c(t)} := \int_0^T c(t)\delta^T(t)\delta(t)dt$ for some positive function $c(t) > 0$. The positive function $c(t)$ is introduced to fit more accurately on important parts along the time axis. For example, in the throwing task, the time around the throwing moment η should be given more weight.

Any neural network can be used for f_β. Inspired by deep operator learning [23], we adopt a linear basis function architecture for $\hat{q}_\beta$ that improves memory and computation efficiency:

$$\hat{q}_\beta(z, t) = \sum_{b=1}^{N_b} \psi^b_\beta(z)\, \theta^b_\beta(t), \quad (4)$$

where $\psi^b_\beta : Z \to \mathbb{R}$ and $\theta^b_\beta : \mathbb{R} \to \mathbb{R}^n$ for $b = 1, \ldots, N_b$ are neural networks. This structure avoids differentiating ψ when computing time derivatives, reducing cost, and requires storing only $\psi(z)$ – not the full network – when inferring a trajectory for a fixed z, improving memory efficiency.

Given a sufficiently low-dimensional latent space Z and the injective immersion conditions on $\hat{q}_\beta \colon z \mapsto \hat{q}_\beta(z, \cdot)$, we can interpret the set of trajectories $\{\hat{q}_\beta(z, \cdot) \mid z \in Z\}$ as an m-dimensional differentiable manifold of trajectories [19], [24], [25], [6], [21], [26]. Although these conditions may not be strictly satisfied everywhere, they hold almost everywhere due to the smoothness properties of neural networks. Following established terminology [5], [6], [7], we adopt the term *Differentiable Motion Manifold (DMM)* to denote this structure, as it captures the intrinsic low-dimensional manifold underlying the set of trajectories encoded by the model.

C. Latent Flow Learning

This section describes how to fit a task-conditioned density model $p(z|\tau)$ in the latent space of the learned differentiable motion manifold, adopting latent flow learning from MMFP [7]. Let z_{ij} be the latent encoding of $(q_{ij}(t), \eta_{ij})$ obtained from the trained encoder g_α, giving a dataset of task–latent pairs $\left(\tau_i, \{z_{ij}\}_{j=1}^{N_i}\right)_{i=1}^M$. Our goal is to train $p(z|\tau)$ so that, given a task τ, we can sample z and generate a motion via the decoder f_β, i.e., $z \mapsto f_\beta(z, \cdot)$.

Because $p(z|\tau)$ may be multimodal and highly nonconvex, we employ a flow-based generative model [27], [28] for sufficient expressiveness. In latent space Z, we define a

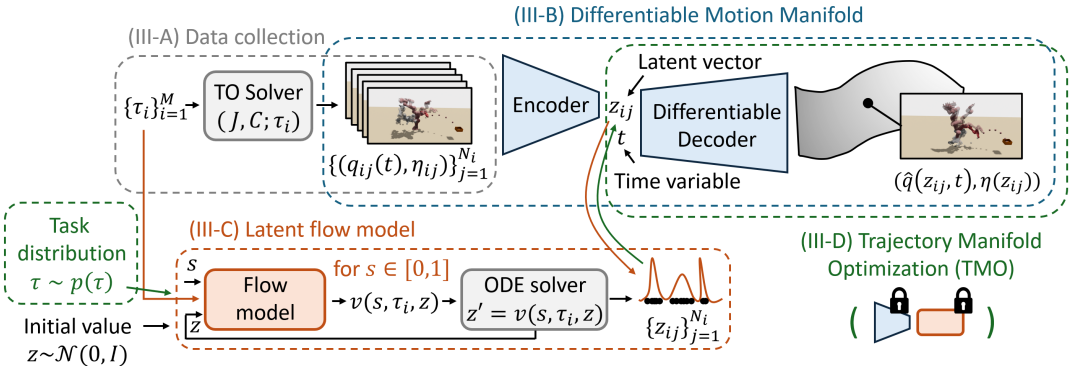

Fig. 2. Illustration of the overall training pipeline for the trajectory manifold. First, given a finite set of task parameters, we collect multiple, diverse trajectories for each task parameter using any existing Trajectory Optimization (TO) method. Second, we fit a Differentiable Motion Manifold (DMM) consisting of an encoder and a differentiable decoder – by differentiable, we mean differentiable in time. Third, we learn a flow-based, task-conditioned distribution in the latent space. Lastly, we fine-tune the trajectory manifold by optimizing the decoder – where we freeze the pre-trained encoder and latent flow model – to ensure that the generated trajectories achieve the tasks and comply with kinodynamic constraints for all task parameters randomly sampled from the task distribution.

