NeRF-Aided Action Selection for Rigid Body Manipulation

Stanley Lewis Onur Bagoren

***Abstract*—Many robotic manipulation tasks assume a-priori knowledge of the relevant objects in the scene. This knowledge can take many forms, but is most commonly in some explicit form such as a CAD model or point cloud. Recent advances in implicit representations, in particular Neural Radiance Fields [1] [2] have allowed for high quality visual and geometric information retrieval with only posed images of the object - allowing for greater scalability in terms of data collection and application. In this work, we explore the usage of neural radiance field representations of several objects as trained via the Instant-NGP pipeline for action selection within a planar pushing task. This representation allows us to obtain high fidelity geometry information at arbitrary locations in space, thus allowing for action selection without and heuristics or similar object-specific approaches.**

I. Introduction

The growth of deep learning has had a crucial impact on the application of areas such as computer vision, natural language processing, and robotics. In robotics, a key area of application of learning-based methods has been the drastic improvement of control and planning, both via direct end-to-end methods using differentiable simulation, and via deeply learned representations such as signed distance fields or neural radiance fields [3] [4] [1]. Due to the highly parameterized representation and use of iterative optimization methods, deep learning approaches such as deep neural networks (DNN) frequently outperform classical methods that can leverage helpful prior information.

In this project, we aim to show initial results for the usage of a Neural Radiance Field representation on arbitrary objects for a planar pushing task. Our key objective with this project is to investigate alternative state and observation space representations for improving the performance of manipulation tasks. This allows for more scalable data collection, and for the ability to run experiments without an explicit representation of the object to be manipulated in the form of a CAD model, point cloud, or similar.

The method for implicitly learning scene geometry and radiance fields through neural methods has revolutionized computer vision and graphics. We mainly concentrate on NeRFs for this task, as it provides several benefits for modeling the scene and state of the manipulation problem we aim to solve, although it is also possible that the techniques presented here may be applicable to occupancy nets, signed distance fields, or other implicit object representations. In contrast to explicit models generated through exhaustive and time-consuming scanning, texturizing, and scaling tasks, NeRFs offer a lightweight solution to scene construction.

With these critical advantages of implicit representations in mind, in this project, we aim to use them to model the objects we will interact with in the manipulator's action space. In particular, this project consists of work on the following:

1) We train a NeRF representation of the rigid bodies to interact with,
2) we use a NeRF representation to model and parameterize the action space of the robot,
3) we train a learned residual dynamics model of a 7 DOF arm to learn dynamics model to perform a planar pushing task, and

The rest of this report is structured as follows: Section II provides technical background and notation on NeRFs, Section III covers the technical work used in the project, Section IV involves the obtained results and discussion, and Section V includes key takeaways and future work.

II. Background: Neural Radiance Fields

For an input 3D location $\mathbf{x} = (x, y, z)$ and viewing direction (θ, ϕ), Neural Radiance Fields (NeRFs) return an emitted color $\mathbf{c} = (r, g, b)$ and volume density σ. The composition of the emitted color and volume density form a continuous scene representation, which is typically performed via a Multi-Layered Perceptron (MLP) network $\mathbf{F}_\theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$. A discrete approximation of the traditional volumetric rendering method renders color from rays cast through the 3D space. For a given epipolar ray r, the numerical integration process is performed as shown in equation(1), where $\hat{C}(r)$ is the estimated pixel color, σ_i the MLP-computed color density at the i-th sample, $\mathbf{c}_i$ the MLP-computed color value at the i-th sample, and δ_i represents the distance between sample i and its adjacent sample.

$$\hat{C}(r) = \sum_{i=1}^{N} T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \mathbf{c}_i \tag{1}$$

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right) \tag{2}$$

The model parameters are then optimized by computing the mean-squared-error (MSE) loss between the rendered pixels and those in the ground truth image.

To render images, the original NeRF paper [1] proposes a frequency based encoding layer akin to the multi-headed

attention used in transformers for the samples before passing them into the MLP. Recent work to accelerate the training process of NeRFs has expanded on this technique by adding geometrical information such as voxelized latent vectors in Liu et al. [5], multi-resolution hash encodings as in Muller et al. [2], or via spherical harmonics as in Yu et al. [6].

III. Technical Approach

This section covers how we use NeRF models of the objects for a learned planar-pushing task. Additionally, the section includes details about the training of the learned dynamics model. The learned dynamics model is used in conjunction with a Model Predictive Path Integral (MPPI) controller to perform the lower level arm commands.

