DA-MMP: Learning Coordinated and Accurate Throwing with Dynamics-Aware Motion Manifold Primitives

Chi Chu[1], Huazhe Xu[2,1]

***Abstract*— Dynamic manipulation is a key capability for advancing robot performance, enabling skills such as tossing. While recent learning-based approaches have pushed the field forward, most methods still rely on manually designed action parameterizations, limiting their ability to produce the highly coordinated motions required in complex tasks. Motion planning can generate feasible trajectories, but the dynamics gap—stemming from control inaccuracies, contact uncertainties, and aerodynamic effects—often causes large deviations between planned and executed trajectories. In this work, we propose Dynamics-Aware Motion Manifold Primitives (DA-MMP), a motion generation framework for goal-conditioned dynamic manipulation, and instantiate it on a challenging real-world ring-tossing task. Our approach extends motion manifold primitives to variable-length trajectories through a compact parametrization and learns a high-quality manifold from a large-scale dataset of planned motions. Building on this manifold, a conditional flow matching model is trained in the latent space with a small set of real-world trials, enabling the generation of throwing trajectories that account for execution dynamics. Experiments show that our method can generate coordinated and smooth motion trajectories for the ring-tossing task. In real-world evaluations, it achieves high success rates and even surpasses the performance of trained human experts. Moreover, it generalizes to novel targets beyond the training range, indicating that it successfully learns the underlying trajectory–dynamics mapping.**

I. INTRODUCTION

Dynamic manipulation [1], [2]—the ability of robots to purposefully interact with objects through high-speed motions—has recently attracted growing interest [3]–[7]. While many manipulation tasks can be accomplished under assumptions of negligible object dynamics, dynamic manipulation requires leveraging object motion and momentum to achieve the goal. This setting poses two key challenges: coping with intricate, hard-to-model dynamics and generating trajectories that effectively exploit velocity and momentum to achieve complex tasks.

Recent dynamic manipulation systems [8]–[11] primarily use learning to model or compensate for complex task dynamics, while the motion generation process is typically based on hand-crafted models or manually designed action parameterizations. This limits their applicability to more complex tasks that require complicated and highly coordinated motion sequences. In dynamic manipulation research, movement primitives (MPs) have been widely used to learn and generate complex motions from demonstrations or planned trajectories [12]–[20]. By parameterizing motions with a small set of basis-function weights, MPs enable compact representation and efficient modulation of key attributes such as speed, amplitude, and timing, making them particularly suitable for high-speed, dynamic tasks. Building on this idea, motion manifold primitives (MMPs) [21]–[25] replace hand-crafted parameter spaces with low-dimensional manifolds learned directly from large collections of feasible trajectories. While recent works have extended MMPs to tasks such as tossing [25], they generally assume fixed-length trajectories, are trained on relatively modest-scale datasets, and have not been fully explored in challenging real-world dynamic manipulation scenarios.

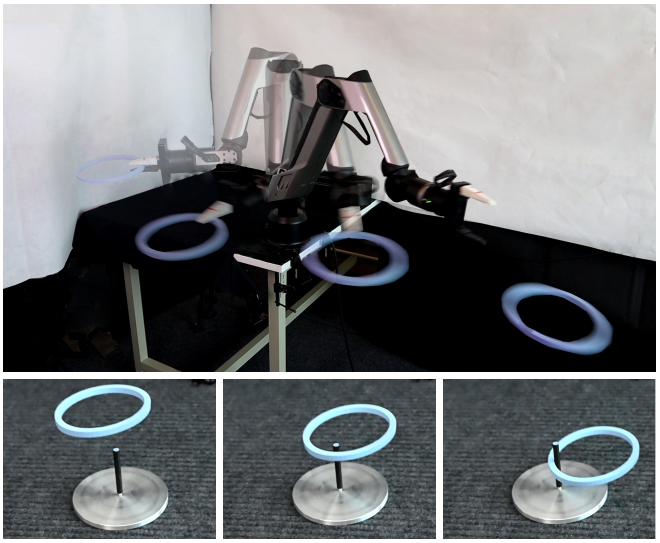

Fig. 1. Real-world ring tossing with trajectories synthesized by DA-MMP. The first line shows the overall sequence, where superimposed snapshots (light to dark) indicate temporal order of the coordinated throw. The second line illustrates three representative frames of the hitting process.

In this work, we bridge these two lines of research by introducing Dynamics-Aware Motion Manifold Primitives (DA-MMP), a motion generation framework for complex dynamic manipulation that integrates motion manifold primitives with large-scale planning data and trajectory-level dynamics learning. We focus on a challenging real-world ring-tossing task, as illustrated in Fig. 1, that requires generating highly coordinated motions to throw a ring around a distant target peg. This setting is particularly difficult due to multiple sources of dynamics gaps, including control inaccuracies of the arm (control gaps), slippage and uncertainty in release timing (contacts gap), and aerodynamic effects during flight (aerodynamic gaps). Our framework extends MMPs to variable-length trajectories via a compact parameterization

[1] Shanghai Qi Zhi Institute; [2] Institute for Interdisciplinary Information Sciences, Tsinghua University.
Contact: cc2018011855@gmail.com, huazhe_xu@mail.tsinghua.edu.cn

that includes execution duration, and leverages a large-scale dataset of planned trajectories generated by sampling-based motion planning to learn a high-quality motion manifold. A conditional generative model is then trained in the learned latent space to generate throwing trajectories. By training on the observed landing outcomes of executed trajectories, this model implicitly captures the relationship between trajectories and their execution dynamics. Experiments demonstrate that our approach achieves high real-world success rates, even rivaling trained human experts, and can generalize to unseen targets, which indicates that it generates valid trajectories and effectively captures task dynamics.