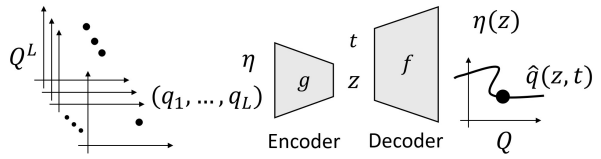

Fig. 3. The encoder takes $(q_1, \ldots, q_L) \in Q^L$ and η and outputs the latent value z; the decoder maps z and time variable t to a configuration $\hat{q}(z,t) \in Q$ and $\eta(z)$.

neural velocity field $v_\gamma(s, \tau, z)$ with parameters γ and evolve it by

$$\frac{dz}{ds} = v_\gamma(s, \tau, z), \quad s \in [0, 1], \tag{5}$$

treating $p_\gamma(z|\tau)$ as the pushforward of a Gaussian prior $p_0(z)$. Sampling from $p_\gamma(z|\tau)$ involves drawing $z_0 \sim p_0$ and integrating the ODE from $s = 0$ to $s = 1$. We train v_γ using flow matching [28], a simulation-free method that is more efficient than maximum-likelihood training.

Finally, by sampling $z \sim p_\gamma(z|\tau)$ and decoding with f_β, we generate motions for any given τ. We refer to this framework as *Differentiable Motion Manifold Flow Primitives (DMMFP)*.

D. Trajectory Manifold Optimization

Trajectories generated by DMMFP are not guaranteed to satisfy kinodynamic constraints. Moreover, since the dataset covers only a finite set of task parameters $\{\tau_i\}_{i=1}^M$, performance on unseen $\tau \in \mathcal{T}$ is not guaranteed. To address this, we propose *Trajectory Manifold Optimization (TMO)* to fine-tune the manifold so that generated trajectories both achieve the task for all $\tau \in \mathcal{T}$ and satisfy the constraints.

Let g_α, f_β, and p_γ be the pre-trained encoder, decoder, and latent flow. We fine-tune only the decoder parameters β, keeping α and γ fixed. Jointly tuning all parameters would require backpropagation through the sampled latent values and the ODE solver, making training computationally prohibitive, whereas adjusting β alone is both efficient and sufficient to modify the manifold.

Define $U(S)$ as the uniform distribution over a set S. We introduce the task loss

$$\mathcal{L}_{\text{task}}(\beta) := \mathbb{E}_{t,\tau,z}\Big[J(\hat{q}_\beta(z,\cdot), \eta_\beta(z); \tau) + W^T\big(\text{ReLU}(C(t,z,\beta))^2\big)\Big], \tag{6}$$

where $t \sim U([0,T])$, $\tau \sim U(\mathcal{T})$, and $z \sim p_\gamma(z|\tau)$. Here,

$$C(t,z,\beta) = C\big(\hat{q}_\beta, \tfrac{\partial}{\partial t}\hat{q}_\beta, \tfrac{\partial^2}{\partial t^2}\hat{q}_\beta, \tfrac{\partial^3}{\partial t^3}\hat{q}_\beta\big) \in \mathbb{R}^k, \tag{7}$$

and $W \in \mathbb{R}^k$ is a positive weight vector. The first term in (6) enforces task success, while the second penalizes constraint violations. Sampling $\tau \sim U(\mathcal{T})$ improves generalization, as the model is no longer limited to the subset $\mathcal{T}_s$ previously used for training.

Finally, adding the reconstruction loss (3) with fixed α, we minimize

$$\mathcal{L}(\beta) = w_{\text{recon}}\mathcal{L}_{\text{recon}}(\beta) + \mathcal{L}_{\text{task}}(\beta), \tag{8}$$

where $w_{\text{recon}} > 0$. We refer to this fine-tuning process as *Trajectory Manifold Optimization (TMO)*.

