Learning from Demonstration via Probabilistic Diagrammatic Teaching

Weiming Zhi[1,*] Tianyi Zhang[1] Matthew Johnson-Roberson[1]

***Abstract*—Learning for Demonstration (LfD) enables robots to acquire new skills by imitating expert demonstrations, allowing users to communicate their instructions in an intuitive manner. Recent progress in LfD often relies on kinesthetic teaching or teleoperation as the medium for users to specify the demonstrations. Kinesthetic teaching requires physical handling of the robot, while teleoperation demands proficiency with additional hardware. This paper introduces an alternative paradigm for LfD called *Diagrammatic Teaching*. Diagrammatic Teaching aims to teach robots novel skills by prompting the user to sketch out demonstration trajectories on 2D images of the scene, these are then synthesised as a generative model of motion trajectories in 3D task space. Additionally, we present the Ray-tracing Probabilistic Trajectory Learning (RPTL) framework for Diagrammatic Teaching. RPTL extracts time-varying probability densities from the 2D sketches, applies ray-tracing to find corresponding regions in 3D Cartesian space, and fits a probabilistic model of motion trajectories to these regions. New motion trajectories, which mimic those sketched by the user, can then be generated. We validate our framework both in simulation and on real robots, which include a fixed-base manipulator and a quadruped-mounted manipulator.**

I. Introduction

Learning from Demonstration (LfD) enables robots to learn novel motions by mimicking a collected set of expert trajectories [1]. LfD is particularly appealing in its ability to specify complex robot movements in the absence of explicit programming or cost design, thereby empowering non-roboticists to teach a robot how to act. Demonstrations are typically collected via kinesthetic teaching where a human physically handles the robot, or via teleoperation where the expert uses a remote controller to collect demonstrations. These approaches can be limiting, as they may require co-location with the robot or proficiency with specialised hardware. These challenges are further amplified when attempting to provide demonstrations to mobile manipulators.

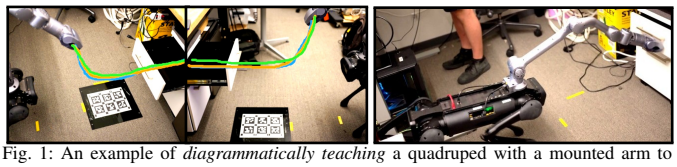

Fig. 1: An example of *diagrammatically teaching* a quadruped with a mounted arm to shut a drawer, by sketching robot demonstrations over 2D images. User sketches are shown in the left, middle.

This paper introduces *Diagrammatic Teaching*, as a paradigm for LfD where a small number of demonstrations are provided by the user through sketches over static two-dimensional images of the environment scene from different views. These images can either be captured by a camera or generated from a scene representation, such as a NeRF model [2]. Diagrammatic Teaching seeks to enable the teaching of new skills to the robot from the sketched demonstrations. We note that humans have a remarkable ability to infer instructions from crude diagrammatic sketches, and we wish to endow robots with this same capability.

Correspondingly, we present *Ray-tracing Probabilistic Trajectory Learning* (RPTL), a framework for Diagrammatic Teaching. RPTL extracts time-varying probability distributions from user-provided sketches over 2D images, and then uses ray tracing to fit a probabilistic distribution of continuous motion trajectories in the task space of a robot. This can be achieved with sketched demonstrations on as few as two images taken at different view poses. We can then adapt the trained model to generate trajectories that account for novel starting configurations. We present extensive evaluations of RPTL in simulation and on real-world robots.

II. Related Work

Learning for Demonstration (LfD) is broadly a strategy to teach robots novel skills by learning on a small set of expert demonstrations. Many attempts at LfD collect demonstrations by *Kinesthetic Teaching*, where the user physically handles the robot to show it the desired motion while recording the robot's joint or end-effector coordinates. These include Dynamical Movement Primitive (DMP) approaches [3], [4], Probabilistic Movement Primitives [5], and stable dynamical system approaches [6], [7], [8]. In these methods, demonstrations can be difficult to obtain as it requires the user to physically interact with the robot. Another approach for obtaining demonstrations is teleoperation, where a human provides demonstrations via a remote controller. This enables humans to provide demonstrations even when not co-located with the robot, allowing for larger demonstration datasets to be collected [9]. However, collecting trajectories via teleoperation is non-trivial and requires proficiency with the remote controller. Compared these approaches, the Diagrammatic Teaching introduced is distinct in enabling sketching as a medium for to specify demonstrations.