A. Implementation Details

1) Generating NeRFs: For the task of generating NeRF models of the objects used for learning state dynamics and improved collision detection, we train our model using a PyTorch [7] implementation of the Instant-NGP model [8]. We generate object-centric training data for the NeRF using Blender [9] to obtain high-resolution renderings of the object, along with the ground truth pose of the camera. To load the models of interest, we use 3D model files contained in the URDF of the model. We save the weights of the MLP and training logs to use in a PyBullet [10] simulation environment. In our PyBullet environment, we use the corresponding URDF models to simulate the physics of interactions. However, we use our proposed collision detection methods to inform costs associated with actions used by a MPPI trajectory optimization framework [11].

B. State and Action Space Representation

We derive the state as action paramaterizations as a function of the implicit density field of the object in question. We parameterize the state as a pose and single rotation angle on the x/y plane, denoted $\mathbf{x} \in \mathrm{SE}(2)$. For the actions $\mathbf{u}$, we parameterize it as an action applied on the surface of the object, on a point $(x, y)_{\text{act}} \in \mathbb{R}^2$ in the world frame, an angle of push $\theta_{\text{act}} \in \mathrm{SO}(2)$ and push length $l_{\text{act}} \in \mathbb{R}$. We assume that the pushing task is quasi-static, additionally containing the problem to objects that remain static when not acted upon.

To determine the object's surface, we sample a grid of x,y values at a fixed z value in the world frame, which are then transformed into the local frame of the NeRF. The NeRF is then evaluated at these point to obtain the set of (r, g, b, σ) values associated with the implicit space. On this obtained set, we apply an indicator function on the volumetric density values to obtain a binary mask of the object's surface, σ_{LS}, where only the density values greater than a threshold τ are used. For the geometries that we use in this project, we set $\tau = 1$. We then use a canny edge detection algorithm to fit a contour with the largest area on the obtained binary mask and uniform sample points along the contour. The applied action on the obtained surface of the NeRF is shown in Fig. 1.

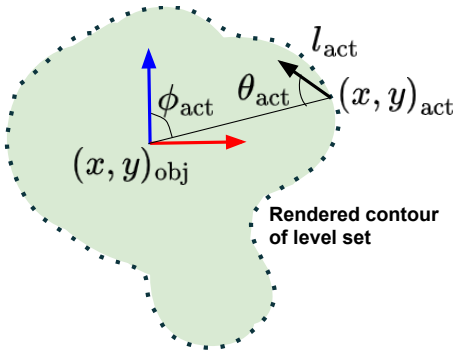

Fig. 1: A visualization of the action space, applied on an arbitrarily shaped object. We denote the rendered surface of the object as checkered lines to indicate the sampled points.

We used the obtained surface and the object pose to compute the action parameters. To determine the position to apply the force to, we sample an angle $\phi \sim U[-\pi, \pi]$ around the object's center of mass (CoM). We draw a unit vector from the CoM in the direction of ϕ. For each point sampled on the surface of the contour, we also draw vectors to the CoM and find the vector that maximizes the dot product between the two vectors. The corresponding point is $(x, y)_{\text{act}}$. The push angle $\theta_{\text{act}} \sim U[-\pi/2, \pi/2]$ and push length $l_{\text{act}} \sim U[0, 1]$.

C. Learning Planar Pushing from State Representations

We use a single-step absolute dynamics model to learn the planar pushing task. Our model is trained on data from the robot arm pushing objects in random trajectories. We note that to generate the random trajectories, we apply the same strategy of selecting applied actions described in Section III-B. To transform points along the trajectory and be able to sample actions over an arbitrarily large trajectory size, we apply a transformation to each sampled point on the contour consistent with the resulting displacement of the robot.

The objective of the absolute dynamics model $f_\psi(\mathbf{x}_t, \mathbf{u}_t)$ is to directly predict the next states, given the current state $\mathbf{x}_t$, and action $\mathbf{u}_t$, as shown in Eqn. (3). This is a contrast to the residual dynamics model, which aims to predict the displacement of the state given the current state and action. From empirical results, we find that the absolute dynamics model performs better than the residual dynamics model; hence we use the absolute dynamics model.

$$\hat{\mathbf{x}}_{\mathbf{t+1}} = f_\psi(\mathbf{x}_t, \mathbf{u}_t) \tag{3}$$

The training of this model requires a state-action-next state tuple $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$, and a single-step loss that uses an SE(2) pose loss over the predicted state and the ground truth state. The single-step loss is used for computing the loss between two rigid bodies in a 2D space, which is then used to optimize over the parameters ψ of the absolute state dynamics model. The computation of the SE(2) pose loss is shown in Eqns. (5)-(7), where r_g is the radius of gyration, which we approximate across all of our tests to be equivalent to that of a rectangle.

