Differentiable Physics Simulation of Dynamics-Augmented Neural Objects

Simon Le Cleac'h, *Graduate Student Member, IEEE*, Hong-Xing Yu, Michelle Guo, Taylor Howell, Ruohan Gao, Jiajun Wu, *Member, IEEE*, Zachary Manchester, *Member, IEEE*, and Mac Schwager, *Member, IEEE*

***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.**

Manuscript received 17 October 2022; accepted 27 February 2023. Date of publication 15 March 2023; date of current version 3 April 2023. This letter was recommended for publication by Associate Editor E. Mingo Hoffman and Editor L. Pallottino upon evaluation of the reviewers' comments. This work was supported in part by Toyota Research Institute (TRI), in part by Qualcomm, in part by Amazon, in part by NSF CCRI under Grant 2120095, in part by NSF RI under Grant 2211258, in part by ONR MURI under Grant N00014-22-1-2740, and in part by the Stanford Institute for Human-Centered AI (HAI). *(Corresponding author: Simon Le Cleac'h.)*

Simon Le Cleac'h is with the Multi-robot Systems Laboratory, Stanford University, Stanford, CA 94305 USA, and also with the Robotic Exploration Laboratory, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: simonlc@stanford.edu).

Mac Schwager is with the Multi-robot Systems Laboratory, Stanford University, Stanford, CA 94305 USA (e-mail: schwager@stanford.edu).

Hong-Xing Yu, Michelle Guo, Ruohan Gao, and Jiajun Wu are with the Stanford Vision and Learning Laboratory, Stanford University, Stanford, CA 94305 USA (e-mail: koven@cs.stanford.edu; mguo95@cs.stanford.edu; rhgao@cs.stanford.edu; jiajunwu@cs.stanford.edu).

Taylor Howell and Zachary Manchester are with the Robotic Exploration Laboratory, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: thowell@stanford.edu; zacm@cmu.edu).

Video: https://youtu.be/Md0PM-wv_Xg

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3257707, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3257707

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

[1]The 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.

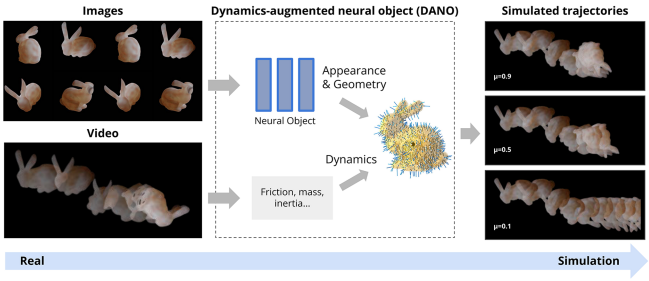


Fig. 1. Our pipeline for simulating the motion of neural objects. A neural object model such as a NeRF is trained from still images (top left), and object trajectories are extracted from videos of the object in motion (bottom left). Object mass and friction properties are computed and candidate surface 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.

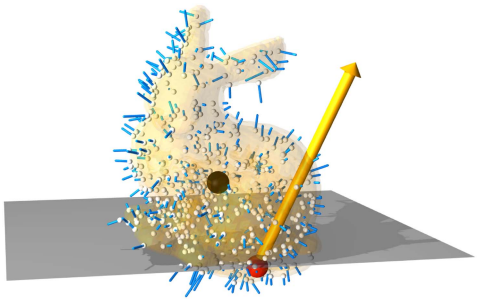


Fig. 2. DANO obtained from a neural object model of the Stanford bunny. We sample points (white dots) from the neural density field and compute the field gradients (blue arrows) to obtain an approximate outward normal, which is used to compute contact forces in our simulator. We estimate the mass, center 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 DiffCloud [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 rollout, 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

contact model into an existing differentiable simulator, Dojo [7]. Note that the proposed DANO and contact model can also be integrated with other implicit integration-based differentiable simulators [30], [32].