III. The Diagrammatic Teaching Problem

Diagrammatic teaching follows the typical setup for Learning for Demonstration: we are assumed to have a small dataset of demonstrated trajectories and seek to learn a generative model which produces trajectories to imitate the demonstrations. Diagrammatic Teaching is unique in that the trajectories

*email: wzhi@andrew.cmu.edu.
[1] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA

provided are **not** trajectories in the robot's Cartesian end-effector space nor its configuration space, but are instead 2D trajectories sketched onto images, while the desired generative model produces end-effector trajectories.

Formally, we have a dataset $\mathcal{D} = \{(v^j, \boldsymbol{\zeta}_i^j)_{i=1}^{n_j}\}_{j=1}^2$, where v^1 and v^2 denote two unique views from which the demonstrations are collected, and $\boldsymbol{\zeta}_i^j$ denotes the ith trajectory collected from the jth view. The user is shown images of the scene rendered from the views, and the user sketches how the end-effector movement is expected to look from the views. Images from different views may come from a NeRF scene representation or cameras at different locations. The collected *view space* trajectories $\boldsymbol{\zeta}$ contain sequences of normalised pixel coordinates (x, y) along with a normalised time t, i.e. $\boldsymbol{\zeta} = \{t_k, x_k, y_k\}_{k=1}^l$. The length of the trajectory is denoted by l. We assume that x, y, t are all normalised to be in $[0, 1]$.

We aim to learn the generative model of trajectories $p(\boldsymbol{\xi}|\mathcal{D})$, where $\boldsymbol{\xi}$ denotes motion trajectories in the robot's end-effector task space. Here, $\boldsymbol{\xi}$ are represented as functions that map from normalised time $t \in [0, 1]$ to Cartesian space $\mathbf{x} \in \mathbb{R}^3$. Figure 1 shows an example of applying Diagrammatic Teaching to teach a quadruped robot with a mounted manipulator to shut a draw by providing it demonstrations as sketches.

IV. RPTL

In this section, we propose Ray-tracing Probabilistic Trajectory Learning (RPTL), a framework to solve the Diagrammatic Teaching problem. The user is prompted to sketch trajectories onto images of the scene, which could be generated from a NeRF model or taken by cameras in multiple poses. The camera poses can be accurately estimated by tools such as COLMAP [10], [11], or with identifiable shared objects, such as an AprilTag [12], in the images taken. RPTL constructs time-varying density estimates over the collected 2D trajectories and then uses ray-tracing to find regions in 3D Cartesian space corresponding to the densities. Subsequently, a distribution of continuous-time motion trajectories can be fitted over these regions.

A. Density Estimation in View Space

We begin by estimating the time-varying densities over 2D coordinates of demonstrations from each view, which we denote as $p(x^j, y^j|t)$ for $j \in \{1, 2\}$. We use flexible normalizing flow models [13] to estimate the joint distribution $p^j(t, x^j, y^j)$, and assume that $p(t)$ is uniform over $[0, 1]$. A normalizing flow model uses neural networks to learn an invertible and differentiable function that transforms the arbitrarily complex data distribution into a known base distribution, typically a standard Gaussian. Let $\mathbf{y}^j = [t, x^j, y^j]$ be time-stamped coordinates from demonstrated trajectories from the jth view and $\hat{\mathbf{y}}^j \in \mathbb{R}^3$ be corresponding latent variables linked by invertible functions g^j, such that $\hat{\mathbf{y}}^j = g^j(\mathbf{y}^j)$. The densities over $\mathbf{y}^j$ and $\hat{\mathbf{y}}^j$ are linked by the change of variables,

$$p(\mathbf{y}^j) = p(g^j(\mathbf{y}^j))|\det \mathbf{J}^j(\mathbf{y}^j)|, \tag{1}$$