We make this assumption, as the geometries we are working with can be approximated as a rectangle. We initially aimed to use the inertial properties of the objects from their URDF files but found that despite different geometries and mass properties, the inertial properties for a lot of the data were kept the same, and we found that this approximation was closer than using the values from the URDF.

$$\mathbf{q}_1 = \begin{bmatrix} x_1 & y_1 & \theta_1 \end{bmatrix}^\top \in \text{SE(2)} \tag{4}$$

$$\mathbf{q}_2 = \begin{bmatrix} x_2 & y_2 & \theta_2 \end{bmatrix}^\top \in \text{SE(2)} \tag{5}$$

$$r_g = \sqrt{\frac{w^2 + l^2}{12}} \tag{6}$$

$$\mathcal{L}_{\text{SE(2)}}(\mathbf{q}_1, \mathbf{q}_2) = \|x_1 - x_2\|^2 + \|y_1 - y_2\|^2 + r_g\|\theta_1 - \theta_2\|^2 \tag{7}$$

We use an Adam optimizer to optimize the model parameters ψ. We use 1000 epochs to train the absolute dynamics model and use a learning rate of $1e-3$.

IV. Results and Discussion

A. Manipulation of Arbitrary Objects Using NeRF Representations

In this section, we provide qualitative results indicating the success of using NeRFs for extracting an easy-to-sample representation of the object geometry. We motivated our reason for using NeRFs for their compact representation that inherently comes from the implicitly learned volumetric function. An additional advantage posed by this implicit representation is the ability to extract an explicit representation of the geometry. This makes it easy to use it for things such as sampling points while also keeping a compact and amenable representation. We provide qualitative results in Fig. 2.

We point out the variety of the geometries with which we train the NeRF and successfully obtain the contours of. In order to leverage unique, non-convex geometries, we rendered parts of the robotic arm, the Panda Franka.

A key point to mention is the necessity of a converged NeRF model to be used. We observed that if we use a NeRF model that is not trained for a sufficient number of iterations, there were issues in the construction of the contours. An example of this, which does not affect the pushing model to a great extent, can be seen in Fig. 2b, where we extract the contour of a clamp at it's median z point. The top end of the clamp, despite not being significant, is deformed compared to it's rendered output in Fig. 2f. The reason for this is the continuous volumetric density function that the NeRF implicitly learns during training time. If the continuous function is not optimized for with further training, minute details of the scene still remain to be accounted for. As we extract the points at a level set fixed at a certain z and threshold at a certain τ value, if the function is not trained to convergence, minute deformations due to an ill-fit function is to be expected.

In order to mitigate more major levels of failure, we train the NeRF model for 60k iterations, which takes around 15 minutes to train.

B. Planar Pushing Task

Our key contribution to the planar pushing task is from the method in which we parameterize our action space and use a NeRF model to aid the sampling of points to apply action to arbitrarily shaped geometries. We additionally present a method for computing the point on which to apply the action to the object, similarly utilizing the NeRF model for it.

Our proposed approach utilizes a purely sampling-based approach to determine the points, angles, and lengths to which we apply the action. In order to motivate the sampling-based approach, we bring to attention previous methods that we developed to determine the point to apply actions.

Our initial approach was to design an alternative method to pick the position on the contour to which we apply the action to. This method involved setting the action space to a reasonably large area, specifically an area within the configuration space of the 7 DOF robot, which is a 4-meter square centered around the base of the robot. During the training data generation task, the strategy was to sample data in this large area and set that position as $(x, y)_{\text{act}}$, and then uniformly sample θ_{act} and l_{act}. This brought up problems for training our model. As the positions that we would sample largely be in the free space, this would often result in the robot applying an action to free space, collecting data that did not help the learning process of the absolute dynamics model. Despite training data that was not representative of the target-pushing task, the training and validation losses during the training process decreased and converged to a steady value. We suspect that the underlying reason for this was that the model correctly learned that when pushing actions were applied to free space, the object did not move. This took away the ability of the model to learn the underlying dynamics model of pushing; hence did not perform well in our test cases.

The advantage of our proposed method is that the sampling-based method guarantees that, under the condition that the NeRF-based contour is representative of the geometry, an interaction point on the object will be sampled.

In order to validate that the proposed method was able to perform the planar pushing task, we tested it by training the absolute dynamics for 1000 epochs and running the pushing task to a target pose 10 times. We additionally tracked the loss at the end of the training model. We use these quantities to both validate our method and justify our selection of the absolute dynamics model over the residual dynamics model. We compare the loss at the end of the training of the model in Table I. The results specifically point to the improvement of using the absolute dynamics model over the residual dynamics model. The residual model tends to have a lower train, but a higher validation loss. The ability of the absolute dynamics model to perform better on unseen data to predict the next state based on the applied action and current state is the primary reason for selecting the absolute dynamics model.