Our work makes the following contributions at the intersection of dynamic manipulation and motion manifold learning:

- **Challenging dynamic manipulation task.** We design and address a real-world ring-tossing task that requires generating complex, highly coordinated motions, providing a demanding benchmark for evaluating expressive motion generation.
- **Variable-length motion manifold primitives with dynamics learning.** We extend the MMP framework to represent variable-length trajectories through a compact parameterization that includes execution duration, and introduce a conditional generative model in the latent space that captures the relationship between trajectories and their execution dynamics.
- **Large-scale dataset and efficient real-world learning.** Leveraging sampling-based motion planning, we collect a large-scale dataset of feasible trajectories to learn a high-quality motion manifold, and further enable data-efficient learning from limited real-world trials.

II. Related Work

Our approach builds on and connects two main lines of research. The first line of research is motion manifold primitives, which learn low-dimensional latent representations of feasible trajectories. The second line is learning-based goal-conditioned dynamic manipulation, which applies machine learning to address the challenges of high-speed, target-directed manipulation. We review both directions in the following subsections.

A. Motion Manifold Primitives

Motion Manifold Primitives [22] build upon the idea of movement primitives [12], [15], [16], which offer structured representations for generating robot motions. Instead of relying on hand-crafted parameter spaces, MMPs learn low-dimensional manifolds directly from large collections of feasible trajectories. A closely related idea was explored by Noseworthy et al. [21], who used task-conditioned variational autoencoders to learn latent spaces of movement primitives. Subsequent work has expanded MMPs in several directions, including conditional generative models for task-specific motion synthesis [24], extensions with parametric curves for temporal and via-point modulation [23], and a differentiable formulation that enables further optimization of the learned manifold with downstream objectives [25].

While these efforts have demonstrated MMPs across diverse domains, they typically assume fixed-length trajectories, are trained on relatively modest-scale datasets, and have not been fully explored in challenging real-world dynamic manipulation. In contrast, our work extends MMPs to variable-length trajectories via a compact parameterization, learns high-quality manifolds from large-scale planned motion data without requiring downstream post-optimization, and incorporates a conditional generative model that implicitly captures the relationship between trajectories and their execution dynamics.

B. Learning-based goal-conditioned dynamic manipulation

Recent works have applied deep learning to high-speed, goal-conditioned manipulation tasks with intricate dynamics. Among the extensive literature, we highlight several representative efforts. TossingBot [8] learns residual release velocities conditioned on grasp and object's visual features to compensate for various dynamics gaps. IRP [9] addresses dynamic manipulation of deformable objects by learning delta dynamics in an iterative residual policy framework, enabling online adaptation to new dynamics and hardware. DartBot [10] leverages tactile sensing and reinforcement learning to exploit contact interactions, learning slippage and release timing in real-world overhead dart throwing. Whole-body throwing with legged manipulators [26] combines a residual control policy to address control gaps with tube acceleration [27] to handle release-time uncertainties. The IPA policy [11] further tackles dynamic manipulation with soft-body tools by implicitly identifying the underlying physics of soft–rigid interactions through system identification. Unlike these approaches, which largely rely on hand-crafted motion generation rules with constrained expressiveness, our method synthesizes high-dimensional trajectories through planning and manifold learning. Moreover, target-level residual learning, as in TossingBot, is unsuitable in our setting since distinct trajectories for the same target can yield different landings; instead, we learn dynamics directly at the trajectory level to capture such differences.

Like our work in spirit, earlier studies also applied movement primitives to dynamic manipulation to generate smooth and consistent trajectories. Representative examples include learning motor primitives for hitting and batting [14], swinging and padding [13], and applying mixtures of motor primitives to table tennis [28], [29]. These approaches relied on adapting from a limited number of demonstrations, whereas our method learns motion manifold primitives from large-scale planned trajectories and uses limited real-world data to capture dynamics.

III. Method

In this section, we present DA-MMP, a framework that leverages motion manifold learning for dynamics-aware trajectory generation. The framework consists of two stages, as shown in Fig. 2: (1) data collection and parameterization,

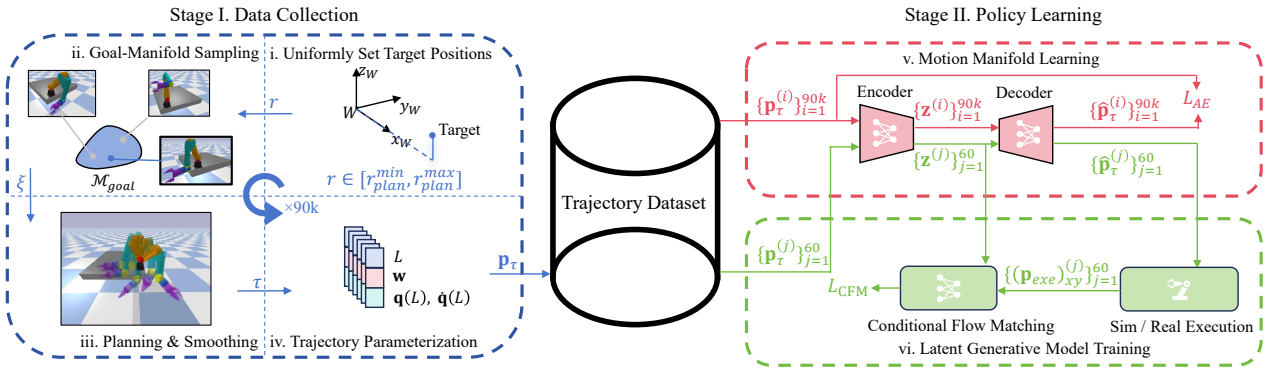

Fig. 2. Overview of DA-MMP. Stage I (blue): data collection from goal-manifold sampling, motion planning, and parameterization, yielding 90k planned trajectories. Stage II (red/green): policy learning with an autoencoder to learn motion manifolds and a conditional flow-matching model for trajectory-dynamics mapping.

and (2) policy learning using a motion manifold representation and a conditional flow-matching model. We begin by introducing the problem formulation, followed by a detailed description of each stage.