$$p(\hat{\mathbf{y}}^j) = p(g^{j^{-1}}(\hat{\mathbf{y}}^j))|\det \mathbf{J}^j(g^{j^{-1}}(\hat{\mathbf{y}}^j))|^{-1} \tag{2}$$

where $\det$ is the determinant and $\mathbf{J}^j$ is the Jacobian of g^j . We wish to learn g^j such that the distribution of latent variables matches a standard Gaussian, i.e. $p(\hat{\mathbf{y}}^j) = p(g^j(\mathbf{y}^j)) \approx \mathcal{N}(0, \mathbf{I})$. To ensure g^j is invertible, we model them as invertible neural network models described in [14], [15], with trainable parameters. We can then train g^j from each view, by minimising the negative log-likelihood of $g^j(\mathbf{y}^j)$ being drawn from a standard Gaussian over dataset $\mathcal{D}$. We arrive at the following loss [16]:

$$\mathcal{L} = -\sum_{\mathbf{y}^j \in \mathcal{D}} \left\{ \log p(g^j(\mathbf{y}^j)) + \log|\det \mathbf{J}(\mathbf{y}^j)| \right\}. \tag{3}$$

As the number of demonstrations is typically small, to prevent the densities from collapsing into a delta function during training, we inject a small Gaussian noise into the data. After training up normalizing flow density estimators for each view, we can obtain densities $p(x^j, y^j|t)$ over each view by evaluating eq. (1).

B. Trajectory Distribution Fitting via Ray-Tracing

We apply ray-tracing along the 2D pixels of high estimated density from both views to find regions in 3D space where the rays intersect. We can then fit our generative model onto these regions. In this section, we shall (1) introduce the parameterisation of our generative model, and then (2) elaborate on finding and fitting the model on corresponding regions. An example of learning to push a box is provided as fig. 2, where the view space density and the resulting distribution over trajectories are visualised.

Distributions of Trajectories: Generated motion trajectories are represented as functions $\boldsymbol{\xi} : t \to \mathbf{x}$, where $t \in [0, 1]$ is a normalised time variable and $\mathbf{x} \in \mathbb{R}^3$ is a corresponding location. Trajectories are modelled by a linear combination of M basis functions $\phi_i : t \to \mathbb{R}$ for $m = 1, \ldots, M$, and are parameterised by a matrix of weights $\mathbf{W} \in \mathbb{R}^{M \times 3}$. We have:

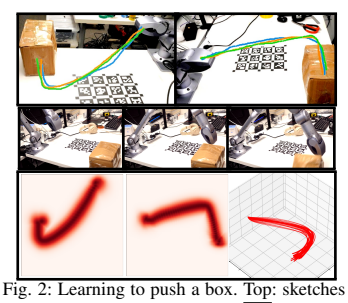

Fig. 2: Learning to push a box. Top: sketches over two different views. Middle: Robot executing learned skill. Bottom: Densities in 2D view space with time axis collapsed and trajectories in 3D task space.

$$\boldsymbol{\xi}(t) = \mathbf{W}^\top \Phi(t), \qquad \Phi(t) = [\phi_1(t), \ldots, \phi_M(t)]^\top \tag{4}$$

where Φ is a M-dimension feature vector with each basis function evaluated once for each spatial dimension in Cartesian space. The basis functions can be selected to enforce *a priori* assumptions on the motion trajectories, such as smoothness of periodicity. In this work, each basis is a squared exponential function that enforces the smoothness of motion, specifically, $\phi_i(t) = \exp(-\gamma||t - t_i||_2^2)$, for $i = 1, \ldots, M$, where γ is a length-scale hyper-parameter where smaller values correspond to smoother trajectories. The M times, t_i, are evenly spaced values between 0 and 1.