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 relightable 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 relightable, 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 $\rho = (\rho_r, \rho_g, \rho_b) \in \mathbb{R}^3$ is the cumulative radiance transfer to model the object appearance and $\phi(x)$ is the volumetric density that models the object geometry. The ρ function takes as input a 3D point $x \in \mathbb{R}^3$ in the object coordinate frame, an incoming distant light direction ω_{light}, and an outgoing radiance direction ω_{out} at that location, while the density $\phi(x)$ only requires the 3D point. The OSF can be re-posed in a background scene, or with respect to other neural objects through a pose transform applied to $x, \phi(Rx + \tau)$, where (R, τ) 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 ρ, 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:

$$\bar{\phi}(x) = \alpha \phi(x), \tag{1}$$

$$m = \int_{x \in \mathbb{R}^3} \bar{\phi}(x) dx \approx \sum_{x \in X} \bar{\phi}(x), \tag{2}$$

$$\mu = \frac{1}{m} \int_{x \in \mathbb{R}^3} x \bar{\phi}(x) dx \approx \frac{1}{m} \sum_{x \in X} x \bar{\phi}(x), \tag{3}$$

$$J = \int_{x \in \mathbb{R}^3} \bar{\phi}(x)(x^T x I - x x^T) dx \approx \sum_{x \in X} \bar{\phi}(x)(x^T x I - x x^T), \tag{4}$$

where we denote $\bar{\phi}$ the volumetric mass density field, α the scaling factor, m the mass, J the moment of inertia, and μ the center of mass. The parameter α 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 $\bar{\phi}$ 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 $\bar{\phi}$ 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 a 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 normal and friction 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 quantifying 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,

$$\psi = \int_{x \in \mathbb{R}^3} \phi^A(x) \phi^B(x) dx, \tag{5}$$

of the product of their density fields ϕ^A and ϕ^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 $\phi^A(x) = P(H_x^A = 1)$ is the probability of hitting object A when sampling a point x in $\mathbb{R}^3$.[4] Here, H_x^A is a random variable taking value 1 to indicate collision and 0 otherwise. Then assuming H_x^A and H_x^B are independent random variables (i.e., the shape

[2]Object-centric means that the model represents a single object in a body-fixed frame, and can be reposed through a rigid body transform with respect to a global frame. In contrast, the large majority of neural scene representations (like NeRF) have no notion of individual objects or poses.

[3]For tractability reasons, the gradient through collision detection is approximate, e.g. it does not take into account the differentiation of the contact normal between two neural objects.

[4]In practice, ϕ is an unnormalized density field; nevertheless, our contact model only requires computing volume ψ up to a scaling factor.

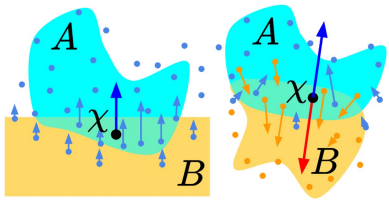


Fig. 3. Left: contact between a DANO (A) and the ground (half-space B). Each point sampled on the density field below ground level contributes to the repulsive force on the DANO. Points with a larger density value generate more force. The repulsive forces are applied at point χ, which represents the 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.

of one object is independent of the shape of the other), $\phi^A(x) \cdot \phi^B(x) = P(H_x^A = 1 \cap H_x^B = 1)$ is the probability of hitting both objects A and B when sampling point x. Finally, ψ (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 ψ.

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 contact forces,

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

When $\psi = 0$, there is no interpenetration, and the computation of χ is unnecessary.

Neural Object Sampling Procedure. Exact computation of the integrals in (5), (6) are 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 $\mathcal{B} = \{x \in \mathbb{R}^3 | f(x) \leq 0\}$ where, for example, $f(x) = a^T x + b$ 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 $\phi^B(x) = \mathbf{1}_{x \in \mathcal{B}}$, where $\mathbf{1}$ denotes the indicator function. Finally, we approximate the interpenetration volume and its centroid (5), (6) as sums over sampled points:

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

$$\chi \approx \frac{1}{\psi N} \sum_{x \in X^A \cap \mathcal{B}} x \phi^A(x). \tag{8}$$

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 ψ and χ, modifying the number of sampling points belonging to $\mathcal{B}$. For example, if object A is positioned halfway through the ground, ψ will be large, whereas if object A is above the ground, ψ 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 χ, e.g. $n = (\chi - c)/||\chi - c||_2$ 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

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

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

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 $n = \bar{n}/||\bar{n}||_2$. 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 ψ, applied at the centroid of the interpenetration volume χ, and applied in the direction normal to the contact surface, as shown in Fig. 4). We define the linear velocity $v \in \mathbf{R}^3$ of point χ attached to frame A with respect to frame B, and the angular velocity $\omega \in \mathbf{R}^3$ of frame A with respect to frame B. We apply a force normal to the contact to oppose interpenetration,

$$F_n = -\psi \left(I_{\text{spring}} n + I_{\text{damper}} v_n \right), \tag{11}$$

