Abstract—We present a differentiable pipeline for simulating the motion of objects that represent their geometry as a continuous density field parameterized as a deep network. This includes Neural Radiance Fields (NeRFs), and other related models. From the density field, we estimate the dynamical properties of the object, including its mass, center of mass, and inertia matrix. We then introduce a differentiable contact model based on the density field for computing normal and friction forces resulting from collisions. This allows a robot to autonomously build object models that are visually and dynamically accurate from still images and videos of objects in motion. The resulting Dynamics-Augmented Neural Objects (DANOs) are simulated with an existing differentiable simulation engine, Dojo, interacting with other standard simulation objects, such as spheres, planes, and robots specified as URDFs. A robot can use this simulation to optimize grasps and manipulation trajectories of neural objects, or to improve the neural object models through gradient-based real-to-simulation transfer. We demonstrate the pipeline to learn the coefficient of friction of a bar of soap from a real video of the soap sliding on a table. We also learn the coefficient of friction and mass of a Stanford bunny through interactions with a Panda robot arm from synthetic data, and we optimize trajectories in simulation for the Panda arm to push the bunny to a goal location.

Index Terms—Simulation and animation, contact modeling, neural object representations, differentiable contact simulation, real-to-sim transfer.

I. INTRODUCTION

We present the Dynamics-Augmented Neural Object (DANO), a novel object representation that augments a neural object with dynamical properties, so that its motion under applied forces and torques can be simulated with a differentiable physics engine. We also propose a method for computing contact forces due to collisions between the DANO and other objects in the simulation. We specifically focus on neural objects that are trained from RGB images only, and encode their geometry through a neural density field—a continuous density field represented as a deep neural network, such as the Object-centric Neural Scattering Function (OSF [1]), which is an object-centric NeRF [2], [3], or related models [4], [5]. DANOs capture both geometric and dynamical properties of rigid objects from the underlying neural density field. This allows a robot to build a dynamical simulation of an object from observed RGB images, and fine-tune that model through real-to-sim transfer using videos of the object in motion.

The main challenge with neural density fields is that they do not give a distinct object surface, and are better interpreted as a differential probability of occupancy. Since contact forces (friction and normal forces) arise from interactions between surfaces, how does one simulate contact for objects with neural density fields, which have no distinct surface? One naive approach is to choose a single level set of the neural density to stand in as the object surface, then compute a traditional mesh model from this level set using, e.g., marching cubes [6] and simulate motion with the resulting mesh. We show that this leads to poor-quality meshes with significant spurious artifacts that prevent accurate simulation. Instead, we propose an inherently probabilistic differentiable model of contact for objects with neural density fields, and derive Monte Carlo techniques for computing contact forces under this model. We show that this model can give high-fidelity simulation trajectories nearly indistinguishable from trajectories simulated with ground-truth knowledge of the 3D object geometry.

Our method is illustrated in Fig. 1. We first train a neural object model to represent the object geometry implicitly as a neural density field. We then compute a Monte Carlo estimate of the mass, center of mass, and inertia matrix from the neural density field, up to an unknown mass scale factor.1 We propose a Monte Carlo approach to simulate the contact forces (normals and friction forces) based on sampled candidate surface points

1The mass and inertia matrix can only be found up to a scale factor from images and video alone. The mass scale factor can be resolved by observing object trajectories with known forces, or contact interactions with other objects of known mass, such as a known robot arm.

Authorized licensed use limited to: Stanford University. Downloaded on August 25,2023 at 20:10:04 UTC from IEEE Xplore. Restrictions apply.
Fig. 1. Our pipeline for simulating the motion of neural objects. A neural object model such as a NeRF is trained from still images (top left), and object trajectories are extracted from videos of the object in motion (bottom left). Object mass and friction properties are computed and candidate surface points and normals are sampled to produce a Dynamics-Augmented Neural Object (DANO) (middle). The DANO is simulated in a differentiable physics simulator interacting with planes, robots, and other rigid objects (right). The resulting simulation can be used for real-to-simulation transfer, or to synthesize robot behaviors, e.g., for manipulation.