We can extend our parameterisation of a single trajectory to a *distribution* of trajectories $p(\boldsymbol{\xi})$, by estimating a distribution over the weight matrix $\mathbf{W}$. For tractability, we assume inde-

pendent Gaussian distributions over each element in $\mathbf{W}$. Let us denote each element in $\mathbf{W}$ as $w_{m,n}$ where $m = 1, \ldots, M$ and $n = 1, 2, 3$, then the joint distribution over $\mathbf{W}$ is simply the product of the distribution over each element,

$$p(\mathbf{W}) = \prod_{m=1}^{M} \prod_{n=1}^{3} p(w_{m,n}) = \prod_{m=1}^{m} \prod_{n=1}^{3} \mathcal{N}(\mu_{m,n}, \sigma^2_{m,n}), \quad (5)$$

where the means and standard deviations of the distribution over each $w_{m,n}$ are denoted as $\mu_{m,n}$ and $\sigma_{m,n}$. Fitting the distribution of trajectories involves finding each $\mu_{m,n}$ and $\sigma_{m,n}$ to match given data.

Ray-tracing from view space densities: Provided time-varying densities over pixels from different views, $p(x^j, y^j|t)$ for $j \in \{1, 2\}$, we wish to fit a trajectory distribution $p(\boldsymbol{\xi})$ by tracing the path of rays. We follow classical rendering methods [17] used for NeRF models [2] and assume pin-hole cameras at each view. We construct the ray in 3D which passes through each coordinate in 2D view space as, $f_r(d, x^j, y^j) = \mathbf{o}^j + \boldsymbol{\omega}(x^j, y^j)d$, where $\mathbf{o}^j$ is the origin of the camera, $\boldsymbol{\omega}(x^j, y^j)$ is a direction, and d is a distance bounded between d_{near} and d_{far}. These are given by the camera in use. From each view, the region corresponding to the density above threshold ϵ at a given time t is the codomain of f_r. We define this as the set:

$$R^j_t = \{\mathbf{x} \in \mathbb{R}^3 | f_r(d, x^j, y^j), \text{for all } d \in [d_{near}, d_{far}] \text{ and } x^j, y^j \in [0, 1], \text{such that } p(x^j, y^j|t) \geq \epsilon\}.$$

We seek the intersection of the 3D regions which corresponds to densities from both views, i.e. $R^1_t \cap R^2_t$. This can be approximated by sampling regular-spaced grid points over d and view space coordinates (x^j, y^j), for both views. If the distance between a sample from one view and its closest sample from the other view is below a specified distance threshold δ, then we consider both samples to be in the intersecting region. In practice, the nested loops are vectorised and can be efficiently executed on GPUs using deep learning frameworks, such as PyTorch [18]. We can obtain a set of n_s intersecting 3D spatial coordinates along the normalised time t, i.e. $\mathcal{S} = \{(t_i, \mathbf{x}_i)\}_{i=1}^{n_s}$.

To fit our generative model on $\mathcal{S}$, we need to compute the time-conditional distributions in Cartesian space, i.e. $p(\boldsymbol{\xi}|t)$. Let us first stack our train-able mean and variance parameters from eq. (5),

$$\mathcal{M} = \begin{bmatrix} \mu_{1,1} & \mu_{1,2} & \mu_{1,3} \\ \vdots & \vdots & \vdots \\ \mu_{M,1} & \mu_{M,2} & \mu_{M,3} \end{bmatrix} \quad \boldsymbol{\Lambda} = \begin{bmatrix} \sigma^2_{1,1} & \sigma^2_{1,2} & \sigma^2_{1,3} \\ \vdots & \vdots & \vdots \\ \sigma^2_{M,1} & \sigma^2_{M,2} & \sigma^2_{M,3} \end{bmatrix}. \quad (6)$$

As the weight distribution $p(\mathbf{W})$ is Gaussian and involves a linear transformation with Φ, as given in eq. (4), we have:

$$p(\boldsymbol{\xi}|t) = \mathcal{N}(\mathcal{M}^\top \Phi(t), \text{Diag}(\boldsymbol{\Lambda}^\top \Phi(t)^2)), \quad (7)$$

where $\text{Diag}(\cdot)$ produces a square matrix with the inputted vector as its diagonal. We minimise the Gaussian negative log-likelihood of $p(\boldsymbol{\xi}|t)$ over $\mathcal{S}$ to fit parameters $\mathcal{M}$, $\boldsymbol{\Lambda}$.

