23 DoF Grasping Policies from a Raw Point Cloud

Martin Matak[1], Karl Van Wyk[2], Tucker Hermans[1,2]

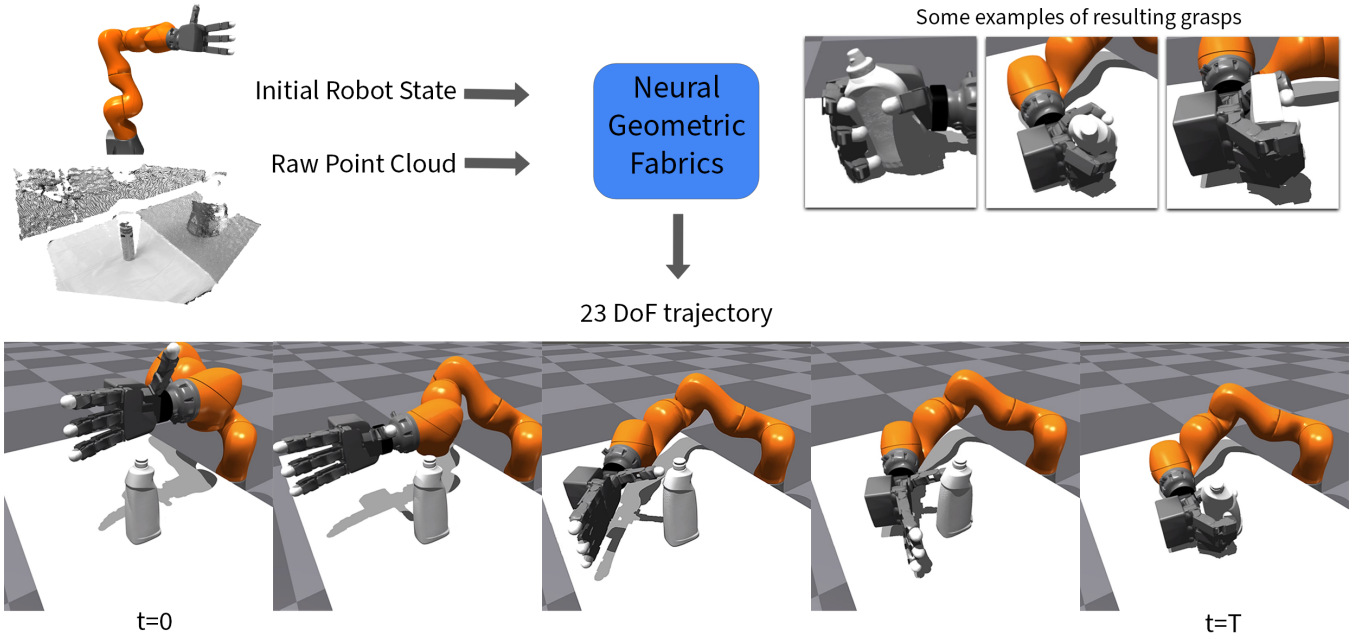

Fig. 1: Our approach predicts 23 DoF trajectories based on raw point cloud and robot state (position and velocity) to grasp the object of interest. Some examples of resulting grasps are shown in top right. An example of a generated trajectory is shown in the bottom part of the figure.

***Abstract*— Coordinating the motion of robots with high degrees of freedom (DoF) to grasp objects gives rise to many challenges. In this paper, we propose a novel imitation learning approach to learn a policy that directly predicts 23 DoF grasp trajectories from a partial point cloud provided by a single, fixed camera. At the core of the approach is a second-order geometric-based model of behavioral dynamics. This Neural Geometric Fabric (NGF) policy predicts accelerations directly in joint space. We show that our policy is capable of generalizing to novel objects, and combine our policy with a geometric fabric motion planner in a loop to generate stable grasping trajectories. We evaluate our approach on a set of three different objects, compare different policy structures, and run ablation studies to understand the importance of different object encodings for policy learning.**

I. Introduction

Autonomous grasping with multi-fingered hands has the potential to enable high throughput, generalized pick-place operations across a wide range of objects. However, achieving high performance in this domain is difficult due to the high-dimensional embodiments, sensory noise, partial observability, and large variation in object dynamics. Existing strategies for computing grasping trajectories are typically an outcome of a complex optimization on multiple levels of the stack [1]–[15]. While optimization-based approaches might lead to high grasp success rates, they suffer from being computationally expensive, especially in these high-dimensional, continuous action spaces.