III. Case Study: Dynamic Throwing with a 7-DoF Robot Arm

In this section, we demonstrate our method on a dynamic throwing task with a 7-DoF Franka Panda arm, a challenging scenario that requires solving a kinodynamic optimization with nonlinear objectives and tight constraints. The solution depends heavily on initial conditions and is computationally expensive, making it ideal to showcase the benefits of our approach.

Task setup. As shown in Fig. 4, the task parameter is the target box position $\tau = (r\cos\theta, r\sin\theta, h) \in \mathbb{R}^3$ from the robot base. We exploit rotational symmetry by fixing $\theta = 0$, since nonzero θ can be handled by rotating the first joint. Therefore, the task parameter space for training models is

$$\mathcal{T} = \{(r, 0, h) \mid r \in [1.1, 2.0],\ h \in [0.0, 0.3]\},$$

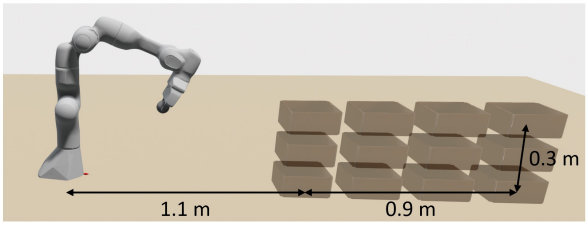

Fig. 4. Task parameter space $\mathcal{T}$, a set of positions of the target box.

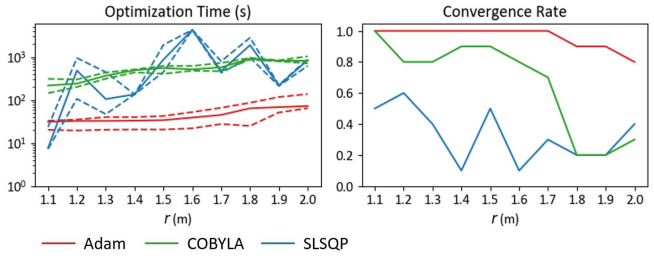

Fig. 5. Quartiles of the optimization times and convergence rates as functions of $r \in [1.1, 2.0]$, where the target box position is $\tau = (r, 0, 0)$. The quartiles are computed using only the successful cases from 10 optimization attempts.

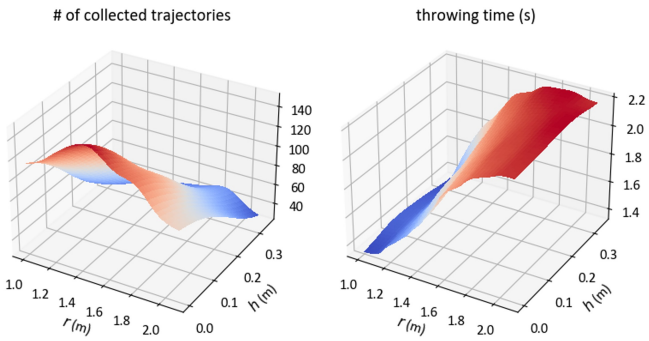

Fig. 6. The number of collected trajectories and throwing times as functions of r and h, where the target box position is $\tau = (r, 0, h)$.

excluding close targets where throwing is unnecessary. The throwing time $\eta \in [0, T]$ with $T = 5$ is an additional variable, and the release phase is $(q(\eta), \dot{q}(\eta))$. We assume the object is released instantaneously at time η, with the velocity determined by the end-effector at that instant.

Objective. Using free-fall dynamics, the landing position is computed and the objective is defined as

$$J(q(\cdot), \eta; \tau) = J_{\text{task}}(q(\cdot), \eta; \tau) + w_1 \int_0^T \| \dddot{q}(t) \|^2 dt, \quad (9)$$

where J_{task} is the squared position error when the thrown object crosses the target box's z-level (height).