Fig. 2. DANO obtained from a neural object model of the Stanford bunny. We sample points (white dots) from the neural density field and compute the field gradients (blue arrows) to obtain an approximate outward normal, which is used to compute contact forces in our simulator. We estimate the mass, center of mass (black dot), and inertia matrix by integrating over the neural density field. A contact force (yellow arrow) due to the collision between the DANO and the plane is applied at the centroid of the overlap volume (red dot), which is exaggerated for visualization.

and normals, as shown in Fig. 2, and integrate this contact model within an existing differentiable physics simulator, Dojo [7]. Using DANO models in Dojo, we demonstrate real-to-sim transfer by obtaining, from real video sequences of a sliding bar of soap, an estimate of the coefficient of friction of the soap. We also estimate, from synthetic trajectory data, the coefficient of friction and the missing mass scale factor for a neural object model of a Stanford bunny. Finally, we leverage the differentiability of the simulation pipeline to optimize a trajectory in simulation for a robot arm to push the Stanford bunny to a goal location. Our key contributions are:

- Estimating inertial and friction properties and sampling candidate surface points and normals from a neural density field to form a Dynamics-Augmented Neural Object (DANO).
- A differentiable formulation of rigid-body contact forces for DANOs.
- Integration of DANO models with the Dojo differentiable physics engine.
- Demonstration of the differentiable pipeline to learn physical properties from real videos and synthetic data, and to optimize a pushing trajectory for a robot arm.

II. RELATED WORK

Extracting accurate object models from real perception data to allow a robot to plan and interact with those objects is a grand challenge in robotics with substantial existing work from multiple related fields.

Dynamic NeRF. Recent years have witnessed an explosion in neural scene representations [8], [9], [10]. In particular, Neural Radiance Fields (NeRF) [2] have shown impressive novel view synthesis results from real-world images without any 3D input, indicating the promise to reconstruct both appearance and geometry from only RGB images. To allow modeling dynamics in addition to appearance, there have been various extensions to NeRF for deformable objects [11], [12], [13], [14], [15] and general motions [16], [17], [18], [19], [20]. However, these methods only fit given dynamic events from videos without generalization ability to new scenes or novel motions. The closest to our work are a few recent works that aim to learn dynamic NeRFs for planning and control [21], [22], [23]. Nevertheless, all of these focus on learning to approximate dynamics by black-box latent representations without accounting for the underlying physics, and thus they cannot generalize to unseen motions or scenarios. In contrast, our approach explicitly infers dynamic physical parameters, and simulates motion in a dynamics engine that respects the known laws of Newtonian mechanics. Because we rely on a physics engine instead of a learned dynamics model, we can predict motions in scenarios that are arbitrarily different from the training data, including previously unseen collisions between multiple neural objects, or between a neural object and URDF robot models, or arbitrary arrangements of shape primitives such as half-spaces, spheres, and capsules.

Real-to-Simulation. Some previous works have relied on videos of the object in motion to identify parameters of a dynamical model [23], [24], [25], [26], [27]. ACID [23] and DiffCliff [25] focus on identifying models for deformable objects, while we focus on rigid objects in this letter. ContactNets [26] regresses a rigid contact model from observed 3D object trajectories obtained from videos of objects with Apriltags [28]. ContactNets starts with a geometrical object model, which can be fine-tuned through this trajectory data. In contrast, we obtain our DANO model from untagged RGB images without prior geometrical information, and fine-tune mass and friction coefficients through untagged videos. Finally, GradSim [27] combines a differentiable rendering tool and a differentiable physics simulator. Our work differs from GradSim in two respects: GradSim represents objects as meshes, while we leverage the neural density field learned directly from images; GradSim demonstrates impressive system identification capabilities in simulated environments, while we further perform experiments with real-world data.

Differentiable Simulation. Differentiable simulators promise the ability to back-propagate gradients through a simulation roll-out, to allow for optimization of simulation parameters to fit observed data (System Identification), or to optimize robot motions or robot policies. Several differentiable rigid-body simulators have been recently proposed [7], [29], [30], [31], [32]. They handle contact for simple shape primitives (spheres, capsules, etc.) and more complex models decomposed into a union of convex shapes. However, none of them support the simulation of neural objects represented by density fields. In this work, we propose a novel differentiable contact model for rigid bodies represented by a neural density field. We seamlessly embed this
III. OBJECT-CENTRIC NEURAL SCATTERING FUNCTION (OSF) MODEL

We use a specific object-centric model\(^2\) called the Object-Centric Neural Scattering Function (OSF) [1]. Unlike NeRFs, which assume a static scene with fixed illumination, OSFs are relatable and compositional. Thus, OSFs allow representing dynamic scenes where objects can move and change appearances (e.g., an object’s shadow changes with its pose), while NeRFs do not support this purpose. OSFs are also relatable, meaning they can render from a variety of different lighting conditions.