A. Problem Formulation

We study goal-conditioned dynamic manipulation tasks that require complex motions, exemplified by a ring-tossing task. In this task, the objective is to throw a ring of radius R_{ring} such that it lands around a vertical cylinder of radius R_{cyl} and height z_{cyl} on the ground. To simplify the problem, we assume a fixed hand-crafted grasp of the ring. A tossing trajectory is defined as

$$\tau = \{\mathbf{q}(t), \dot{\mathbf{q}}(t)\}_{t=0}^{L}, \tag{1}$$

where $\mathbf{q}(t)$ and $\dot{\mathbf{q}}(t)$ denote the joint positions and velocities at time t, and L is the execution horizon, which can vary across different trajectories. At time $t = L$, the gripper starts opening. During a subsequent settling horizon h_{settle}, the end-effector maintains its linear and angular velocity, allowing sufficient time for the ring to be released from the gripper. Inspired by how humans perform ring tossing, we design the robot to start and end at the same joint configuration with zero initial velocity:

$$\mathbf{q}(0) = \mathbf{q}(L), \quad \dot{\mathbf{q}}(0) = \mathbf{0}. \tag{2}$$

Given the target position, we exploit the rotational symmetry of the manipulator's first joint and represent the target in polar coordinates. Without loss of generality, we fix the polar angle $\alpha = 0$, so that the problem reduces to generating a trajectory $\tau(r)$ for a given radial distance r. In practice, however, the target placement and the ring landing position are not perfectly aligned with $\alpha = 0$, so we represent both using their (x, y) coordinates, where y is nearly zero. We consider a trial as successful if, at height $z = z_{\text{cyl}}$,

$$\left\|(x_{\text{ring}}(z_{\text{cyl}}),\, y_{\text{ring}}(z_{\text{cyl}})) - (x_T, y_T)\right\|_2 \;\le\; R_{\text{ring}} - R_{\text{cyl}}, \tag{3}$$

where $(x_{\text{ring}}(z_{\text{cyl}}),\, y_{\text{ring}}(z_{\text{cyl}}))$ are the horizontal coordinates of the ring center at $z = z_{\text{cyl}}$, and (x_T, y_T) denotes the horizontal coordinates of the target top center. This serves as an approximation, since the ring may not remain perfectly horizontal during flight.

B. Data Collection

1) Goal-Manifold Sampling: We adopt the idea of goal-manifold sampling from prior work on robotic throwing [30], [31], where projectile equations are used to generate candidate throw states ξ, forming a manifold of feasible goals $\mathcal{M}_{\text{goal}}$ used for motion planning. Each candidate specifies the desired end-effector (EE) pose and velocity at release. If the planner fails to generate a feasible trajectory, a new candidate is drawn until success.

Formally, we parameterize each throw state in the ring space, since the EE state is uniquely determined via a constant grasp transform ${}^{E}T_{R}$. The grasp is designed such that the ring center lies along the EE z-axis and the release direction aligns with the EE x-axis, which reduces the likelihood of post-release collisions with the gripper. We denote a ring throw state by ξ_R, consisting of its pose and twist in the world frame:

$$\xi_R = \left({}^{W}T_R,\ {}^{W}\mathbf{v}_R,\ {}^{W}\boldsymbol{\omega}_R\right) \in SE(3) \times \mathbb{R}^6,$$

which comprises twelve degrees of freedom. In the following we describe the constraints that define this feasible manifold of throws.

The twelve degrees of freedom of ξ_R are constrained as follows, with reference to the coordinate frames in Fig. 3. Let ${}^{W}\mathbf{p}_R$ and ${}^{W}\mathbf{p}_T$ denote the positions of the ring center and the target top center in the world frame W. The orientation is fixed by geometry, where (x_R, y_R, z_R) denote the unit axes of the ring frame R:

$$x_R \parallel ({}^{W}\mathbf{p}_T - {}^{W}\mathbf{p}_R)_{xy}, \qquad z_R \parallel z_W, \qquad y_R = z_R \times x_R.$$

The angular velocity is restricted to spin about the ring normal:

$${}^{W}\boldsymbol{\omega}_R = (0,\, 0,\, \omega_z)^{\top},$$

where ω_z is sampled from the range $[\omega_{\min}, \omega_{\max}]$. The ring center position ${}^{W}\mathbf{p}_R = (x, y, z)$ is sampled within the workspace. In practice we use a set of surrogate variables

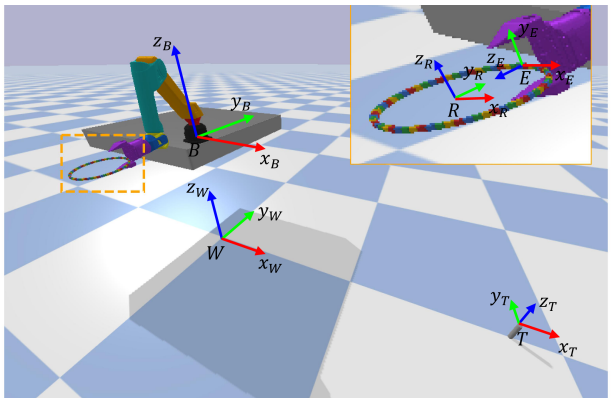

Fig. 3. Coordinate frames used for goal-manifold sampling. World W is fixed to the ground; B attaches to the robot base; the end-effector E rigidly grasps the ring frame R via ${}^{E}T_{R}$; the target frame T sits at the cylinder top center. Inset: enlarged view of E and R for clarity.