C. Trajectory Generation

After training to obtain fitted means, $\mathcal{M}$, and variances, $\boldsymbol{\Lambda}$, we can generate a collection of trajectories by sampling elements in $\mathbf{W}$ from $w_{i,j} \sim \mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$, and then evaluate eq. (4). However, often times we have additional knowledge of where the trajectory shall be at the starting position. For example, the generated task space trajectory at the initial time should match its current end-effector position ($\mathbf{x}_{eef}$), i.e. enforcing, $\boldsymbol{\xi}(0) = \mathbf{x}_{eef}$.

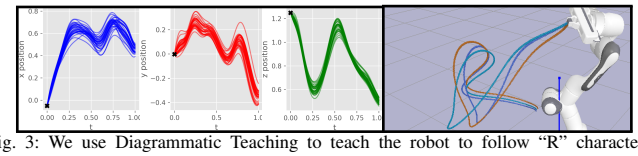

Fig. 3: We use Diagrammatic Teaching to teach the robot to follow "R" characters. Left: The x, y, z positions of sampled trajectories over normalised time t. The initial positions of the trajectory samples are enforced at the black marker. Right: Three end-effector trajectories, conditioned to start from the current position.

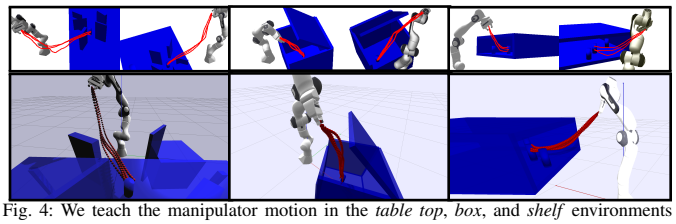

Fig. 4: We teach the manipulator motion in the *table top*, *box*, and *shelf* environments (left, middle, right respectively). The top row images are two camera views provided to the user, and the red trajectories are 2D demonstrations sketched by the user. We draw five 3D motion trajectories from the trajectory distribution model and illustrate them in red in the images on the bottom row.

Let us begin by defining notation: let $\mathbf{W}_1$ be the first row of $\mathbf{W}$, and $\mathbf{W}_{2:}$ be the others; let $\Phi(0)_1$ be the first element in $\Phi(0)$ and $\Phi(0)_{2:}$ denote the others. For a specific sampled trajectory, we draw a sample of $\mathbf{W}$ and then alter $\mathbf{W}_1$ from the enforced condition. Specifically, at the beginning of the trajectory, we wish to enforce $\boldsymbol{\xi}(\mathbf{0}) = \mathbf{W}_1^\top \Phi(0)_1 + \mathbf{W}_{2:}^\top \Phi(0)_{2:} = \mathbf{x}_{eef}$, allowing us to solve for $\mathbf{W}_1$, together with the known $\mathbf{W}_{2:}$. An example of generated trajectories conforming to starting at the current end-effector position is shown in fig. 3.

V. Experimental Evaluation

We explore the performance of RPTL for Diagrammatic Teaching. Specifically, we examine the quality of the trained model, checking if motion produced is consistent with user expectations. Then, we examine using RPTL to teach the tracing of challenging alphabet characters. Finally, we test the robustness of RPTL on real robots.

A. Quality and Consistency of RPTL

We set up three simulated environments with a Franka manipulator, following the *table top*, *box*, and *shelf* environment types in [19], using the physics simulator PyBullet [20]. We place cameras at three different poses in the environment and ask the user to provide three demonstrations per camera view to reach a cylindrical goal object. The goal position is not given to the model, and the only specifications given are the diagrammatic demonstrations. We select two views and their demonstrations for training and retain those from the third view as a test set. To benchmark motion trajectories generated, we project the 3D trajectories into the 2D view of the third and compute distances between the projected trajectories and the retained test trajectories. Distances computed are all normalised by the width of the test image. By testing against a hidden test set, we can access whether the produced motions

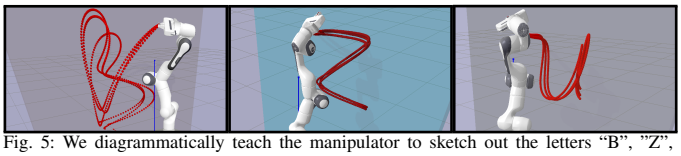

Fig. 5: We diagrammatically teach the manipulator to sketch out the letters "B", "Z", "U", and sample three trajectories from the trained model.