Constraints. We enforce limits on joint position, velocity, acceleration, and jerk; end-effector velocity limits; torque limits derived from the dynamics; and self-collision margins using capsule models. All constraints are expressed as $C(q, \dot{q}, \ddot{q}, \dddot{q}) \leq 0$, with added safety margins. We use the standard limits of the Franka Emika Panda, except that the end-effector velocity limits are doubled to enable throws of up to 2 m (in simulation).

Implementation. We weight trajectory errors by $c(t) = \exp\big(-4(t-\eta)^2\big)$ in (3) to achieve higher accuracy near the throwing time. In the basis model (4), $N_b = 100$. All modules use fully connected networks with GELU activations (4-6 layers, 1024 nodes). The subset of task parameters for optimization is

$$\mathcal{T}_s = \{(r, 0, h) \mid r \in \{1.1, \ldots, 2.0\},\ h \in \{0.0, 0.1, 0.2, 0.3\}\}.$$

All kinematics, dynamics, and constraints are implemented in PyTorch. The latent ODE (5) is integrated using the Euler method with step size $ds = 0.1$, providing a good balance between accuracy and speed.

A. Data Collection via Trajectory Optimizations

We use the parametric trajectory model $q_{q_0, q_T, w}$ (2) with $B = 20$, so $w \in \mathbb{R}^{20\times 7}$. The optimization variable w is initialized to zero, and η is set to 2. To randomly initialize q_0 and q_T within joint limits, we sample $v_0, v_T \in \mathbb{R}^7$ from a standard Gaussian, apply the sigmoid function, and scale them by the joint limits. Empirically, initialization in the vicinity of the origin leads to improved performance compared to uniform sampling, since extreme initial configurations can adversely affect convergence during optimization.

We first compare trajectory optimizers SLSQP [17], COBYLA [16], and Adam [18] by computation time and success rate. For Adam, constraints are added to the objective via a ReLU penalty. Optimization stops when constraints meet thresholds and position error < 0.01, or after 10,000 iterations. Fig. 5 shows that all solvers take over 10 s and are sensitive to initialization; Adam performs best but still fails beyond $r = 1.7$, highlighting the problem's complexity and the challenge of achieving sub-second planning.

We collect trajectories by repeatedly solving the optimization with the Adam optimizer. For each $\tau \in \mathcal{T}_s$, 300 trajectories are optimized, yielding 12,000 candidates across 40 parameters. After excluding failure cases, 3,523 valid trajectories remain. Fig. 6 (*Left*) shows their distribution over task parameters, and Fig. 6 (*Right*) shows the mean throwing time η as a function of the task parameters. As expected, fewer trajectories are obtained and η increases as the target distance grows.

B. Comparisons of Planning Performance

In this section, we compare *Trajectory Optimization (TO)* with the parametric curve models detailed in section III-A, and motion manifold-based methods: *Motion Manifold Primitives (MMP)* with a Gaussian mixture prior [5], *Motion Manifold Flow Primitives (MMFP)* [7], and our *Differentiable Motion Manifold Flow Primitives (DMMFP)*, each of which is trained with a 32-dimensional latent space. To train MMP, MMFP, and DMMFP, we use the same trajectory dataset collected via trajectory optimizations in section III-A. We fine-tune the DMMFP using *Trajectory Manifold Optimization (TMO)* introduced in section II-D, and denote it by $DMMFP + TMO$. Some generated trajectories by DMMFP + TMO may not meet the required constraints or

TABLE I

AVERAGE SUCCESS RATES, POSITION ERROR, AND VARIOUS CONSTRAINT SATISFACTION RATES. COMPUTATIONS ARE PERFORMED USING AN AMD MILAN 2.8 GHZ 8-CORE PROCESSOR FOR TRAJECTORY OPTIMIZATIONS AND AMD RYZEN 9 5900X 12-CORE PROCESSOR AND NVIDIA GEFORCE RTX 3090 FOR MOTION MANIFOLD PRIMITIVES.