to generate these samples, but omit the details here for simplicity. For linear velocity, we impose

$$v_z = 0, \qquad {}^{W}\mathbf{v}_R \parallel x_R,$$

where v_z denotes the vertical component of ${}^{W}\mathbf{v}_R$. Its magnitude is determined by projectile motion to reach the target height $z = z_{\text{cyl}}$:

$$\|{}^{W}\mathbf{v}_R\| = d_{xy}\sqrt{\frac{g}{2\,(z_R - z_{\text{cyl}})}}, \tag{4}$$

where $d_{xy} = |({}^{W}\mathbf{p}_T - {}^{W}\mathbf{p}_R)xy|$ is the horizontal distance to the target, and g is the gravitational acceleration constant. Together these constraints define a low-dimensional manifold of throws. In practice, candidates are drawn from this manifold and checked for feasibility using inverse kinematics and collision detection; if a candidate fails, sampling is repeated until a feasible one is found.

2) Sampling-Based Motion Planning: Sampling-based motion planning is renowned for its computational efficiency in high-dimensional spaces [32], which enables us to generate a large-scale dataset of feasible trajectories. Given a sampled throwing state, we construct the motion planning problem by setting the goal state to this specified throwing state and the initial state to the same joint configuration with zero velocity. We employ DIMT-RRT as our kinodynamic planning algorithm [33], where each planning attempt is allowed up to N_{planning} samples. Successful plans are further refined with $N_{\text{smoothing}}$ iterations of a trajectory smoothing algorithm based on [34]. We also adopt the quick rejection strategy proposed in [30] to efficiently discard infeasible candidates before planning. During planning, we perform discrete collision checking with a time resolution of Δt_{col}.

After a trajectory is found, we execute it twice in simulation and compare the resulting landing positions; the trajectory is retained only if the deviation between the two trials is below a threshold, thus filtering out unstable throwing trajectories. Finally, the accepted trajectory is interpolated to match the control frequency f_{ctrl}.

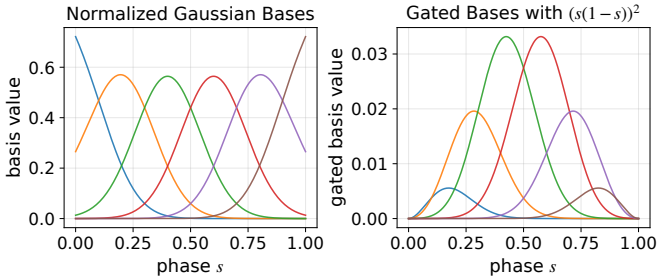

Fig. 4. Illustration of basis functions used for trajectory parameterization. Left: normalized Gaussian bases over the phase. Right: gated bases obtained by multiplying the normalized bases with the polynomial gate $(s(1-s))^2$. For illustration, we show $K = 6$.

C. Policy Learning

1) Variable-Length Parameterization of Motion Trajectories: A key challenge in trajectory parameterization is that planned trajectories have variable durations and cannot be directly resampled or aligned, since naive interpolation effectively accelerates or decelerates the motion and alters the release velocity that defines the throwing state. To handle variable durations without distorting the motion, we include the trajectory length L as an explicit component of the trajectory parameterization. Another challenge is ensuring that the generated trajectories are sufficiently smooth for high-speed execution, which we address by drawing inspiration from movement primitives and representing trajectories with radial basis functions. Formally, each trajectory dimension is approximated as a weighted combination of normalized Gaussian radial basis functions,

$$\begin{aligned} q(s;\mathbf{w}) &= \mathbf{w}^{\top}\boldsymbol{\phi}(s), \\ \phi_i(s) &= \frac{\kappa_i(s)}{\sum_{j=1}^{K}\kappa_j(s)}, \qquad i = 1,\dots,K, \\ \kappa_i(s) &= \exp\left(-\frac{(s-c_i)^2}{h}\right), \end{aligned} \tag{5}$$

where $s \in [0,1]$ is the normalized phase, c_i are the basis centers, h is the bandwidth, and $\mathbf{w}$ are the basis weights for this trajectory dimension. Here $\boldsymbol{\phi}(s) = [\phi_1(s),\dots,\phi_K(s)]^{\top}$ denotes the vector of normalized basis functions.

We find that directly using radial basis functions often leads to oscillations in the end-point velocity. To address this, we adopt via-point movement primitives [16], which explicitly constrain the start- and end-point positions and velocities. Formally, each trajectory is expressed as

$$q(s;\mathbf{w}) = \psi(s) + \mathbf{w}^{\top}\boldsymbol{\phi}(s), \tag{6}$$

where $\psi(s)$ is a cubic Hermite spline interpolating the boundary conditions $\{q(0), q(1), \dot{q}(0), \dot{q}(1)\}$, and $\boldsymbol{\phi}(s)$ denotes the gated basis functions, obtained by multiplying the normalized Gaussian bases with the polynomial gate function $(s(1-s))^2$. For illustration, we show examples of the normalized Gaussian and gated bases in Fig. 4.