More specifically, an OSF models a volumetric function \((x, \omega_{\text{light}}, \omega_{\text{out}}) \rightarrow (\phi, \rho)\), where \(\rho = (\rho_x, \rho_y, \rho_z) \in \mathbb{R}^3\) is the cumulative radiance transfer to model the object appearance and \(\phi(x)\) is the volumetric density that models the object geometry. The \(\rho\) function takes as input a 3D point \(x \in \mathbb{R}^3\) in the object coordinate frame, an incoming distant light direction \(\omega_{\text{light}}\) and an outgoing radiance direction \(\omega_{\text{out}}\) at that location, while the density \(\phi(x)\) only requires the 3D point. The OSF can be reposed in a background scene, or with respect to other neural objects through a pose transform applied to \(x, \phi(Rx + \tau)\), where \((R, \tau)\) are a rotation matrix and translation vector, respectively, defining the pose of the object. In this work, we use the learned density field \(\phi(x)\) of the object as a proxy for modeling its geometry. We do not use the appearance information \(\rho\), though inferring dynamical properties such as mass density and friction coefficient from appearance is a promising direction for future work.

IV. DYNAMICS AUGMENTED NEURAL OBJECTS (DANOS)

In this section, we describe how to obtain a DANO from a neural density field by estimating the mass, center of mass, and inertia matrix directly from the field. We formulate a probabilistic contact model, and detail computations for contact forces between the DANO and a shape primitive (e.g., a half-space, sphere, or mesh), and between two DANOS. Ultimately, we integrate this contact model into the Dojo simulator [7], which uses the implicit function theorem to differentiate through the integrator.\(^3\)

Estimating Mass, Center of Mass, and Inertia Matrix. In an offline phase, we estimate the object’s mass, inertia matrix, and center of mass by treating the neural density field \(\phi(x)\) as a scaled mass density field. We compute Monte Carlo estimates of the integrals that define the mass, inertia matrix, and center of mass in terms of the mass density. Specifically, we uniformly sample a set of points \(X\) from the density field \(\phi(x)\), and compute the mass, inertia matrix, and center of mass as:

\[
\tilde{\phi}(x) = \alpha \phi(x),
\]

where we denote \(\tilde{\phi}\) the volumetric mass density field, \(\alpha\) the scaling factor, \(m\) the mass, \(J\) the moment of inertia, and \(\mu\) the center of mass. The parameter \(\alpha\) is the unknown mass scale factor that can only be estimated from observed interactions between the object and known forces or other objects of known mass. We discuss the identification of this parameter, along with friction parameters, in Section V.

We note that the scaled density field \(\tilde{\phi}\) does not align well with the full mass distribution of the real object on a point-by-point basis. However, as illustrated in the experiments in Section V, we find empirically that integrating over the density field does give close enough approximations to the mass, center of mass, and moment of inertia matrix to provide dynamically plausible simulations. These estimates can serve as initial guesses for a system identification method to further refine them to match the real mass, center of mass, and inertia matrix of the object.

The scaled density field \(\phi\) is the best guess we can make from still images. However, incorporating information from a video of the object in motion would allow for refinement of these initial estimates and leads to more accurate estimation of these quantities.

Probabilistic Contact Model. Given the mass, centroid, and inertia matrix of the neural object computed above, we can simulate its behavior in free space. However, contact interactions with the environment or with a robot require us to compute friction and contact forces. In physics simulation, the contact force direction and magnitude are computed to limit interpenetration between the object and its environment [33], [34]. This requires determining the location and the amount of interpenetration between an object and its environment. Classical formulations of contact interaction rely on signed distance functions (SDFs) [32] to measure this quantity. With neural objects, we only have access to the volumetric density field \(\phi : \mathbb{R}^3 \rightarrow \mathbb{R}_+\), which has no distinct notion of a surface.