One way to overcome this limitation is the use of geometrically aware policy structures for control in high-dimensional continuous spaces. Such approaches include Dynamic Movement Primitives (DMPs) [16], Riemannian Motion Policies (RMPs) [17], Geometric Dynamics Systems (a stable subclass of RMPs) [18], Operational Space Control (OSC) [19], and geometric fabrics [20]. Geometric fabrics are nonlinear, second-order differential equations that are provably stable and have been shown to outperform these existing control methods in learning contexts [21]. While fabrics have been used for grasping [21], [22], they have not yet been shown to generalize to novel object geometries from a single camera view. Xie et al. [21] leverage estimates of object pose to train object-specific grasping policies that mimic human demonstrations. Chen et al. [22] leverage a manually derived geometric fabric to move the robot to a single predicted grasp palm pose and hand configuration.

The main contribution of this paper is a model that reliably predicts a smooth 23-DoF grasping trajectory directly in joint space to grasp a previously unseen object given only a single RGBD camera view, when the robot is placed in the general vicinity (not a specific grasp target) of the object for grasping. We create an NGF policy to learn this behavior, building upon the prior success of NGFs [21]. An example trajectory is shown in Fig. 1. Our results show that NGFs produce smooth and stable behavior that generalizes over

[1]School of Computing and the Robotics Center, University of Utah. [2]NVIDIA, USA. martin.matak@utah.edu

object shape and pose variation. An off-the-shelf motion planner drives the robot to the general vicinity of the object, alleviating the burden of learning how to grasp from arbitrary far away, initial robot configurations.

II. Imitation Learning Setup

We aim to learn policies that generate trajectories to grasp objects of interest given the current state of the robot, an object encoding $\mathbf{o}$, and an initial robot placement in the vicinity of the object. We train these policies via imitation learning by first generating a dataset of successful grasping trajectories in simulation using geometric fabrics [20] and subsequently construct a surrogate expert as a function of the data. This allows for on-policy imitation learning, DAgger [23], to train our NGF policy.

A. Data Collection

Data collection consists of two primary steps: (1) finding grasps and (2) generating robust grasping trajectories.

Finding in-contact grasps We place an object onto a table and try to find a grasp that lifts the object successfully. To do so, we use a dataset of grasps collected by [24]. After filtering out the poses for reachability and collision, we move the robot's EE to the remaining poses, one by one, until a successful grasp is found or all poses are exhausted. The preshape hand configuration is fixed and a simple heuristic is used to close the hand. If the grasp is successful, we store it in a set of successful grasps. We repeat this process for different objects and poses.

Next, we cluster the successful grasps based on EE pose in the object frame and save only N grasp poses per object ($N = 7$). We do this step hoping to have a low variance in grasp distribution, which eases the learning process. Then, we place the object across the grid on the table and try out the N grasps one by one until we find a successful one, which we store in the dataset $\mathcal{D}_g$ we use to generate training trajectories. Here $\mathcal{D}_g = \{(\mathbf{q}^g, \mathbf{o})_i\}_{i=1}^{i=|\mathcal{G}|\times|\mathcal{O}|}$ where $\mathcal{G}$ is set of poses on the grid, $\mathcal{O}$ is set of objects, $\mathbf{q}^g$ is robot joint position and $\mathbf{o}$ is object encoding.