Given a reference trajectory defined on L phase points $\{(q(s_l), \dot{q}(s_l))\}_{l=1}^{L}$ with $s_l = \frac{l}{L}$, the weights $\mathbf{w}$ are obtained

by least-squares fitting with equal weighting of position and velocity terms,

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \sum_{l=1}^{L} \left(q(s_l) - \psi(s_l) - \boldsymbol{\phi}(s_l)^\top \mathbf{w}\right)^2 + \sum_{l=1}^{L} \left(\tfrac{1}{\alpha}\dot{q}(s_l) - \psi'(s_l) - \boldsymbol{\phi}'(s_l)^\top \mathbf{w}\right)^2, \tag{7}$$

where $\psi(s_l)$ and $\psi'(s_l) = \frac{d\psi}{ds}(s_l)$ denote the spline and its phase derivative, $\boldsymbol{\phi}(s_l)$ and $\boldsymbol{\phi}'(s_l) = \frac{d\boldsymbol{\phi}}{ds}(s_l)$ are the gated basis functions and their phase derivatives evaluated at s_l, and $\alpha = \frac{ds}{dt} = \frac{1}{L}$ is the time-to-phase scaling factor (so $(1/\alpha)\dot{q} = dq/ds$). Each trajectory is therefore represented by weights $\mathbf{w}$, the end-point conditions $(q(1), \dot{q}(1))$, and the trajectory length L, which we collectively denote as the parameter vector $\mathbf{p}_\tau$.

2) Learning Motion Manifold Primitives: Given the trajectory parameterization $\mathbf{p}_\tau$, we learn motion manifold primitives by first embedding trajectories into a low-dimensional latent space $\mathbf{z}_\tau$ using an autoencoder, which captures the underlying manifold structure and reduces dimensionality. The autoencoder is trained to minimize the reconstruction loss

$$\mathcal{L}_{\text{AE}} = \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{p}_\tau^{(i)} - \hat{\mathbf{p}}_\tau^{(i)} \right\|_2^2, \tag{8}$$

where $\hat{\mathbf{p}}_\tau^{(i)}$ is the decoded reconstruction of $\mathbf{p}_\tau^{(i)}$.

On top of this latent representation, we train a conditional flow-matching model [35]–[38], where the conditioning signal is given by the actual landing point $(x_{\text{exe}}, y_{\text{exe}})$ of the executed trajectory measured at $z = z_{\text{cyl}}$, implemented via classifier-free guidance [39]. The model is optimized with the standard conditional flow-matching loss

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{u,\, \mathbf{z}_\tau,\, \mathbf{z}_{\text{noise}}} \left[\left\| v_\theta(\mathbf{z}(u), u, \mathbf{c}) - v^\star(u) \right\|_2^2 \right], \tag{9}$$

where $u \in [0, 1]$ denotes the flow time parameter, $\mathbf{z}(u)$ is the linear interpolation between a Gaussian prior sample $\mathbf{z}_{\text{noise}}$ and the ground-truth latent variable of the trajectory $\mathbf{z}_\tau$, $v^\star(u)$ denotes the corresponding target velocity field, and $\mathbf{c} = (x_{\text{exe}}, y_{\text{exe}})$ is the conditioning signal.

At inference time, given the conditioning signal $\mathbf{c} = (x_T, y_T)$, a noisy latent sample $\mathbf{z}_{\text{noise}}$ is progressively transported to a trajectory latent $\mathbf{z}_\tau$ by integrating the learned vector field $v_\theta(\mathbf{z}(u), u, \mathbf{c})$ over $u \in [0, 1]$. The resulting latent $\mathbf{z}_\tau$ is then decoded by the autoencoder to obtain the parameterized trajectory, which is used for the ring-tossing task.

This design implicitly captures the dynamics gap between planned and executed motions. Notably, the autoencoder is trained on a large set of planned trajectories to learn a high-quality motion manifold, while the flow-matching model is learned in the latent space and thus requires only a modest amount of real-world data to capture trajectory structure and dynamics information.

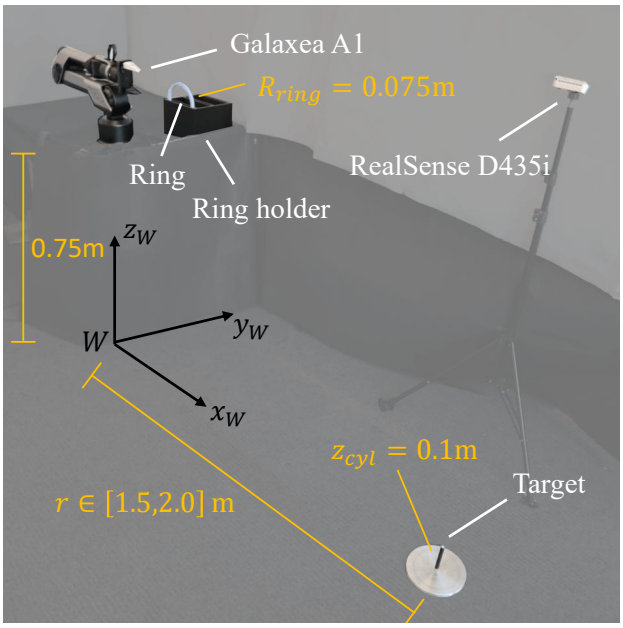

Fig. 5. Experimental setup of real-world ring tossing.

IV. EXPERIMENTS

A. Setup

1) Simulation and Real-World Setup: We evaluate our approach in both simulation and real-world experiments using a 6-DoF Galaxea A1 fixed-base robot arm, with simulation implemented in PyBullet [40]. The arm is mounted on a fixed table of height 0.75 m, providing sufficient flight time for the ring. Its joint acceleration limits are set to $[12.5, 12.5, 12.5, 15.0, 15.0, 15.0]$ m/s^2 to balance safety and motion capability. The ring has a radius of $R_{\text{ring}} = 0.075$ m, and the target is a vertical cylinder of radius $R_{\text{cyl}} = 0.005$ m and height $z_{\text{cyl}} = 0.1$ m placed on the ground. In the real setup (Fig. 5), a RealSense D435i depth camera is used to detect the x, y coordinates of the target and the ring position at $z = 0.1$ m. For the ring, contours are extracted by OpenCV Canny edge detection and fitted with ellipses for robust localization [41].