The core idea behind our contact model is to measure the amount of interpenetration between two objects. Given two objects \(A\) and \(B\), we measure the overlap as the integral,

\[
\psi = \int_{x \in \mathbb{R}^3} \phi^A(x) \phi^B(x) dx,
\]

where \(\phi^A\) and \(\phi^B\) are the product of their density fields \(\phi^A\) and \(\phi^B\) over the whole workspace \(\mathbb{R}^3\). A probabilistic interpretation of this expression gives an intuitive justification for this choice. Let us assume that \(\phi^A(x) = P(H^A_x = 1)\) is the probability of hitting object \(A\) when sampling a point \(x\) in \(\mathbb{R}^3\). Here, \(H^A_x\) is a random variable taking value 1 to indicate collision and 0 otherwise. Then assuming \(H^A_x\) and \(H^B_x\) are independent random variables (i.e., the shape
of one object is independent of the shape of the other, \( \phi^A(x) \cdot \phi^B(x) = P(H^A_x = 1 \cap H^B_x = 1) \) is the probability of hitting both objects \( A \) and \( B \) when sampling point \( x \). Finally, \( \psi \) (5) represents the expected interpenetration volume between objects \( A \) and \( B \). We let the amount of repulsive force applied between objects \( A \) and \( B \) be proportional to \( \psi \).

Additionally, we compute the expected centroid of the interpenetration volume. This is the geometric center of the interpenetration volume and the point at which we apply the repulsive force contacts,

\[
\chi = \frac{1}{\psi} \int_{x \in \mathbb{R}^3} x \phi^A(x) \phi^B(x) \, dx, \quad \text{if } \psi \neq 0. \tag{6}
\]

When \( \psi = 0 \), there is no interpenetration, and the computation of \( \chi \) is unnecessary.

**Neural Object Sampling Procedure.** Exact computation of the integrals in (5), (6) is intractable. We approximate them using a Monte-Carlo sampling scheme. As with the mass, center of mass, and inertia matrix computations, we sample a set of points \( X \) uniformly from the neural density field \( \phi(x) \) workspace. For contact interactions, we keep the points between a minimal and maximal density value and discard the rest, since low-density points tend to be outside the object, and high-density points tend to be farther in the interior of the object. This biases the sampling of points towards the boundary of the object. This way, we densely cover the boundary of the object while limiting the number of sample points for computational efficiency. In all the experiments presented in this letter, we use 5000 sample points. We denote \( X^A \) the set of \( N^A \) points sampled from object \( A \).

**Contact Between Neural Object & Shape Primitive.** To illustrate our contact model, we choose a simple scenario (Fig. 3, left) where a neural object \( A \) collides with the ground represented by a half-space \( B \). The shape primitive can be expressed as \( B = \{ x \in \mathbb{R}^3 \mid f(x) \leq 0 \} \) where, for example, \( f(x) = ax^T x + b \) for a half-space, or \( f(x) = ||x - c|| - r \) for a sphere with center \( c \) and radius \( r \). We represent the density function for the shape primitive as \( \phi^B(x) = 1_{x \in B} \), where 1 denotes the indicator function. Finally, we approximate the interpenetration volume and its centroid (5), (6) as sums over sampled points:

\[
\psi \approx \frac{1}{N} \sum_{x \in X^A} \phi^A(x) \phi^B(x) = \frac{1}{N} \sum_{x \in X^A \cap B} \phi^A(x), \tag{7}
\]

\[
\chi \approx \frac{1}{\psi} \sum_{x \in X^A \cap \mathbb{R}^3} x \phi^A(x). \tag{8}
\]

Each sampling point \( x \in X^A \) is expressed in a frame attached to object \( A \). Thus the value of \( \phi^A(x) \) is independent of the position and orientation of objects \( A \) and can be computed offline. Translations and rotations of object \( A \) influence the values of \( \psi \) and \( \chi \), modifying the number of sampling points belonging to \( B \). For example, if object \( A \) is positioned halfway through the ground, \( \psi \) will be large, whereas if object \( A \) is above the ground, \( \psi \) should be close to zero. Since \( \phi^A(x) \) is pre-computed, we can simulate contact without resampling the density field online. The contact normal \( n \) is the outward facing normal to the shape primitive computed at the point \( x \), e.g., \( n = (x - c)/||x - c||_2 \) for a sphere centered in \( c \). We implement this to simulate contact with half-spaces and spheres, but this approach generalizes to a variety of shape primitives, including capsules, boxes, and compositions of shape primitives.