We additionally ran tests where we ran the absolute dynamics model 10 times with Link 6, shown in Fig. 2d. The model succeeded in pushing the object to the goal pose in 50% of

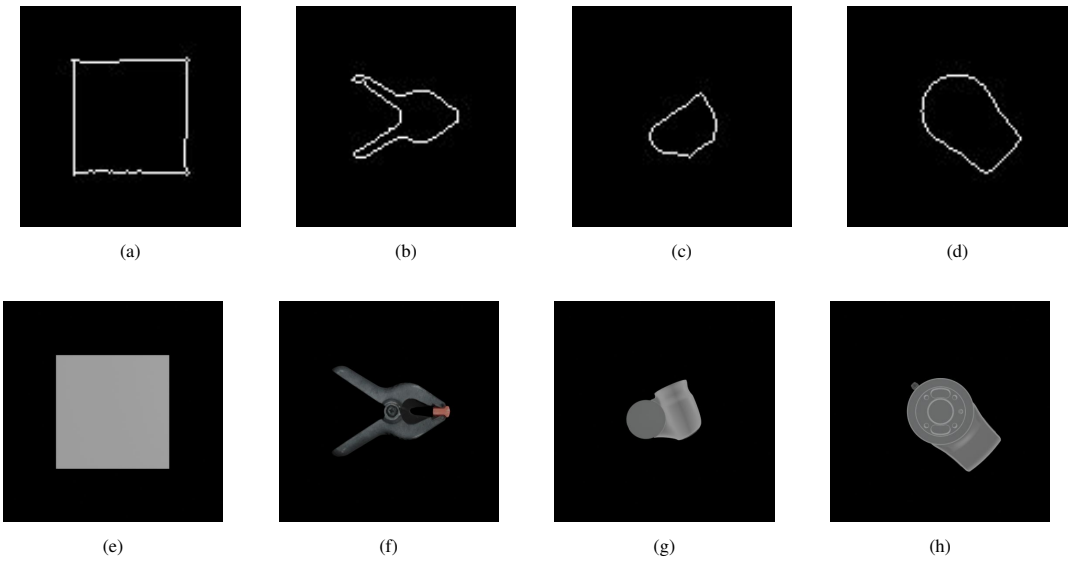

Fig. 2: Qualitative results on the extracted contours of arbitrary geometries. We use these contours to sample locations to act on, both for collecting training data and for the planar pushing task. The top row represents contours extracted from various types of geometries: 2a a cube, 2b a clamp, 2c link 6 of the Frank Panda arm, 2d link 7 of the Franka Panda arm. The bottom row represents the renderings of each object from the NeRF, to display the consistency of the geometries that our contours obtain.

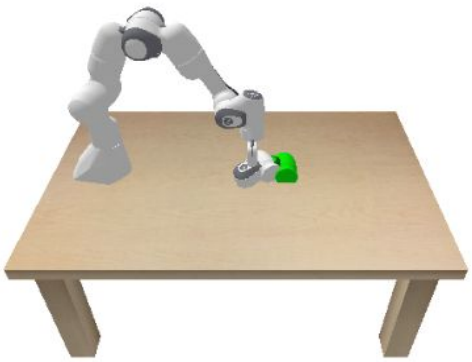

Fig. 3: An example visualization of a PyBullet environment where the robot is interacting with a nonconvex object, pushing it toward a goal pose shown in bright green.

TABLE I: The loss reported at the end of the training after 1000 epochs. We compare the losses across each model corresponding to the same object. We use a diverse set of objects to study the generalization of the models while also using them as benchmarks for how we select our model.

Losses Reported at End of Training

Object	Model	Train Loss	Validation Loss
Link 7	Absolute Dynamics Model	0.000457	0.000804
	Residual Dynamic Model	0.000478	0.000838
Link 6	Absolute Dynamics Model	0.001572	0.002055
	Residual Dynamic Model	0.001309	0.002684
Cube	Absolute Dynamics Model	0.000834	0.001564
	Residual Dynamic Model	0.000741	0.001718

the attempts.