	Seen Task Parameter									Unseen Task Parameter									# Traj.	Time
	SR	Error	JL	JVL	JAL	JJL	CVL	JTL	COL	SR	Error	JL	JVL	JAL	JJL	CVL	JTL	COL		
TO (Adam) [18]	97	0.01	100	100	100	100	100	100	100	100	0.01	100	100	100	100	100	100	100	1	10 $\sim$ 100 s
TO (SLSQP) [17]	33	0.01	100	100	100	100	100	100	100	100	0.01	100	100	100	100	100	100	100	1	10 $\sim$ 3000 s
TO (COBYLA) [16]	66	0.01	100	100	100	100	100	100	100	100	0.01	100	100	100	100	100	100	100	1	100 $\sim$ 1000 s
MMP [5]	1.05	0.53	0.95	0.0	0.0	0.0	0.0	0.0	92.6	0.81	0.53	96.4	0.0	0.0	0.0	0.0	0.0	94.4	100	0.003s
MMFP [7]	77.4	0.05	97.7	0.0	0.0	0.0	0.0	0.0	99.0	15.0	0.15	0.0	0.0	0.0	0.0	0.0	0.0	99.3	100	0.011s
DMMFP	17.5	0.18	99.2	72.5	25.9	100	89.5	64.7	98.6	4.96	0.26	99.4	79.5	32.7	100	93.1	87.4	98.4	100	0.012s
DMMFP + TMO	95.8	0.01	100	93.0	99.9	100	99.9	100	100	94.1	0.02	100	80	98.2	100	100	100	100	100	0.012s
DMMFP + TMO + RS	100	0.01	100	100	100	100	100	100	100	100	0.01	100	100	100	100	100	100	100	91	0.227s

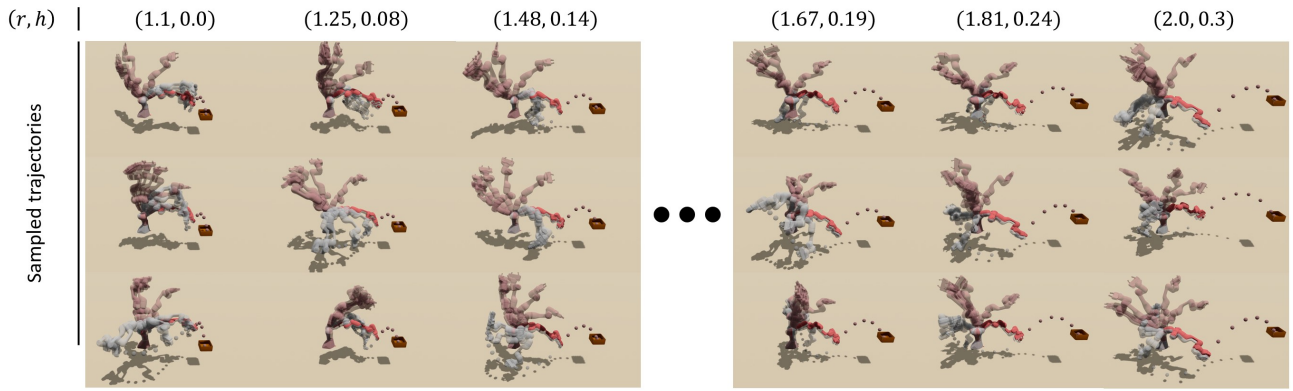

Fig. 7. Examples of diverse generated trajectories by *Differentiable Motion Manifold Primitives (DMMFP) + Trajectory Manifold Optimization (TMO) + Rejection Sampling (RS)* for various task parameters $\tau = (r, 0, h)$. The moment of the throw is depicted in deep red, preceded by gray and followed by transparent red.