**Contact Between Two Neural Objects.** We follow the same approach to model contact between two neural objects \( A \) and \( B \) (Fig. 3 right). We process object \( A \)’s density field to extract a set of \( N^A \) sampled points \( X^A \) expressed in a frame attached to object \( A \). We do the same for object \( B \). Then our Monte-Carlo sampling scheme gives

\[
\psi \approx \frac{1}{N^A + N^B} \sum_{x \in X^A \cap X^B} \phi^A(x) \phi^B(x), \tag{9}
\]

\[
\chi \approx \frac{1}{\psi(N^A + N^B)} \sum_{x \in X^A \cap X^B} x \phi^A(x) \phi^B(x). \tag{10}
\]

We precompute \( \phi^A(x^A) \) for all \( x^A \in X^A \) and \( \phi^B(x^B) \) for all \( x^B \in X^B \). However, the value of \( \phi^A(x^B) \) for \( x^B \) in \( X^B \) varies with the relative configuration of objects \( A \) and \( B \). Thus, we compute these quantities online each time the relative position of the two objects changes. Identifying the normal to the contact requires both offline and online computing. Offline, we sample outward facing normals aligned with the neural density field gradient \( n(x) = -\nabla_x \phi(x)/||\nabla_x \phi(x)||_2 \) using a finite-difference scheme (blue arrows Fig. 2). Online, we compute the normal \( n \) to the contact as a weighted average of the offline sampled contact normals,

\[
\bar{n} \approx \sum_{x \in X^A} n^A(x) \phi^A(x) \phi^B(x) - \sum_{x \in X^B} n^B(x) \phi^A(x) \phi^B(x)
\]

where \( \bar{n} = \bar{n} / ||\bar{n}||_2 \). The minus sign accommodates for the opposite direction of \( n^A(x) \) and \( n^B(x) \) as illustrated in Fig. 3.

**Computing Contact Forces.** We compute contact forces and torques proportional to the interpenetration volume \( \psi \), applied at the centroid of the interpenetration volume \( \chi \), and applied in the direction normal to the contact surface, as shown in Fig. 4. We define the linear velocity \( v \in \mathbb{R}^3 \) of point \( \chi \) attached to frame \( A \) with respect to frame \( B \), and the angular velocity \( \omega \in \mathbb{R}^3 \) of frame \( A \) with respect to frame \( B \). We apply a force normal to the contact to oppose interpenetration,

\[
F_n = -\psi (I_{spring} n + I_{damper} v_n), \tag{11}
\]

where \( n \) is the unit vector normal to the contact, and \( v_n \) is the component of the velocity \( v \) normal to the contact.

\( I_{spring} \) and \( I_{damper} \) are parameters modeling the stiffness of the contact. We apply a force tangential to the contact to oppose

---

Fig. 3. Left: contact between a DANO (A) and the ground (half-space B). Each point sampled on the density field below ground level contributes to the repulsive force on the DANO. Points with a larger density value generate more force. The repulsive forces are applied at point \( \chi \), which represents the geometric center of the interpenetration volume between object \( A \) and \( B \). The direction of the repulsive force is the outward normal of the half-space. Right: contact between two DANOS. Points, where both objects have large density values, generate large repulsive forces. Each repulsive force is directed along its local density field gradient and contributes to the overall contact normal proportionally to its magnitude.
Contact modeling between a DANO $A$ and a primitive shape $B$ (half-space). All forces and torques are applied at the centroid of the interpenetration volume $\chi$. The normal force $F_n$, opposing contact interpenetration is applied along the contact normal $n$. A tangential force $F_t$ modeling sliding friction is applied. This force opposes the velocity of the point $\chi$ in the plane tangential to the contact. Similarly, rolling friction is applied via a torque $\tau_n$ in the tangential plane. Finally, $\tau_n$ generates torsional friction.