Generating trajectories We initialize the scene in a configuration $(\mathbf{q}^g, \mathbf{o})_i$ from the dataset and sample a target pose $\mathbf{x_0}$ for the EE above the object in a prespecified region, as shown in Fig. 2. This region is a design choice and any other region could be used instead. Then, we use geometric fabrics [20] as a motion planner to move the robot from the grasp configuration $\mathbf{q}^g$ to the target pose $\mathbf{x_0}$, resulting in a trajectory $\overleftarrow{\tau_i} = [(\mathbf{q}^g, \dot{\mathbf{q}}^g), ..., (\mathbf{q}^0, \dot{\mathbf{q}}^0)]$. Importantly, here we add the object as an obstacle when computing a trajectory to the target pose to ensure collision free trajectory. Finally, we reverse the generated trajectory to store a trajectory that moves into the grasp configuration, $\overrightarrow{\tau_i} = \texttt{reverse}(\overleftarrow{\tau_i})$. We generate M trajectories per object pose ($M = 256$). We further encode the object's partial view point cloud $\mathbf{z} = F_\theta(\mathbf{o})$ where F_θ is the encoder described in Section III-D. This results in a dataset $\mathcal{D}_\tau = \{(\overrightarrow{\tau_m}, \mathbf{z}_i)\}_{i=1,m=1}^{i=K,m=M}$ where $|\mathcal{D}_\tau| = MK$ that we use for training our policies using imitation learning.

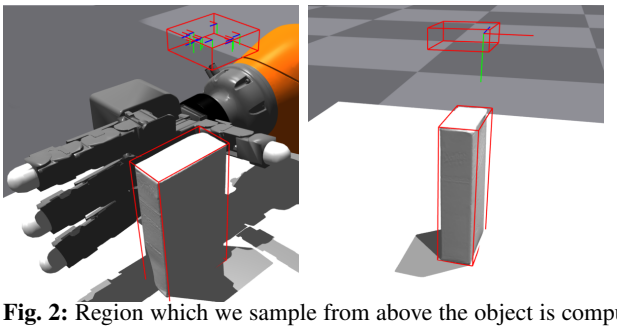

Fig. 2: Region which we sample from above the object is computed based on the object's bounding box. **Left:** Training data is collected by generating trajectories from the grasp configuration to the sampled poses and then reversing those trajectories. **Right:** At evaluation time, we sample a target pose above the object and use a motion planner to get there. Then, the policy predicts a trajectory to grasp the object.

B. Surrogate Expert and Training

With the above dataset $\mathcal{D}_\tau$, we now construct a surrogate expert that can be efficiently queried on-the-fly, enabling DAgger-style imitation learning. First, we select a randomly sampled configuration, $\mathbf{q}_t^d, \dot{\mathbf{q}}_t^d$, from trajectory $\overrightarrow{\tau_k}$ at time index t in our dataset. Then, the surrogate expert, $\pi_e(\cdot)$, uses a PD controller to compute an acceleration action to steer towards the next configuration along the trajectory, $(\mathbf{q}_{t+1}^d, \dot{\mathbf{q}}_{t+1}^d) \in \overrightarrow{\tau_k}$. With a sufficient number of consecutive executions, π_e will produce motion that collapses onto trajectory $\overrightarrow{\tau_k}$ from the dataset. Formally, this expert is:

$$\pi_e(\mathbf{q}_{t+1}^d, \mathbf{q}_t, \dot{\mathbf{q}}_t) = k_p(\mathbf{q}_{t+1}^d - \mathbf{q}_t) - k_d\dot{\mathbf{q}}_t \tag{1}$$

where $\mathbf{q}_t$ and $\dot{\mathbf{q}}_t$ are the current joint position and velocity. With this expert in place, we train policy π_θ online to predict desired accelerations following the optimization problem

$$\theta^* = \arg\min_\theta \quad ||\pi_\theta(\mathbf{q}_t, \dot{\mathbf{q}}_t) - \pi_e(\mathbf{q}_t^d, \mathbf{q}_t, \dot{\mathbf{q}}_t)|| \tag{2}$$

where $||\cdot||$ is L2 norm and this loss function is calculated over a batch of evaluations before updating the policy parameters. In practice, after sampling $\mathbf{q}_t^d, \dot{\mathbf{q}}_t^d$, we add ϵ noise from a narrow uniform distribution to obtain $\mathbf{q}_t, \dot{\mathbf{q}}_t$. We then identify the trajectory $\overrightarrow{\tau_g} \in \mathcal{D}_\tau$ that has the closest configuration $\mathbf{q}_i^d, \dot{\mathbf{q}}_i^d$ (i is some time index) to $\mathbf{q}_t, \dot{\mathbf{q}}_t$. The expert π_e then targets subsequent configurations in this identified trajectory $\overrightarrow{\tau_g}$ for the remainder of the rollout of the policy.