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

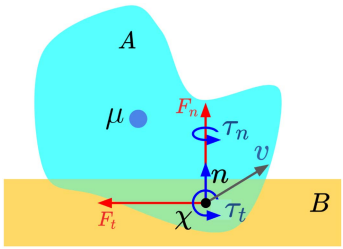


Fig. 4. Contact modeling between a DANO A and a primitive shape B (half-space). All forces and torques are applied at the centroid of the interpenetration volume χ. The normal force F_n opposing contact interpenetration is applied along the contact normal n. A tangential force F_t modeling sliding friction is applied. This force opposes the velocity of the point χ in the plane tangential to the contact. Similarly, rolling friction is applied via a torque τ_t in the tangential plane. Finally, τ_n generates torsional friction.

TABLE I
CONTACT MODEL PARAMETERS.

parameter	min	max	effect
impact spring	10^4	10^5	↗ stiffer impact
impact damper	10^5	10^6	↗ more stable simulation
sliding friction	0	1	↗ less sliding
sliding drag	0	0.1	↗ more stable simulation
rolling friction	0	0.1	↗ less rolling
rolling drag	0	0.1	↗ more stable simulation
torsional friction	0	0.1	↗ less spinning
torsional drag	0	0.1	↗ more stable simulation

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.

relative sliding between objects A and B

$$F_t = -||F_n|| \left(S_{\text{friction}} \frac{v_t}{||v_t||} + S_{\text{drag}} v_t \right), \tag{12}$$

where v_t is the tangential component of v, S_{friction} and S_{drag} are parameters modeling dry and viscous frictions. We apply a torque normal to the contact to encode torsional friction,

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

where ω_n is the normal component of ω, T_{friction} and T_{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,

$$\tau_t = -||F_n|| \left(R_{\text{friction}} \frac{\omega_t}{||\omega_t||} + R_{\text{drag}} \omega_t \right), \tag{14}$$

where ω_t is the tangential component of ω, T_{friction} and R_{drag} are parameters modeling dry and viscous rotational frictions. In Table I, we provide default values and an explanation of the effect of each parameter on contact simulation. We acknowledge that the simple contact model proposed in this letter is subject to creep, (e.g. an object slowly slides on an inclined plane instead of sticking). This is a limitation commonly seen in many existing physics engines [35]. To address this issue, an optimization-based modeling of friction forces and torques, as proposed in [7], could be implemented. To include contact normal and tangential forces and torques in the simulator, we compute F the contact force applied by body B on body A expressed in the world frame, τ the torque evaluated at the center of mass μ of body A expressed in body A's frame,

$$F = F_n + F_t, \tag{15}$$

$$\tau = \tau_n + \tau_t + \overrightarrow{\mu\chi} \times F. \tag{16}$$

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_Xgyoutu.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.

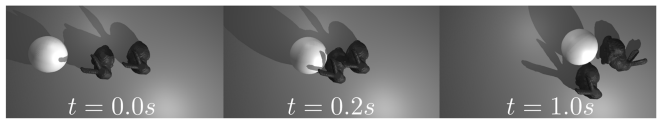


Fig. 5. We simulate an environment with a sphere hitting two neural objects (Stanford bunnies). During simulation, the two neural objects make and break contact propagating the impulse provided by the hitting sphere.

V. APPLICATIONS

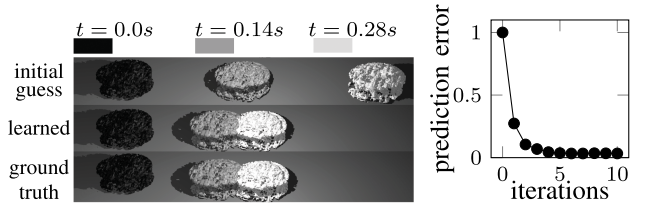
In this section, we use a DANO model of a bar of soap acquired from real images of the soap. We find the coefficient of friction of the soap from a real video of the soap sliding on a table. We then use a DANO model of the Stanford bunny (acquired from synthetic images of the bunny) to find the friction 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

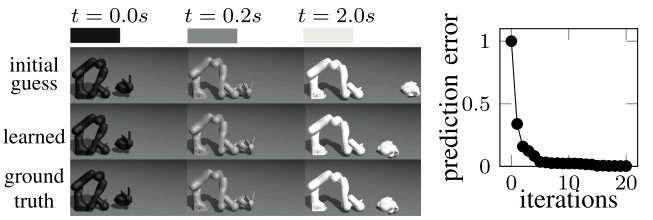
We demonstrate how we can leverage differentiable simulation to efficiently estimate the dynamical properties of the DANO from object trajectories. For a given object, we parameterize our dynamics model with a vector $\theta \in \mathbb{R}^d$. This vector can include contact-model parameters, such as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