\[ F_t = -||F_n|| \left( \frac{\omega_n}{||\omega_n||} + S_{\text{drag}} \omega_n \right), \quad (12) \]

where $\omega_n$ is the normal component of $\omega$, $T_{\text{friction}}$ and $T_{\text{drag}}$ are parameters modeling dry and viscous frictions. We apply a torque normal to the contact to encode torsional friction,

\[ \tau_n = -||F_n|| \left( \frac{T_{\text{friction}}}{||\omega_n||} + T_{\text{drag}} \omega_n \right), \quad (13) \]

where $\omega_n$ is the normal component of $\omega$, $T_{\text{friction}}$ and $T_{\text{drag}}$ are parameters modeling dry and viscous torsional frictions. Finally, we apply rolling friction that opposes the relative rotation of object $A$ with respect to object $B$ when they are in contact,

\[ \tau_n = -||F_n|| \left( R_{\text{friction}} \frac{\omega_n}{||\omega_n||} + R_{\text{drag}} \omega_n \right), \quad (14) \]

where $\omega_n$ is the tangential component of $\omega$, $T_{\text{friction}}$ and $T_{\text{drag}}$ are parameters modeling dry and viscous rotational frictions.

### Table I

<table>
<thead>
<tr>
<th>Parameter</th>
<th>min</th>
<th>max</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>impact spring</td>
<td>$10^4$</td>
<td>$10^6$</td>
<td>stiffer impact</td>
</tr>
<tr>
<td>impact damper</td>
<td>$10^4$</td>
<td>$10^6$</td>
<td>more stable simulation</td>
</tr>
<tr>
<td>sliding friction</td>
<td>0</td>
<td>1</td>
<td>less sliding</td>
</tr>
<tr>
<td>sliding drag</td>
<td>0</td>
<td>0.1</td>
<td>more stable simulation</td>
</tr>
<tr>
<td>rolling friction</td>
<td>0</td>
<td>0.1</td>
<td>less rolling</td>
</tr>
<tr>
<td>rolling drag</td>
<td>0</td>
<td>0.1</td>
<td>more stable simulation</td>
</tr>
<tr>
<td>torsional friction</td>
<td>0</td>
<td>0.1</td>
<td>less spinning</td>
</tr>
<tr>
<td>torsional drag</td>
<td>0</td>
<td>0.1</td>
<td>more stable simulation</td>
</tr>
</tbody>
</table>

For each parameter, we provide a nominal range of values that produce realistic physical behavior. Additionally, we describe how changing this value affects the simulation.

In this section, we use a DANO model of a bar of soap acquired from real images of the soap. We find the coefficient of friction of the soap from a real video of the soap sliding on a table. We then use a DANO model of the Stanford bunny (acquired from synthetic images of the bunny) to find the friction coefficient and mass scaling coefficient of the bunny (in simulation). Finally, we optimize a robot trajectory in simulation for pushing the bunny to a goal location using the differentiability of the simulation pipeline.

#### V. Applications

In this section, we use a DANO model of a bar of soap acquired from real images of the soap. We find the coefficient of friction of the soap from a real video of the soap sliding on a table. We then use a DANO model of the Stanford bunny (acquired from synthetic images of the bunny) to find the friction coefficient and mass scaling coefficient of the bunny (in simulation). Finally, we optimize a robot trajectory in simulation for pushing the bunny to a goal location using the differentiability of the simulation pipeline.

\[ F = F_n + F_t, \quad (15) \]

\[ \tau = \tau_n + \tau_t + \mu \chi \times F. \quad (16) \]

We use a first-order variational integrator identical to Dojo’s [7]. In addition, forces related to neural object contacts are computed explicitly at the current configuration and integrated over one time step. To illustrate the DANO and contact force model described above in a complex simulation scenario, we show a simulation run with contact interactions between two Stanford bunny DANOs, a sphere, and a half-space (ground) in Fig. 5. Please refer to the supplemental video https://youtu.be/Md0PM-wv_Xg for animations of multiple different simulation scenarios. An implementation in the Julia language of the method and applications is publicly available https://github.com/dojo-sim/Dojo.jl/tree/DANO.

#### V. Applications

In this section, we use a DANO model of a bar of soap acquired from real images of the soap. We find the coefficient of friction of the soap from a real video of the soap sliding on a table. We then use a DANO model of the Stanford bunny (acquired from synthetic images of the bunny) to find the friction coefficient and mass scaling coefficient of the bunny (in simulation). Finally, we optimize a robot trajectory in simulation for pushing the bunny to a goal location using the differentiability of the simulation pipeline.