III. Neural Policy Architectures

In this section, we provide a short introduction to geometric fabrics [20] and describe our instantiated NGF policy. We additionally cover the baseline policy, discuss object encodings, numerical integrators for our policies, and connect our policies with off-the-shelf motion planners.

A. Geometric Fabrics

Geometric fabrics are a specific class of autonomous, second-order differential equations that are provably stable (i.e., they are guaranteed to come to rest at a local minimum) and exhibit path consistency in the motion they generate due to their geometric nature. They consist of three major

components: 1) an energized geometry, 2) a driving force that derives from a scalar potential function, and 3) damping. Geometric fabrics are dependent on joint position $\mathbf{q}$, joint velocity $\dot{\mathbf{q}}$, and an additional feature vector $\mathbf{z}$, that together form the following system,

$$\begin{aligned}\ddot{\mathbf{q}} =& \widetilde{\mathbf{h}}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{z}) + \alpha_L \dot{\mathbf{q}} - \mathbf{M}^{-1}(\mathbf{q}, \hat{\dot{\mathbf{q}}}, \mathbf{z}) \partial_{\mathbf{q}} \psi(\mathbf{q}, \mathbf{z}) \\ & - \mathbf{B}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{z}) \dot{\mathbf{q}} - \beta \dot{\mathbf{q}}, \end{aligned} \tag{3}$$

where $\widetilde{\mathbf{h}}$ is a nonlinear geometry (homogeneous of degree 2 in velocity), α_L is an energization coefficient calculated to conserve some particular energy, $\mathbf{M}^{-1}\partial_{\mathbf{q}}\psi$ creates the driving force and dictates the local minima of the fabric, $\mathbf{B}$ is a postive semi-definite damping matrix, $\hat{\cdot}$ denotes a unit vector, and $\beta \in \mathbb{R}^+$ is an additional damping scalar. Full description of these components are outside the scope of this paper and we refer the reader to [25] for in-depth discussion on generalized nonlinear geometries and [20] for geometric fabrics and relevant tools like the energization operator.

B. Neural Geometric Fabric

We instantiate a Neural Geometric Fabric by parameterizing the various components of a geometric fabric via neural networks and object encoding as the feature vector $\mathbf{z}$. For the following, $\mathbf{F}_\theta(\cdot)$ is a neural network consisting of three fully connected, feedforward layers with 512 units each and with hidden layer ELU activations, θ trainable parameters, and 1000 Random Fourier Features (RFFs) over its inputs. Every usage of $\mathbf{F}_\theta(\cdot)$ below is meant as a different neural network instance. For some outputs like the damping coefficient below, the final output layer is also passed through ELU activations to force positive outputs. Otherwise, the final layer is linear.

The geometric portion of the fabric is produced by the metric, $\mathbf{M}_g = \mathbf{U}_g \mathbf{U}_g^T$, with $\mathbf{U}_g = \mathbf{F}_g(\mathbf{q}, \hat{\dot{\mathbf{q}}}, \mathbf{z})$, a lower triangular matrix with positive diagonal elements. The geometric acceleration results from $\pi_g = \dot{\mathbf{q}}^T \dot{\mathbf{q}}\, \mathbf{F}_{\mathcal{X}}(\mathbf{q}, \hat{\dot{\mathbf{q}}}, \mathbf{z})$. The final geometric force is then $\mathbf{f}_g = \mathbf{M}_g \pi_g$.