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & \textstyle\sum_{t=2}^{T} \|\hat{x}_t - x_t\|_W^2 \\ \text{subject to} \quad & x_{t+1} = f(x_t; \theta), \quad t = 1, \ldots, T-1, \\ & x_1 = \hat{x}_1, \\ & \theta_{\text{min}} \leq \theta \leq \theta_{\text{max}}, \end{aligned} \tag{17}$$

where we indicate ground-truth quantities with the $\hat{}$ symbol, x_1 is the system's initial condition, $\|\cdot\|_W$ is a weighted norm, f is the dynamics parameterized by θ, and T is the number of time steps. θ_{min} and θ_{max} are bounds on the parameters enforcing basic constraints; for example, mass and friction coefficients



(a) Left: we visualize how closely the dynamics-augmented neural object with learned friction matches the ground-truth trajectory. Top: simulated trajectory poorly matching the ground truth using an initial guess for the sliding friction coefficient. Middle: simulated trajectory closely matching the ground-truth trajectory with learned sliding friction coefficients. Bottom: ground-truth trajectory extracted from a video of the soap sliding on the ground. Right: the trajectory prediction error (Problem 17) rapidly decreases after a few Newton steps.



(b) Left: we learn the mass, inertia scaling, and sliding friction coefficient of the bunny through interactive perception. We use a spherical end effector to push a neural object (bunny). This interaction facilitates the identification of dynamics parameters such as mass. Right: we closely match ground-truth parameters and trajectories using a dataset of 10 pushes.

Fig. 6. Results on system identification.

necessitate positive quantities. Leveraging the simulator's differentiability i.e., f's derivatives, we apply the Gauss-Newton method [36] to learn the system's parameters.

Soap Bar. We apply system identification on real-world data (Fig. 6(a)). We train an OSF model from a set of still images of a semi-translucent soap bar. Then we extract a pose trajectory from a video of the soap bar sliding on the ground. We formulate the pose tracking as a frame-wise optimization problem using only geometric information: $\min_p \|T(p) - M\|^2 + \|B(T(p)) - B(M)\|^2$, where p denotes the object pose at the current frame, M denotes a binary object mask extracted from the frame (we use the U ^{2} Net [37]), $T(p)$ denotes the accumulated transmittance map rendered using object pose p, and $B(\cdot)$ denotes the barycenter of the mask/transmittance map. The accumulated transmittance map is computed along with the volume rendering process [2], and the barycenter can be easily computed by the weighted mean of the mask/transmittance map multiplied by pixel coordinates. We solve this optimization problem using the Adam optimizer for all video frames to obtain the pose trajectory.

We learn the sliding friction coefficient between the soap and the table from this trajectory. The optimization process takes 2.0 seconds on a laptop equipped with an Intel i7-8750H CPU and 16 GB of RAM. The resulting trajectory simulated with learned friction closely matches the ground-truth trajectory. To find the ground truth coefficient of friction, we experimentally collected trajectory data where the bar of soap is sliding on a tilted plane with a known angle. We obtained a ground-truth value of $S_{\text{friction}} = 0.75$ which is close to the value obtained from system identification: $S_{\text{friction}} = 0.61$.

Stanford Bunny. In simulation, we leverage interactive perception [38] to identify the mass scaling factor α from (2),

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.

and sliding friction coefficient between a dynamics-augmented neural object and a surface (Fig. 6(b)). We implement a simple policy where the end effector pushes the DANO bunny with a known force, thereby allowing the identification of both the mass scale factor and friction coefficient. We successfully identify these parameters with less than 1.5% relative error from 10 pushing trajectories.