2) Data Collection Details: We collected a dataset of 90k feasible trajectories for training the autoencoder, with target radial distances uniformly sampled within $[r_{\text{plan}}^{\min}, r_{\text{plan}}^{\max}] = [1.0, 2.5]$ m. For goal-manifold sampling, the ring angular velocity ω_z was drawn from $[\omega_{\min}, \omega_{\max}] = [1.5\pi, 3\pi]$ rad/s. We set $N_{\text{planning}} = 80$, $N_{\text{smoothing}} = 100$, $\Delta t_{\text{col}} = 1/30$ s, and $f_{\text{ctrl}} = 240$ Hz. For training the flow-matching model, we collected 60 executed trajectories by uniformly setting target distances from $[r_{\text{exec}}^{\min}, r_{\text{exec}}^{\max}] = [1.5, 2.0]$ m in both simulation and real-world settings. Although the dataset is small, we will demonstrate in the experiments that the model effectively captures dynamics information and exhibits generalization capability. The difference between $[r_{\text{plan}}^{\min}, r_{\text{plan}}^{\max}]$ and $[r_{\text{exec}}^{\min}, r_{\text{exec}}^{\max}]$ reflects both safety considerations in real execution and the need to reserve part of the range for evaluating generalization.

TABLE I

SUCCESS RATE (SR) COMPARISON OF DIFFERENT BASELINES AND OUR METHOD IN SIMULATION AND REAL-WORLD EXPERIMENTS.

Method	Simulation SR (%)	Real SR (%)
Motion planning (1 attempt)	0.0	13.3
Motion planning (2 attempts)	0.0	23.3
Residual-style correction	**93.3**	6.7
Ours	53.3	**60.0**
Human novice	–	13.3
Human expert	–	56.7

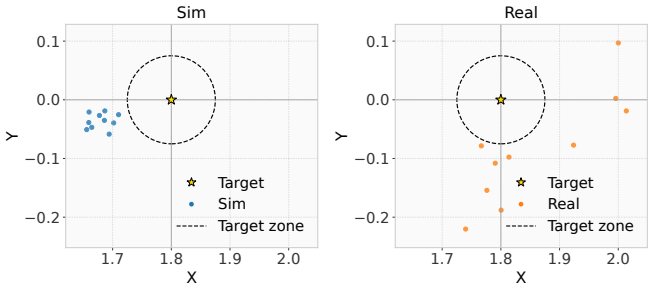

Fig. 6. Landing distributions of throws in simulation and the real world. Ten trajectories are planned targeting $r = 1.8\,\text{m}$, and the actual landing positions are plotted.

3) Policy Learning Details: We use $K = 30$ basis functions, which provides a good balance between representation accuracy and model complexity. The autoencoder maps each trajectory parameterization $\mathbf{p}_\tau$ into a d_z-dimensional latent space, with $d_z = 64$ in our implementation. Both the encoder and decoder are 3-layer MLPs with hidden dimensions [256, 512, 256], and Leaky ReLU activations. The model is trained for up to 10,000 epochs with early stopping, using batch size 256 and Adam optimizer (learning rate 1×10^{-4}, weight decay 1×10^{-5}). The learned latent space then serves as input for the conditional flow-matching model. For this model, the network is a 6-layer MLP with hidden dimensions [256, 512, 1024, 1024, 512, 256] and Swish activations. Training uses batch size 450 with Adam optimizer (learning rate 3×10^{-4}, weight decay 1×10^{-6}) for up to 20,000 epochs. During trajectory generation, we integrate the learned vector field with the midpoint method using a step size of 0.001. All neural network inputs are normalized by subtracting the mean and dividing by the standard deviation.

B. Dynamics-Aware Trajectory Generation

To evaluate the effectiveness of our method in generating accurate throwing trajectories under dynamics gap, we compare it against multiple baselines in both simulation and real-world experiments, where in simulation we additionally introduce air drag by setting linear and angular damping. We consider three baselines in simulation: motion planning with a single attempt, motion planning with two attempts, and a residual-style correction method that replans the second throw by compensating for the error observed in the first attempt, which is applied only when the first throw is unsuccessful. In the real world, we further include two human baselines: novice participants throwing without any prior practice, and expert participants who have practiced for 60 trials before evaluation. We report success rate (SR) as the evaluation metric, defined as the fraction of trials in which the ring successfully lands around the target peg. All results are averaged over three random seeds, each tested on 10 randomly sampled target positions.

The success rates of all methods are summarized in Table I. Residual-style correction performs best in simulation but fails in the real world. Our method achieves the highest success rate in reality, even slightly outperforming human expert performance.

To further understand these results, we visualize the landing distributions of throws in both simulation and the real world, as shown in Fig. 6. In this test, we planned ten trajectories targeting $r = 1.8\,\text{m}$, and plot the actual landing positions of the executed throws. In simulation, the landing points exhibit a consistent bias with small variance, since control and contact dynamics gaps are minimal while the added air drag dominates the discrepancy. In contrast, the real-world distribution exhibits both bias and variance due to additional control and contact dynamics gaps as well as visual perception errors. This explains why the residual-style correction is highly effective in simulation, as a single bias can be compensated by adjusting the second target, but it fails in the real world because variance cannot be corrected in the same way. In fact, its improvement in success rate is even less effective than simply replanning once (two attempts versus one). Our method, by encoding trajectories rather than target positions, can capture trajectory-level dynamics and thus remains effective under real-world variability. Interestingly, when trained with the same number of trials as the human expert, our method slightly outperforms it, highlighting the efficiency of first constructing motion primitives and then learning dynamics on top of them. Moreover, we found that our method demonstrates generalization capability: when attempting throws to a distance of 1.2 m, for which no real trajectories were collected, the policy was still able to generate feasible motions that successfully hit the target. This result indicates that the policy has indeed learned the underlying trajectory–dynamics mapping, rather than merely memorizing training samples.