The driving force of the fabric is produced by the metric, $\mathbf{M}_f = \mathbf{U}_f \mathbf{U}_f^T$, with $\mathbf{U}_f = \mathbf{F}_f(\mathbf{q}, \mathbf{z})$, a lower triangular matrix with positive diagonal elements. The associated scalar potential force results from taking the gradient $\partial_{\mathbf{q}}\psi$ of scalar function $\psi = \mathbf{F}_\psi(\mathbf{q}, \mathbf{z})$. This force is coupled with a learned positive damping scalar, $\beta_f = \mathbf{F}_\beta(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{z})$. The final driving force is then $\mathbf{f}_f = \partial_{\mathbf{q}}\psi + \beta_f \dot{\mathbf{q}}$.

The metrics and forces are summed to produce

$$(\mathbf{M}_g + \mathbf{M}_f)\ddot{\mathbf{q}} + \mathbf{f}_g + \mathbf{f}_f = \mathbf{0}, \tag{4}$$

which can be resolved as

$$\ddot{\mathbf{q}} = \widetilde{\mathbf{h}} - \mathbf{M}^{-1}\partial_{\mathbf{q}}\psi - \mathbf{B}\dot{\mathbf{q}} \tag{5}$$

where $\mathbf{M} = \mathbf{M}_g + \mathbf{M}_f$, $\widetilde{\mathbf{h}} = -\mathbf{M}^{-1}\mathbf{f}_g$, and $\mathbf{B} = \beta_f \mathbf{M}^{-1}\dot{\mathbf{q}}$. We calculate an energization coefficient, α_L, using the energy $L = \frac{1}{2}\dot{\mathbf{q}}^T\dot{\mathbf{q}}$ and geometric acceleration $\ddot{\mathbf{q}} = \widetilde{\mathbf{h}}$, and add this additional acceleration as $\alpha_L \dot{\mathbf{q}}$. Finally, we add an additional positive scalar damping with $\beta = 5$ resulting in the final fabric form in (3). Practically, this additional damping helped stabilize training.

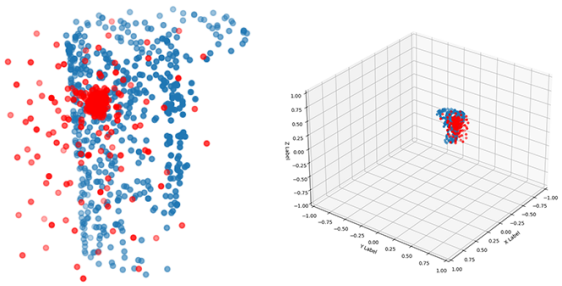

Fig. 3: Example point cloud reconstruction; input in blue, output in red. While imperfect (left), it preserves information about object pose and course geometry (right).

C. Acceleration MLP

We implement an MLP-based acceleration policy as a baseline. Input and output of the policy is the same as for NGF, i.e. input joint position, velocity, and object encoding and output joint acceleration. The input layer is followed by two hidden layers before the output layer. The first hidden layer has a dimensionality of 64, while the second one has 256. The ReLU activation function is used after each layer.

D. Object encoding

We aim to generate grasp trajectories that generalize across poses and object shapes directly from a point cloud. Thus, we need to encode the object's shape and pose. We do so by training an autoencoder on a reconstruction task for pointclouds stored in robot base frame and then use only the encoder from the autoencoder to generate an object encoding. After training on the reconstruction task, the encoder is not further modified, i.e., the weights are frozen. The loss function for the reconstruction task is a weighted sum of Chamfer distance and L2 regularization on the latent space. The data for the reconstruction task is generated by placing an object at 9k different poses across the table and saving the point cloud obtained from the fixed camera. We repeat this process for 4 YCB [26] objects, producing in total 36k samples. The encoder is based on PointNet layers [27] from PyTorch Geometric library [28] while the decoder is a 3 layer MLP. An example of a reconstructed point cloud can be seen in Fig. 3. While imperfect, we show in our experiments that it is sufficiently good for generating grasp trajectories.