A. System Identification

We demonstrate how we can leverage differentiable simulation to efficiently estimate the dynamical properties of the DANO from object trajectories. For a given object, we parameterize our dynamics model with a vector $\theta \in \mathbb{R}^n$. This vector can include contact-model parameters, such as sliding or rolling-friction coefficients, and dynamics parameters, such as the mass and inertia of the object. We learn $\theta$ by minimizing the distance between a ground-truth trajectory and a simulated trajectory starting from the same initial conditions.

\[ \min_{\theta} \sum_{t=2}^{T} \| \dot{x}_t - x_t \|_V \]

subject to $x_{t+1} = f(x_t; \theta), \quad t = 1, \ldots, T - 1, \quad (17)$

where we indicate ground-truth quantities with the asterisk symbol, $x_1$ is the system’s initial condition, $\| \cdot \|_V$ is a weighted norm, $f$ is the dynamics parameterized by $\theta$, and $T$ is the number of time steps. $\theta_{\min}$ and $\theta_{\max}$ are bounds on the parameters enforcing basic constraints; for example, mass and friction coefficients.
To extract a mesh from a neural density, in simulation, we leverage interactive per-December 16, 2021 10:24 UTC from IEEE Xplore. Restrictions apply.

comparison

Fig. 6. Results on system identification.

Fig. 7. We solve a push-and-slide task using trajectory optimization. The objective is to push a dynamics-augmented neural object (bunny) using a fully-actuated spherical end effector (Panda arm). The goal positions of the end effector and the bunny are shown with white and black circular targets, respectively. Leveraging the simulator’s differentiability, we optimize with a gradient-based solver a highly dynamic trajectory where the bunny slides to reach its target position.

B. Trajectory Optimization

Simulating the DANO in a differentiable simulator such as Dojo [7] allows us to optimize robot trajectories that involve contact (e.g., grasping, manipulation, or pushing) using existing gradient-based optimization frameworks. For trajectory optimization, we minimize a cost functional over a time-discretized trajectory of the robot while respecting state and control-input constraints,

\[
\begin{align*}
\min_{x_1:T, u_1:T-1} & \quad \sum_{t=1}^{T-1} l_t(x_t, u_t) + l_f(x_T) \\
\text{subject to} & \\
& x_{t+1} = f_t(x_t, u_t), \quad t = 1, \ldots, T-1, \\
& x_1 = \bar{x}_1, \\
& c_t(x_t, u_t) \leq 0, \quad t = 1, \ldots, T-1, \\
& c_f(x_T) \leq 0,
\end{align*}
\]

where the subscript \(t\) indicates the time step, \(T\) is the number of time steps, \(x\) is the system state, \(u\) is the control input, \(\bar{x}_1\) is the system’s initial conditions, \(l_t\) and \(l_f\) are stage and final cost functions respectively, \(c_t\) and \(c_f\) are state and final constraints respectively, and \(f_t\) is the dynamics.

We optimize dynamic behaviors with contact on a push-and-slide task (Fig. 7). Specifically, a simulated Panda robot arm tries to push a DANO model of the Stanford bunny to a goal location, and return the Panda’s end effector to another goal location. We use a constrained iterative linear quadratic regulator (iLQR) solver [39, 40], which exploits gradients of the simulation.

VI. COMPARISON WITH MESH-BASED SIMULATION

It is natural to ask whether our DANO approach offers benefits over an approach leveraging existing methods and tools. A reasonable approach could be to convert the neural object representation into a mesh and pass this representation to an existing differentiable simulator. In this section, we investigate this approach to point out several unforeseen difficulties emerging with mesh-based simulation methods.

Level-set Selection. To extract a mesh from a neural density field, we apply the Marching Cubes algorithm [6, 41] on the underlying OSF density field. This extraction requires picking a single density level set from the OSF, which begs the question, which density value should we choose? To answer this question,
we extract a series of 3 meshes from an OSF density field of a bar of soap (constructed from real images). Similarly to DANO, we augment these level-set objects with dynamical properties (e.g. mass, inertia, friction). Then we simulate these level-set objects in an existing physics engine: MuJoCo [35]. Simulation results and visualizations of the level-set objects are presented in Fig. 8. Finding a suitable density value is a labor-intensive task, as visual inspection of the generated mesh is often not sufficient due to small masses floating around the object.