C. Ablations

1) Effect of the Autoencoder: To evaluate the role of the autoencoder in facilitating efficient training of the downstream generative model, we compared our approach against training the flow-matching model directly on raw trajectory parameterizations. This variant lacks the manifold regularization provided by the embedding stage and therefore must model high-dimensional inputs directly. On this test, we generate trajectories in an unconditional manner. As shown in Fig. 7, the non-embedded variant yields irregular joint motion profiles that are rarely executable, whereas our method generates smooth and feasible trajectories. This demonstrates that the autoencoder is essential for capturing the intrinsic structure of feasible trajectories and for mapping trajectories with their underlying dynamics.

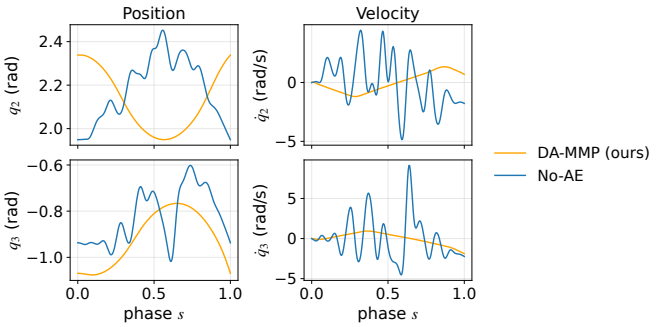

Fig. 7. **Effect of the autoencoder on trajectory generation.** Shown are positions (q_2, q_3) and velocities $(\dot{q}_2, \dot{q}_3)$ of two representative joints. Our method (orange) produces smoother and more coherent trajectories, whereas the no-AE variant (blue) exhibits irregular and oscillatory motions.

TABLE II

EFFECT OF DATASET SCALE ON AUTOENCODER RECONSTRUCTION. RMSE IS MEASURED IN THE NORMALIZED TRAJECTORY PARAMETER SPACE, AND LRE IS DEFINED IN (10).

Dataset size	Parameter-Space RMSE	LRE (%)
0.9k	0.700	22.6
9k	0.005	1.3
90k	**0.001**	**0.7**

2) Effect of Dataset Scale: We further study the effect of dataset scale on the quality of the learned motion manifold. The autoencoder is trained with three different dataset sizes: 0.9k, 9k, and the full 90k planned trajectories, with the network capacity reduced proportionally for smaller datasets to mitigate overfitting. For evaluation, we sample 100 trajectories with a different random seed as a held-out test set. We report two metrics: the normalized parameter-space reconstruction error (RMSE), and the relative length reconstruction error

$$\mathrm{LRE} = \frac{1}{N}\sum_{j=1}^{N}\frac{|L^{(j)} - \hat{L}^{(j)}|}{L^{(j)}} \times 100\%. \tag{10}$$

where $L^{(j)}$ and $\hat{L}^{(j)}$ denote the ground-truth and reconstructed trajectory lengths. As summarized in Table II, larger training sets consistently yield lower RMSE and LRE, demonstrating that a sufficiently large corpus of planned trajectories is necessary to capture the diversity of feasible motions and to establish a high-quality manifold that generalizes across targets.

V. CONCLUSION

In this work, we presented DA-MMP, a framework that builds upon motion manifold primitives for complex goal-conditioned dynamic manipulation tasks. Our approach extends motion manifold primitives to variable-length trajectories through a compact parameterization, and leverages a large-scale dataset of planned trajectories to learn a high-quality manifold. On top of this manifold, we introduced a conditional flow matching model in the latent space that captures the relationship between trajectories and their execution dynamics, while relying on only limited real-world trials. Our experimental evaluation confirms that DA-MMP consistently produces smooth and coordinated throwing motions with high real-world success rates. Future directions include extending the framework to generalize across objects with diverse shapes and physical properties, exploiting its expressive motion generation to control object orientation and spin at release, and applying it to a broader range of goal-conditioned dynamic manipulation tasks.

ACKNOWLEDGMENT

The authors would like to thank Zhecheng Yuan, Kun Lei, Qianwei Han, Xueyi Liu, and Lingxiao Guo for hardware assistance, Kaizhe Hu, Yu Qi for idea discussions, and Prof. Kris Hauser and Dr. Tobias Kunz for helpful discussions on kinodynamic planning algorithms. This work is supported by Tsinghua University Dushi Program.

REFERENCES

[1] M. Mason and K. Lynch, "Dynamic manipulation," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 1993, pp. 152–159.

[2] K. Lynch and M. Mason, "Dynamic nonprehensile manipulation: Controllability, planning, and experiments," *Int. J. Robot. Res.*, vol. 18, no. 1, pp. 64–92, 1999.

[3] Y. Ma, A. Cramariuc, F. Farshidian, and M. Hutter, "Learning coordinated badminton skills for legged manipulators," *Sci. Robot.*, vol. 10, no. 102, p. eadu3922, 2025.

[4] Z. Su, B. Zhang, N. Rahmanian, Y. Gao, Q. Liao, C. Regan, K. Sreenath, and S. Sastry, "Hitter: A humanoid table tennis robot via hierarchical planning and learning," 2025.