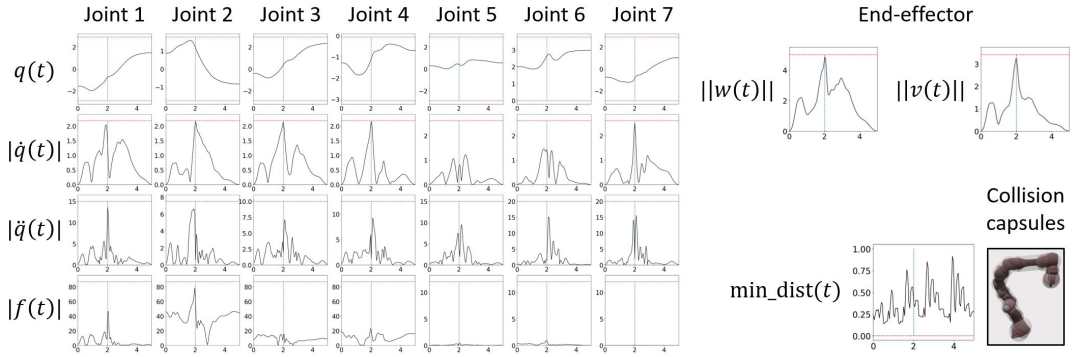

Fig. 8. Trajectories of joint $q(t)$, joint velocity $\dot{q}(t)$, joint acceleration $\ddot{q}(t)$, joint torque $f(t)$, end-effector angular velocity $w(t)$, end-effector linear velocity $v(t)$, and minimum distance between links $\text{min\_dist}(t)$ for an example throwing motion generated by *DMMFP + TMO + RS* given the task parameter $\tau = (2.0, 0, 0.3)$. The limit values are visualized as red horizontal lines. The moment of throw is marked in blue vertical lines.

task goals. Therefore, we use *Rejection Sampling (RS)* to exclude non-compliant samples, denoted by *DMMFP + TMO + RS*, although this step necessitates extra computation to verify the feasibility of the trajectories.

TABLE I presents the averaged Success Rate (SR) – a motion $(q(t), \eta)$ is deemed successful if the position error is less than 0.04 –, Position Error (Error) in meter scale, and Constraint Satisfaction Rates on Joint Limits (JL), Joint Velocity Limits (JVL), Joint Acceleration Limits (JAL), Joint Jerk Limits (JJL), Cartesian Velocity Limits (CVL), Joint Torque Limits (JTL), and Self-Collisions (COL). A trajectory is considered to satisfy a constraint if it meets the requirement at every time $t \in [0, T]$ along the trajectory. These metrics are computed using both seen task parameters $\tau \in \mathcal{T}_s$ and unseen test task parameters $\tau \in \{(r, 0, h) | r \in \{1.15, 1.25, \ldots, 1.95\}, h \in \{0.05, 0.15, 0.25\}\}$. The number of trajectories generated simultaneously is reported as # Traj. in these experiments, along with the corresponding time consumed.

From this table, we derive four key findings. First, trajectory optimization is significantly slower than manifold-based methods, highly sensitive to initialization, and can take over a minute as the target distance increases, making it impractical for online replanning (see Fig. 5). In contrast, motion-manifold methods are much faster, as neural networks enable immediate sampling, even for 100 trajectories

in parallel on a GPU.

Second, MMP, MMFP, and DMMFP all show low constraint-satisfaction rates, indicating that fitting data alone is insufficient. DMMFP performs relatively better, benefiting from inductive bias that enforces some temporal smoothness.

Third, applying TMO markedly improves DMMFP's performance. Residual failures are mainly due to Joint Velocity Limits (JVL), but using Rejection Sampling (RS) raises the success rate to 100%, retaining about 91 of 100 samples.

Lastly, RS slightly increases runtime because constraint checking is required: our implementation takes about 0.2 s to verify 100 trajectories across 100 time points, involving forward kinematics, inverse dynamics, and 45 constraint checks for each $(q, \dot{q}, \ddot{q}, \dddot{q})$ combination. All code is implemented in Python with PyTorch, measured on an AMD Ryzen 9 5900X and an NVIDIA RTX 3090. Although not fully optimized, further acceleration may be achieved through lower-level implementations or by training a neural classifier to quickly predict trajectory feasibility.

Diverse throwing motions generated by *DMMFP + TMO + RS* are visualized in Fig. 7. If only a single trajectory were available, the robot would often need to make inefficient adjustments, especially when its current configuration is far from that trajectory. In contrast, the diversity provided by our approach enables selecting a trajectory that better matches the current configuration.

An example motion for $\tau = (2.0, 0, 0.3)$ is shown in Fig. 8, illustrating joint angles, velocities, and other states. As seen in Fig. 8, the robot accelerates to near the joint and end-effector velocity limits (red dashed lines) to achieve a long throw, then decelerates. Peaks in velocity, acceleration, and torque occur near the throwing moment (blue dashed lines).