		Table	Box	Shelf
RPTL (Ours)	MFD ($\times 10^{-2}$)	$\mathbf{3.1 \pm 0.2}$	$\mathbf{3.9 \pm 0.2}$	$\mathbf{5.3 \pm 0.9}$
	WD ($\times 10^{-2}$)	**2.6**	**2.9**	**3.8**
Linear	MFD ($\times 10^{-2}$)	10.6 ± 0.3	7.4 ± 0.1	9.7 ± 1.0
	WD ($\times 10^{-2}$)	6.9	6.9	6.3
NN	MFD ($\times 10^{-2}$)	17.8 ± 0.8	33.2 ± 0.6	7.9 ± 0.9
	WD ($\times 10^{-2}$)	10.3	19.1	9.2

TABLE I: Comparison of RPTL and baselines on different environments. Lower distances indicate better performance. Lowest in bold.

match the expectation of the user and are consistent when viewed from a different pose.

Metrics: We use the following metrics to measure the quality of the learned trajectory distribution. (1) *Mean Fréchet distance (MFD)*: We compute the discrete Fréchet distance [21] between the mean of the trajectory distribution model and each of the test trajectory, then record the averages and standard deviations. The Fréchet distance measures the distance between curves and can account for the ordering of points and handle varying lengths. (2) *Wasserstein Distance (WD)*: We compute the 2-Wasserstein distance implemented in [22] between five trajectories drawn from our model and the set of test trajectories. Crucially, the WD can measure distances between distributions beyond simply considering the mean of the probabilistic model. **Baselines:** We evaluate our model with respect to the following baseline models. (1) *Linear*: We provide the mean start and end positions of the test trajectories and assume a linear curve between them. (2) *Nearest Neighbour (NN)*: We predict the trajectories as those taken from the nearest camera in the training set. The quantitative performance of our method, along with baseline comparisons, are given in table I. The low distances between our learned model and the test set highlight the ability of our method to train a generative model that is consistent with the trajectories a human user would expect. This is reiterated when qualitatively examining fig. 4, where the top row consists of two images along with user-sketched trajectories at different views, and the bottom rows contain samples from the 3D trajectory model. We observe the produced end-effector motions are highly consistent with the sketches provided.

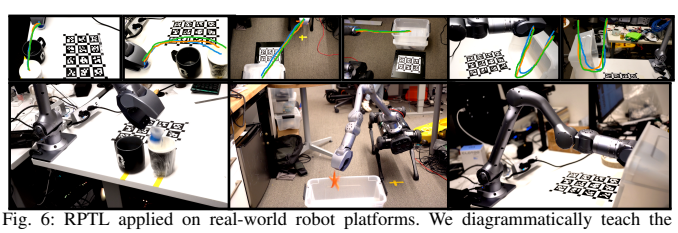

Fig. 6: RPTL applied on real-world robot platforms. We diagrammatically teach the robot for the tasks **Drop into cup**, **Drop into box**, and **Tip box** (left, middle, right respectively). The sketched demonstrations are shown in the top row. The task **Close drawer** is shown in fig. 1 and **Push box** is shown in fig. 2.

B. Tracing Out Letters: a Case of Intricate Trajectories

We seek to investigate whether RPTL can be applied to learn more complex distributions of motion trajectories which may involve multiple turns or swerves. To this end, we diagrammatically teach a simulated Franka to trace out the letters "B", "Z", "U". This requires motion that deviates greatly from a linear trajectory and would be challenging to describe with a motion planning cost function. We randomly select different robot starting configurations facing a surface, and sample motion trajectories to sketch out the letters on the surfaces. These are illustrated in fig. 5. An additional tracing of the letter "R" is shown in fig. 3 We observe that RPTL is able to generate trajectories that accurately capture the intricacies of each of the characters.