E. Acceleration Integrator

Both the learned policies and the surrogate expert described in Sec. II-B produce accelerations. We integrate these accelerations forward in time via the approximate RK2 integration scheme as described in [29]. The exact formulation calculates the next joint position and velocity, $\mathbf{q}_{t+1}$ and $\dot{\mathbf{q}}_{t+1}$, from the current joint position and velocity, $\mathbf{q}_t$ and $\dot{\mathbf{q}}_t$, acceleration $\ddot{\mathbf{q}}_t$, and timestep Δt as

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \Delta t \dot{\mathbf{q}}_t + \frac{1}{2}\Delta t^2 \ddot{\mathbf{q}}_t \tag{6}$$

$$\dot{\mathbf{q}}_{t+1} = \dot{\mathbf{q}}_t + \Delta t \ddot{\mathbf{q}}_t \tag{7}$$

The policies perform control at 30 Hz and therefore $\Delta t = \frac{1}{30}$.

F. Motion Planner In The Loop

A further shortcoming of the previous work [21] is a necessity to always start from a precise initial configuration. While the ability to learn longer trajectories showcases the model's capabilities, we believe one should use existing (traditional) tools when a solution is well known and use learning models only when necessary. To underline this, we leverage an existing off-the-shelf motion planner to move close to the object and lift it after the grasping is completed. The learned model learns only the complex component - grasping the object.

When deploying our model, we sample a pose above the object (Fig. 2) in the same way as we do when generating the trajectories during training. Then, instead of solving for IK, we look for the closest target configuration in the training dataset. To find such a configuration, we find an encoding in the dataset that is closest (L2) to the encoding of the current object we want to grasp. Then, we search across trajectories associated with that encoding and find the one whose EE pose starts closest to the queried pose above the object. We set the corresponding configuration as the target configuration for the motion planner. We found that if we would naively solve IK instead, the deployed policy wouldn't perform as well due to the out-of-distribution robot configurations. We plan on learning this as a separate model for future work since this querying can be expensive if the whole dataset can't fit onto a GPU.

IV. Experiments

We place an object at a random position on the table and simply fit a spline as a motion planner to get from a fixed initial configuration close to the object. After that, we execute a grasping policy to grasp the object. Finally, we use a motion planner [20] to lift the object. If the robot holds the object after executing the lifting trajectory, we label the attempt as a successful grasp. We repeat this 100 times per object. Our experimental setup is shown in Fig. 4.

We evaluate our approach across 3 different objects. The grasp policies have been trained on grasp trajectories for 2 (bleach, sugarbox) out of the 3 objects. Results are shown in Fig. 5 with discussions below. We note that when we ran the trajectories from the dataset in the simulator, only 59% of trajectories successfully picked up 'sugarbox' while that number is 93% for 'bleach.' We think it is important to keep these numbers in mind when reading the results in Fig. 5.

Effect of Policy Structure: When presented with the same object encoding, NGF always outperforms MLP policy across all objects. Most importantly, when a model is presented with an encoded pointcloud, NGF outperforms MLP.

Object Representation: We investigate the impact of object encoding on policy performance across two cases: 1) passing the position (-POS) of the object to the policy, and 2) passing the latent point cloud encoding (-PCD) to the policy. Overall, NGF performance with object position is fairly competitive to the MLP baseline. However, the PCD encoding does improve the NGF policy performance over the position only input. These results reveal that the latent encoder for the *partial* view point cloud produces useful features for grasping operations that meet or exceed position-only policy performance. The main reason for this difference is that the point cloud contains information about the geometry of the object while the position only input does not. This additional information is useful for informing grasp behavior. Overall, this result is encouraging and suggests that effective high-DoF, high-frequency policies can be trained that leverage camera views, which contain only partial information about the scene. This capability will enable policies to be immediately more useful for real-world deployment.