**Mesh Artifacts.** Once a density value is chosen, the resulting level-set object often features mesh artifacts preventing accurate contact simulation. For lower-density values (Fig. 8(a)), contact interaction with the ground is inaccurate due to masses floating around the object. Small artifacts persist even for larger-density values (Fig. 8(b)) and dictate the contact interactions. In contrast, DANO’s contact model is less sensitive to such artifacts. Indeed, small volumes with high densities floating around the object will contribute little to the contact interaction (5). For higher-density values (Fig. 8(c)), the object is mostly void, and contact interaction with robotic hardware such as a 2-finger gripper cannot be accurately simulated. To the best of our knowledge, there is no standard algorithm for fixing either of these two types of artifacts. Fixing these issues might require manual intervention or labor-intensive tuning of an ad-hoc method. Fig. 8(d) shows an image of the true object and Fig. 8(e) shows the pose of our DANO lying flatly on the ground at rest, but rendered with a mesh for easy visualization. Please note the poor quality of the mesh in Fig. 8(d) has nothing to do with the physics simulation, and is just for visualization.

**Contact Simulation.** Finally, existing physics engines targeting robotics applications have limited support for mesh-represented objects. Among the major simulators: MuJoCo [35], Bullet [32]; none can directly simulate non-convex mesh-represented objects. A pre-processing step decomposing the object into a set of convex shapes is required. Voxelized Hierarchical Approximate Convex Decomposition (V-HACD) [42] is a commonly used algorithm with its own set of tuning parameters; prominently the number of convex shapes used to represent the object. Comparatively, DANO’s approach has been demonstrated with non-convex objects like the Stanford bunny (Figs. 6 and 7) without requiring additional convex decomposition, manual tuning, or multiple algorithmic stages with human oversight and input.

Overall, the traditional mesh-based simulation approach requires, in practice, multiple mesh processing steps and labor-intensive parameter tuning involving several arbitrary choices (density value selection, convex decomposition). Our DANO approach bypasses mesh-related issues by directly operating on the density field.

**VII. DISCUSSIONS**

We presented the Dynamically-Augmented Neural Object (DANO), which appends a neural object model with essential dynamical information, making its motion simulatable in a physics simulation environment. We also propose a contact force model to compute normal and friction forces for the DANO. We see these tools as an effort to bridge the gap between perception and simulation in robotics. Ultimately, we hope this work is a step towards endowing a robot with the ability to autonomously build its own physics simulations of its environment using only its own sensor inputs—an essential ingredient of robot spatial intelligence.

**Limitations.** Our method does have some noteworthy limitations. (i) One limitation of our method is that our current implementation applies forces at a single point. One could instead consider all the sampled points independently to better handle torsional friction and strongly non-convex objects. (ii) Additionally, articulated body and soft body contact simulation is an interesting and important topic for future research. These kinds of objects are not covered in our current work, as rigid body objects present many challenges on their own. (iii) Another limitation of our method is that extracting rigid body pose trajectories from RGB video is challenging. Our pose extraction method in Section V-A uses a 2D mask-based loss, which has numerous local minima for object orientation and thus is sensitive to initialization. A better approach would be to solve for the pose trajectory by minimizing the photometric error between the actual video and a video generated by the neural object renderer and the differentiable simulator together. (iv) This points to the second limitation, which is that the method in this letter stops one step short of integrating the dynamics simulator with the renderer to give a differentiable end-to-end “torques-to-pixels” simulator. This is possible with the tools we describe, and we plan to pursue this as our immediate next step. Finally, (v) in real videos, the apparent color of a point on the object changes as it moves relative to the ambient light field, changing reflections, specularity, shadows, and color. We ignore these effects in this letter, although it is possible to reproduce such effects with the OSF model. It would be interesting to attempt to capture these lighting effects in an end-to-end differentiable “torques-to-pixels” simulator.

---

5 For instance, MuJoCo approximates non-convex meshes with a convex hull.
REFERENCES