C. Diagrammatic Teaching in the Real-world

To test the robustness of RPTL in the real world, we diagrammatically teach a 6-DOF Unitree Z1 manipulator new skills. Additionally, we also demonstrate the applicability of RPTL to a Unitree Aliengo quadruped with the Z1 arm attached. We take two photos with a camera, find the camera poses via AprilTags [12], and collect demonstrations from the user. The robot end-effector, along with the quadruped, then tracks trajectories sampled from the trained models.

We diagrammatically teach skills for the following tasks to the Z1 arm: **Push box**: Push a box from its side, such that it drops off the table; **Drop into cup**: Move past a mug, and hover right over a paper cup. The gripper is subsequently released and the held object shall be dropped into the paper cup; **Tip box**: Reach into a box and tip it by moving in a "U"-shaped motion. We then mount the arm onto a quadruped and teach the quadruped + arm robot skills for the following tasks: **Drop into box**: Reach out towards a box on the floor. The gripper is released and the object held is placed into the box; **Close drawer**: Close an open drawer.

Images of the provided demonstrations and subsequent execution are given in figs. 1, 2 and 6. We observe that RPTL can robustly teach the robot the specified skills. Moreover, Diagrammatic Teaching demonstrates its utility on mobile manipulators where kinesthetic teaching would be impractical. In particular, the task **Drop into box** requires the dog to bend its knees while the arm moves towards the box for the mounted arm to reach sufficiently low. The task **Close drawer** requires even more coordination between the dog and the arm, as the dog needs to concurrently march forward towards the drawer as the arm shuts the drawer.

VI. Conclusions

We introduce Diagrammatic Teaching, with which a user specifies desired motion via sketched curves, as a novel paradigm for learning from demonstration. We present Ray-tracing Probabilistic Trajectory Learning (RTPL) as a framework for Diagrammatic Teaching. RPTL estimates probability densities over 2D user sketches and traces rays into the robot's task space to fit a distribution of motion trajectories. We evaluate RTPL in both simulation and on real-world robots.

REFERENCES

[1] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters. 2018.

[2] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

[3] J. Kober and J. Peters, "Learning motor primitives for robotics," in *IEEE International Conference on Robotics and Automation*, 2009.

[4] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, 2013.

[5] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic movement primitives," in *Proceedings of the 26th International Conference on Neural Information Processing Systems*, 2013.

[6] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, 2011.

[7] S. M. Khansari-Zadeh and A. Billard, "Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions," *Robotics Auton. Syst.*, vol. 62, pp. 752–765, 2014.

[8] W. Zhi, T. Lai, L. Ott, and F. Ramos, "Diffeomorphic transforms for generalised imitation learning," in *Learning for Dynamics and Control Conference, L4DC*, 2022.

[9] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, S. Savarese, and L. Fei-Fei, "Roboturk: A crowdsourcing platform for robotic skill learning through imitation," in *Conference on Robot Learning*, 2018.

[10] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixel-wise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[12] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *IEEE International Conference on Robotics and Automation*, 2011.

[13] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *J. Mach. Learn. Res.*, 2021.

[14] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *International Conference on Learning Representations*, 2017.

[15] L. Ardizzone, J. Kruse, C. Rother, and U. Köthe, "Analyzing inverse problems with invertible neural networks," in *International Conference on Learning Representations*, 2019.

[16] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[17] J. Fong, M. Wrenninge, C. Kulla, and R. Habel, "Production volume rendering: Siggraph 2017 course," in *ACM SIGGRAPH 2017 Courses*, SIGGRAPH '17, 2017.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019.

[19] C. Chamzas, C. Quintero-Peña, Z. K. Kingston, A. Orthey, D. Rakita, M. Gleicher, M. Toussaint, and L. E. Kavraki, "Motionbenchmaker: A tool to generate and benchmark motion planning datasets," *IEEE Robotics and Automation Letters*, 2022.

[20] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning." http://pybullet.org, 2016–2019.

[21] T. Eiter and H. Mannila, "Computing discrete fréchet distance," *Technical Report CD-TR 94/64*, 1994.

[22] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer, "Pot: Python optimal transport," *Journal of Machine Learning Research*, vol. 22, no. 78, pp. 1–8, 2021.