C. Online Adaptation

In this section, we demonstrate the online adaptability of our model in dynamically changing environments, enabled by its rapid planning speed. Consider a scenario where the robot is following an initially planned throwing trajectory for a given target box position, but the target position suddenly changes. We quickly replan a new throwing trajectory, allowing the robot to smoothly transition and track the updated trajectory without interruption.

In the replanning step, we sample 100 candidate trajectories $\{(q_i(t), \eta_i)\}_{i=1}^{100}$ conditioned on the new target. We then select the trajectory closest to the current configuration q_c by solving

$$(i^*, t^*) = \arg\min_{i,t} \|q_c - q_i(t)\| \quad \text{s.t. } t < \eta_i.$$

Next, we construct a transition trajectory from the current phase $(q_c, \dot{q}_c)$ to $(q_{i^*}(t^*), \dot{q}_{i^*}(t^*))$ in the selected trajectory, using the following parametric model:

$$\begin{aligned} q(t) = {} & q_0 + (q_T - q_0)(3 - 2s)s^2 + \dot{q}_0 s - (2\dot{q}_0 + \dot{q}_T)s^2 \\ & + (\dot{q}_0 + \dot{q}_T)s^3 + s^2(s-1)^2\Phi(s)w, \end{aligned} \tag{10}$$

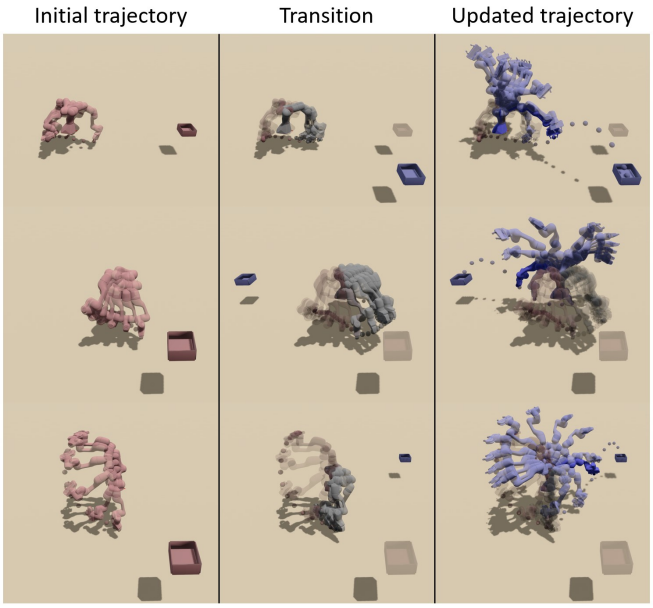

Fig. 9. Online adaptation to changes in target box position: The robot follows an initial trajectory (red), detects a target change, replans an updated trajectory (blue), and executes a transition trajectory (gray) connecting its current phase to the new plan.

in a manner analogous to (2). This model enforces $(q_0, \dot{q}_0)$ at $t = 0$ and $(q_T, \dot{q}_T)$ at $t = T$, and accommodates nonzero boundary velocities.

The transition trajectory must satisfy kinodynamic constraints, which reduces to finding a suitable w. Empirically, random Gaussian initialization of w often yields at least one valid solution. The process is fast enough to run online, enabling the robot to adapt in real time.

Fig. 9 illustrates examples: red robots follow the initial plan until the target moves (after 1.8 s), gray robots execute a transition trajectory over $\sim$1 s, and blue robots follow the updated throwing trajectory, completing the throw within 3–5 s depending on target distance.

IV. Conclusion

We have proposed Differentiable Motion Manifold Primitives (DMMP), which learns a manifold of continuous-time, differentiable trajectories offline and enables fast kinodynamic motion planning through online search. We have trained DMMP through four steps: data collection, manifold learning, latent-flow learning, and manifold optimization. Through dynamic throwing experiments with a 7-DoF arm, we have demonstrated that DMMP can rapidly generate trajectories that satisfy both task objectives and kinodynamic constraints.