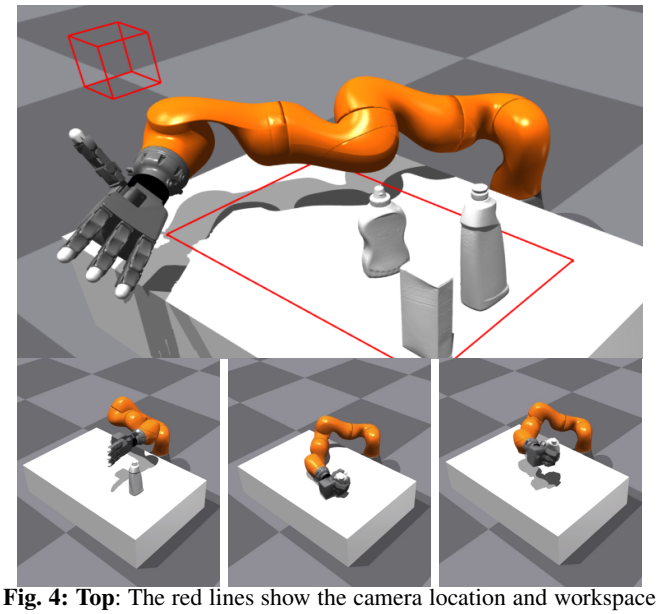

Fig. 4: Top: The red lines show the camera location and workspace boundaries. The objects are (left to right): 'mustard', 'sugarbox', 'bleach'. **Bottom**: Our manipulation pipeline from left to right: a motion planner moves the robot close to the object, our policy grasps the object, and then a motion planner lifts the object.

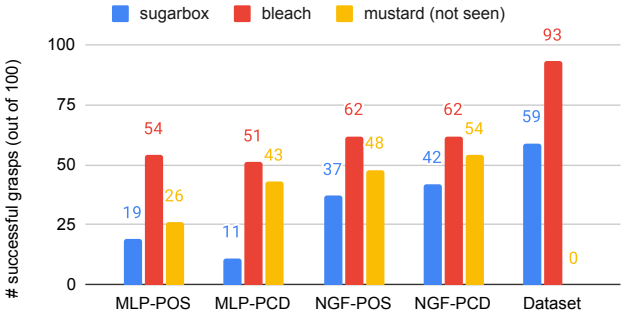

Fig. 5: Grasp success rates for different grasp-trajectory predicting models and the dataset we use for training. Not all samples in the dataset are successful grasps. Input modalities: POS-position only, PCD - pointcloud.

V. Conclusion

We show how to train an NGF policy without any human demonstrations and how to use the policy in a loop with an off-the-shelf motion planner. Our policy generates 23 DoF grasp trajectories that generalize across previously unseen objects directly from a partial view point cloud.

Acknowledgment

This work was supported in part by NSF Award #1846341.

REFERENCES

[1] M. T. Ciocarlie and P. K. Allen, "Hand posture subspaces for dexterous robotic grasping," *Intl. Journal of Robotics Research*, vol. 28, no. 7, 2009.

[2] D. Chen, V. Dietrich, Z. Liu, and G. von Wichert, "A probabilistic framework for uncertainty-aware high-accuracy precision grasping of unknown objects," *Journal of Intelligent & Robotic Systems*, vol. 90, no. 1, 2018.

[3] A. Miller and P. Allen, "Examples of 3d grasp quality computations," in *IEEE Intl. Conf. on Robotics and Automation*, vol. 2, 1999, pp. 1240–1246 vol.2.

[4] M. A. Roa and R. Suarez, "Computation of independent contact regions for grasping 3-d objects," *IEEE Trans. on Robotics*, vol. 25, no. 4, 2009.

[5] C. Rosales, L. Ros, J. M. Porta, and R. Suárez, "Synthesizing grasp configurations with specified contact regions," *Intl. Journal of Robotics Research*, vol. 30, no. 4, 2011.

[6] Y. Zheng and W.-H. Qian, "Coping with the grasping uncertainties in force-closure analysis," *Intl. Journal of Robotics Research*, vol. 24, no. 4, 2005.

[7] K. Hang, M. Li, J. A. Stork, Y. Bekiroglu, F. T. Pokorny, A. Billard, and D. Kragic, "Hierarchical fingertip space: A unified framework for grasp planning and in-hand grasp adaptation," *IEEE Trans. on Robotics*, vol. 32, no. 4, 2016.