[5] D. D'Ambrosio, S. Abeyruwan, L. Graesser, A. Iscen, H. Ben Amor, A. Bewley, B. Reed, K. Reymann, L. Takayama, Y. Tassa, K. Choromanski, E. Coumans, D. Jain, N. Jaitly, N. Jaques, S. Kataoka, Y. Kuang, N. Lazic, R. Mahjourian, S. Moore, K. Oslund, A. Shankar, V. Sindhwani, V. Vanhoucke, G. Vesom, P. Xu, and P. Sanketi, "Achieving human level competitive robot table tennis," 2025.

[6] Y. Zhang, T. Liang, Z. Chen, Y. Ze, and H. Xu, "Catch it! learning to catch in flight with mobile dexterous hands," 2024.

[7] B. Huang, Y. Chen, T. Wang, Y. Qin, Y. Yang, N. Atanasov, and X. Wang, "Dynamic handover: Throw and catch with bimanual hands," 2023.

[8] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," in *Proc. Robotics: Science and Systems (RSS)*, 2019.

[9] C. Chi, B. Burchfiel, E. Cousineau, S. Feng, and S. Song, "Iterative residual policy for goal-conditioned dynamic manipulation of deformable objects," in *Proc. Robotics: Science and Systems (RSS)*, 2022.

[10] S. Aslam, K. Kumar, P. Zhou, H. Yu, M. Wang, and Y. She, "Dartbot: Overhand throwing of deformable objects with tactile sensing and reinforcement learning," *IEEE Trans. Autom. Sci. Eng.*, vol. 22, pp. 13 644–13 661, 2025.

[11] Z. Wang and A. Qureshi, "Implicit physics-aware policy for dynamic manipulation of rigid objects via soft body tools," 2025.

[12] A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Comput.*, vol. 25, no. 2, pp. 328–373, 2013.

[13] J. Kober and J. Peters, "Learning motor primitives for robotics," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2009, pp. 2112–2118.

[14] J. Kober, K. Mülling, O. Krömer, C. Lampert, B. Schölkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 853–858.

[15] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2013.

[16] Y. Zhou, J. Gao, and T. Asfour, "Learning via-point movement primitives with inter- and extrapolation capabilities," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 4301–4308.

[17] A. Pervez, Y. Mao, and D. Lee, "Learning deep movement primitives using convolutional neural networks," in *Proc. IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2017, pp. 191–197.

[18] S. Bahl, M. Mukadam, A. Gupta, and D. Pathak, "Neural dynamic policies for end-to-end sensorimotor learning," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 5058–5069.

[19] G. Li, Z. Jin, M. Volpp, F. Otto, R. Lioutikov, and G. Neumann, "Prodmp: A unified perspective on dynamic and probabilistic movement primitives," *IEEE Robot. Autom. Lett.*, vol. 8, no. 4, pp. 2325–2332, 2023.

[20] M. Saveriano, F. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic movement primitives in robotics: A tutorial survey," *Int. J. Robot. Res.*, vol. 42, no. 13, pp. 1133–1184, 2023.

[21] M. Noseworthy, R. Paul, S. Roy, D. Park, and N. Roy, "Task-conditioned variational autoencoders for learning movement primitives," in *Proc. Conf. on Robot Learning (CoRL)*, 2020, pp. 933–944.

[22] B. Lee, Y. Lee, S. Kim, M. Son, and F. Park, "Equivariant motion manifold primitives," in *Proc. Conf. on Robot Learning (CoRL)*, 2023, pp. 1199–1221.

[23] Y. Lee, "Mmp++: Motion manifold primitives with parametric curve models," *IEEE Trans. Robot.*, vol. 40, pp. 3950–3963, 2024.

[24] Y. Lee, B. Lee, S. Kim, and F. Park, "Motion manifold flow primitives for task-conditioned trajectory generation under complex task-motion dependencies," *IEEE Robot. Autom. Lett.*, vol. 10, no. 7, pp. 7412–7419, 2025.

[25] Y. Lee, "Differentiable motion manifold primitives for reactive motion generation under kinodynamic constraints," 2025.

[26] Y. Ma, Y. Liu, K. Qu, and M. Hutter, "Learning accurate whole-body throwing with high-frequency residual policy and pullback tube acceleration," 2025.

[27] Y. Liu and A. Billard, "Tube acceleration: Robust dexterous throwing against release uncertainty," *IEEE Trans. Robot.*, vol. 40, pp. 2831–2849, 2024.

[28] K. Muelling, J. Kober, and J. Peters, "Learning table tennis with a mixture of motor primitives," in *Proc. IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, 2010, pp. 411–416.

[29] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Robot. Res.*, vol. 32, no. 3, pp. 263–279, 2013.

[30] Y. Zhang, J. Luo, and K. Hauser, "Sampling-based motion planning with dynamic intermediate state objectives: Application to throwing," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2012, pp. 2551–2556.

[31] A. Pekarovskiy and M. Buss, "Optimal control goal manifolds for planar nonprehensile throwing," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2013, pp. 4518–4524.

[32] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[33] T. Kunz and M. Stilman, "Probabilistically complete kinodynamic planning for robot manipulators with acceleration limits," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2014, pp. 3713–3719.

[34] K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 2493–2498.

[35] Y. Lipman, R. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," 2023.

[36] X. Liu, C. Gong, and Q. Liu, "Flow straight and fast: Learning to generate and transfer data with rectified flow," 2022.

[37] M. Albergo, M. Goldstein, N. Boffi, R. Ranganath, and E. Vanden-Eijnden, "Stochastic interpolants with data-dependent couplings," 2024.

[38] Y. Lipman, M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat, "Flow matching guide and code," 2024.

[39] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022.

[40] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[41] G. Bradski, "The opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.