An observation we made during the testing time was regarding the points to which the actions were applied. Despite our method of sampling actions guaranteeing contact for arbitrary shapes with nonconvex and complicated geometries, the planar pushing task that we designed is set up for applying the action at a fixed z value, which is separate from the z value that we set for extracting the level set of the NeRF. In addition, in between actions that are applied, the robot lifts the end effector to avoid collisions when moving to the next position to apply the action. An issue that arises with this is that for highly unstructured objects, the fixed point at which the contact is made may be under another part of the object. The unintended effect of this is that the robot presses down on the object, "squeezing" it out and registering that as part of the planar pushing task. This has obvious implications of hindering the learning objective, as the data collected, albeit better than our previous method, does not generate data that is representative of the learning objective. We supplement our submission of .git files showing both the robot succeeding in pushing the object to the goal pose, and this mentioned struggle.

V. Conclusion and Future Work

In this project, we propose a novel method for training a planar-pushing model through the use of NeRFs. Our motivation for this task is that for objects with arbitrary geometries, parametrizing the action space can be done so that contact points are guaranteed for learning dynamics from state representations. We formulate the problem so that the object that is being pushed around is a NeRF representation of the geometry. We take a level set at a fixed z value on the world frame and transform it into the NeRF frame to obtain a set of colors (r, g, b) and volumetric density σ. A threshold of $\tau = 1$ is used on the volumetric density σ to mask out

samples that are considered as not the boundary. We then use a contour detection method to fit a contour with a maximum area to construct the geometry of the surface at the level set.

Our representation of the surface is then used for sampling points to which the pushing task applies a pushing action. We sample points in a stochastic manner in order to generate training data. We particularly generate training data by running a trajectory of pushing actions and storing them in a state-action-next state tuple. We use the generated training data to train an absolute dynamics model with a single-step SE(2) pose loss. For the SE(2) pose loss, we approximate the geometry as a rectangle with the height and width of the extent of the contours. We use the learned dynamics model and use MPPI to perform a pushing task to move the object to a goal pose.

Quantitative and qualitative evaluations of our methods are done to verify the ability of the learned dynamic model to push objects with arbitrary geometries in an obstacle-free environment, along with verifying the correct construction of the surface geometry of objects. We note that the key advantage of our proposed method includes not only the fact that we can generalize to arbitrary geometries but that the obtaining of arbitrary geometries is done in a quick manner as well, using an implicitly learned volumetric function from camera data, rather than generating dense explicit models.

We show that the parametrization of the action space and the sampling method for guaranteeing that actions are applied on the surface of the object help us train models for effectively learning pushing dynamics.

For future work, we aim to work towards generalizing the pushing task to operate in 3D spaces rather than on a plane. A key limitation of our method is that for arbitrarily shaped, complex objects, the act of "squeezing" the object can occur when the robot end effector aims to move to a specified position in the 3D space. This inherently occurs because the robot's end effector moves directly down for our pushing task. Despite the dynamics model learning this motion as part of the pushing dynamics, it deviates from the original learning objective, producing suboptimal results.

We propose that a more complex parameterization of the action space will be able to handle robot motion in 3D spaces. A key thing to consider is to enforce a uniform, and consistent forces applied to the object. As we assume that the pushing task is quasi-static, the only forces that result in the movement are from the pushing forces, besides the friction that makes the object static when not acted upon. Hence, an action sampling method that guarantees not only contact with the surface of the object but also consistently uniform forces, resulting in an equal magnitude of speed on each interaction, can be a promising direction to enable more complicated interactions.

REFERENCES

[1] 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," Aug. 2020, arXiv:2003.08934 [cs]. [Online]. Available: http://arxiv.org/abs/2003.08934

[2] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 1–15, Jul. 2022, arXiv:2201.05989 [cs]. [Online]. Available: http://arxiv.org/abs/2201.05989

[3] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," *Advances in neural information processing systems*, vol. 31, 2018.

[4] D. Driess, J.-S. Ha, M. Toussaint, and R. Tedrake, "Learning models as functionals of signed-distance fields for manipulation planning," in *Conference on Robot Learning*. PMLR, 2022, pp. 245–255.

[5] L. Liu, J. Gu, K. Zaw Lin, T.-S. Chua, and C. Theobalt, "Neural sparse voxel fields," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 651–15 663, 2020.

[6] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," *arXiv preprint arXiv:2112.05131*, 2021.

[7] 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 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[8] J. Tang, "Torch-ngp: a pytorch implementation of instant-ngp," 2022, https://github.com/ashawkey/torch-ngp.

[9] B. O. Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: http://www.blender.org

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

[11] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore: IEEE, May 2017, pp. 1714–1721. [Online]. Available: https://ieeexplore.ieee.org/document/7989202/

[12] J. Ye, Y. Chen, N. Wang, and X. Wang, "GIFS: Neural Implicit Function for General Shape Representation," Jul. 2022, arXiv:2204.07126 [cs]. [Online]. Available: http://arxiv.org/abs/2204.07126