Our work can be further strengthened by improving the initial data-collection step – currently based on randomizing initial and final configurations – as the diversity of collected trajectories directly determines the richness of the learned manifold and, consequently, the efficiency of online adaptation. Additionally, while this study focuses on planning, integrating tracking control and real-world experiments

would more comprehensively validate and demonstrate the effectiveness of the proposed approach.

REFERENCES

[1] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *International Conference on Learning Representations*, 2014.

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[3] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.

[4] M. Noseworthy, R. Paul, S. Roy, D. Park, and N. Roy, "Task-conditioned variational autoencoders for learning movement primitives," in *Conference on robot learning*. PMLR, 2020, pp. 933–944.

[5] B. Lee, Y. Lee, S. Kim, M. Son, and F. C. Park, "Equivariant motion manifold primitives," in *7th Annual Conference on Robot Learning*, 2023.

[6] Y. Lee, "Mmp++: Motion manifold primitives with parametric curve models," *IEEE Transactions on Robotics*, 2024.

[7] Y. Lee, B. Lee, S. Kim, and F. C. Park, "Motion manifold flow primitives for task-conditioned trajectory generation under complex task-motion dependencies," *IEEE Robotics and Automation Letters*, 2025.

[8] L. Chen, I. Mantegh, T. He, and W. Xie, "Fuzzy kinodynamic rrt: a dynamic path planning and obstacle avoidance method," in *2020 international conference on unmanned aircraft systems (ICUAS)*. IEEE, 2020, pp. 188–195.

[9] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 6741–6747.

[10] D. Malyuta, T. P. Reynolds, M. Szmuk, T. Lew, R. Bonalli, M. Pavone, and B. Açıkmeşe, "Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently," *IEEE Control Systems Magazine*, vol. 42, no. 5, pp. 40–113, 2022.

[11] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "Sampling-based optimal kinodynamic planning with motion primitives," *Autonomous Robots*, vol. 43, no. 7, pp. 1715–1732, 2019.

[12] M. Yavari, K. Gupta, and M. Mehrandezh, "Lazy steering rrt*: An optimal constrained kinodynamic neural network based planner with no in-exploration steering," in *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, 2019, pp. 400–407.

[13] P. Atreya and J. Biswas, "State supervised steering function for sampling-based kinodynamic planning," *arXiv preprint arXiv:2206.07227*, 2022.

[14] A. H. Qureshi, J. Dong, A. Baig, and M. C. Yip, "Constrained motion planning networks x," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 868–886, 2021.

[15] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip, "Motion planning networks: Bridging the gap between learning-based and classical motion planners," *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 48–66, 2020.

[16] M. J. Powell, *A direct search optimization method that models the objective and constraint functions by linear interpolation*. Springer, 1994.

[17] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

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

[19] G. Arvanitidis, L. K. Hansen, and S. Hauberg, "Latent space oddity: on the curvature of deep generative models," *arXiv preprint arXiv:1710.11379*, 2017.

[20] Y. Lee, H. Kwon, and F. Park, "Neighborhood reconstructing autoencoders," *Advances in Neural Information Processing Systems*, vol. 34, pp. 536–546, 2021.

[21] Y. Lee, S. Yoon, M. Son, and F. C. Park, "Regularized autoencoders for isometric representation learning," in *International Conference on Learning Representations*, 2022.

[22] Y. Lee and F. C. Park, "On explicit curvature regularization in deep generative models," in *Topological, Algebraic and Geometric Learning Workshops 2023*. PMLR, 2023, pp. 505–518.

[23] L. Lu, P. Jin, and G. E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," *arXiv preprint arXiv:1910.03193*, 2019.

[24] Y. Lee, "A geometric perspective on autoencoders," *arXiv preprint arXiv:2309.08247*, 2023.

[25] Y. Lee, S. Kim, J. Choi, and F. Park, "A statistical manifold framework for point cloud data," in *International Conference on Machine Learning*. PMLR, 2022, pp. 12 378–12 402.

[26] H. Heo, S. Oh, J. Y. Lee, Y. M. Kim, and Y. Lee, "Isometric regularization for manifolds of functional data," in *The Thirteenth International Conference on Learning Representations*, 2025.

[27] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[28] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," *arXiv preprint arXiv:2210.02747*, 2022.