[8] M. S. Siddiqui, C. Coppola, G. Solak, and L. Jamone, "Grasp stability prediction for a dexterous robotic hand combining depth vision and haptic bayesian exploration," *Frontiers in Robotics and AI*, 2021.

[9] Q. Lu, K. Chenna, B. Sundaralingam, and T. Hermans, "Planning Multi-Fingered Grasps as Probabilistic Inference in a Learned Deep Network," in *Intl. Symposium on Robotics Research*, 2017.

[10] Q. Lu and T. Hermans, "Modeling Grasp Type Improves Learning-Based Grasp Planning," *IEEE Robotics and Automation Letters*, 2019.

[11] Q. Lu, M. V. der Merwe, B. Sundaralingam, and T. Hermans, "Multi-fingered grasp planning via inference in deep neural networks," *IEEE Robotics & Automation Magazine*, 2020.

[12] Q. Lu, M. V. der Merwe, and T. Hermans, "Multi-Fingered Active Grasp Learning," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2020.

[13] M. Van der Merwe, Q. Lu, B. Sundaralingam, M. Matak, and T. Hermans, "Learning Continuous 3D Reconstructions for Geometrically Aware Grasping," in *IEEE Intl. Conf. on Robotics and Automation*, 2020.

[14] M. Matak and T. Hermans, "Planning visual-tactile precision grasps via complementary use of vision and touch," *IEEE Robotics and Automation Letters*, vol. 8, no. 2, pp. 768–775, 2023.

[15] A. Mousavian, C. Eppner, and D. Fox, "6-dof graspnet: Variational grasp generation for object manipulation," in *Intl. Conf. on Computer Vision*, 2019.

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

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

[18] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "Rmp flow: A computational graph for automatic motion policy generation," in *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*. Springer, 2020, pp. 441–457.

[19] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[20] K. Van Wyk, M. Xie, A. Li, M. A. Rana, B. Babich, B. Peele, Q. Wan, I. Akinola, B. Sundaralingam, D. Fox, B. Boots, and N. D. Ratliff, "Geometric fabrics: Generalizing classical mechanics to capture the physics of behavior," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3202–3209, 2022.

[21] M. Xie, A. Handa, S. Tyree, D. Fox, H. Ravichandar, N. D. Ratliff, and K. V. Wyk, "Neural geometric fabrics: Efficiently learning high-dimensional policies from demonstration," in *6th Annual Conference on Robot Learning*, 2022. [Online]. Available: https://openreview.net/forum?id=GTyBkq36tjx

[22] Q. Chen, K. V. Wyk, Y.-W. Chao, W. Yang, A. Mousavian, A. Gupta, and D. Fox, "Learning robust real-world dexterous grasping policies via implicit shape augmentation," in *6th Annual Conference on Robot Learning*, 2022. [Online]. Available: https://openreview.net/forum?id=bUUf1CT1sNu

[23] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 627–635. [Online]. Available: https://proceedings.mlr.press/v15/ross11a.html

[24] D. Turpin, T. Zhong, S. Zhang, G. Zhu, E. Heiden, M. Macklin, S. Tsogkas, S. Dickinson, and A. Garg, "Dexgrasp-1m: Dexterous multi-finger grasp generation through differentiable simulation," in *IEEE Intl. Conf. on Robotics and Automation*, 2023.

[25] N. D. Ratliff, K. Van Wyk, M. Xie, A. Li, and M. A. Rana, "Generalized nonlinear and finsler geometry for robotics," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 10 206–10 212.

[26] B. Calli, A. Singh, A. Walsman, S. Srinivasa, P. Abbeel, and A. M. Dollar, "The ycb object and model set: Towards common benchmarks for manipulation research," in *2015 International Conference on Advanced Robotics (ICAR)*, 2015, pp. 510–517.

[27] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85.

[28] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[29] N. Gruver, M. Finzi, S. Stanton, and A. G. Wilson, "Deconstructing the inductive biases of hamiltonian neural networks," *arXiv preprint arXiv:2202.04836*, 2022.