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,

$$\begin{aligned} \underset{x_{1:T}, u_{1:T-1}}{\text{minimize}} \quad & \sum_{t=1}^{T-1} l_t(x_t, u_t) + l_T(x_T) \\ \text{subject to} \quad & x_{t+1} = f_t(x_t, u_t), \quad t = 1, \ldots, T-1, \\ & x_1 = \hat{x}_1, \\ & c_t(x_t, u_t) \leq 0, \qquad t = 1, \ldots, T-1, \\ & c_T(x_T) \leq 0, \end{aligned} \tag{18}$$

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, $\hat{x}_1$ is the system's initial conditions, l_t and l_T are stage and final cost functions respectively, c_t and c_T are stage 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,

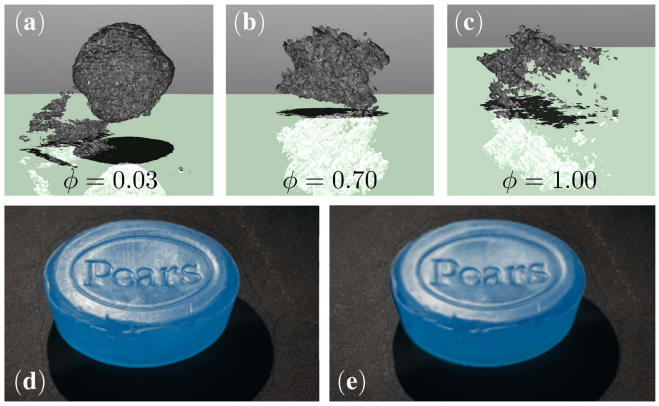


Fig. 8. (a)–(c) Meshes extracted from 3 different level set values of the neural density, simulated lying on the floor at rest in MuJoCo. (a) With a low level-set value of $\phi = 0.03$, spurious mesh artifacts float around the actual soap bar, preventing it from actually contacting the floor. (b) The level-set value $\phi = 0.70$ best captures the original shape of the soap bar, but there are still spurious artifacts that cause the soap to lie on the ground with an unnatural tilt. (c) The level-set $\phi = 1.00$ allows the soap to lie flat on the floor; however, the object is mostly void, leading to inaccurate mass, center of mass, and inertia matrix computations. Simulations with contact will be highly unreliable when most of the object is modeled as free space. (d) Picture of the soap bar, (e) Simulation of the DANO at rest on the floor, it lies flat safely ignoring spurious artefacts.

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.[5] 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.[6] 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 out 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.

[6]For completeness, we mention that we applied a convex-decomposition to each level-set mesh [43]. However, due to the chaotic surface of the meshes, the resulting decomposition featured sharp and tiny convex shapes that were not handled by the MuJoCo simulator.

REFERENCES

[1] H.-X. Yu et al., "Learning object-centric neural scattering functions for free-viewpoint relighting and scene composition," 2023, *arXiv:2303.06138*.

[2] B. Mildenhall, P.P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. Eur. Conf. Comput. Vis.*, pp. 405–421, 2020.

[3] B. Yang et al., "Learning object-compositional neural radiance field for editable scene rendering," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 13779–13788.

[4] R. Gao, Y.-Y. Chang, S. Mall, L. Fei-Fei, and J. Wu, "ObjectFolder: A dataset of objects with implicit visual, auditory, and tactile representations," in *Proc. Conf. Robot Learn.*, 2022, pp. 466–476.

[5] R. Gao et al., "ObjectFolder 2.0: A multisensory object dataset for sim2real transfer," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 10598–10608.

[6] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *ACM SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, 1987.

[7] T. A. Howell, S. Le Cleac'h, Z. Kolter, M. Schwager, and Z. Manchester, "DOJO: A differentiable simulator for robotics," 2022, *arXiv:2203.00806*.

[8] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene representation networks: Continuous 3D-structure-aware neural scene representations," in *Proc. 33rd Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 1121–1132.

[9] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 165–174.

[10] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3D reconstruction in function space," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4460–4470.

[11] K. Park et al., "Nerfies: Deformable neural radiance fields," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 5865–5874.

[12] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-NeRF: Neural radiance fields for dynamic scenes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 10318–10327.

[13] E. Tretschk, A. Tewari, V. Golyanik, M. Zollhöfer, C. Lassner, and C. Theobalt, "Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 12959–12970.

[14] K. Park et al., "HyperNeRF: A higher-dimensional representation for topologically varying neural radiance fields," *ACM Trans. Graph.*, vol. 40, no. 6, pp. 1–12, 2021.

[15] L. Liu, M. Habermann, V. Rudnev, K. Sarkar, J. Gu, and C. Theobalt, "Neural actor: Neural free-view synthesis of human actors with pose control," *ACM Trans. Graph.*, vol. 40, no. 6, pp. 1–16, 2021.

[16] Z. Li, S. Niklaus, N. Snavely, and O. Wang, "Neural scene flow fields forspace-time view synthesis of dynamic scenes," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 6498–6508.

[17] W. Xian, J.-B. Huang, J. Kopf, and C. Kim, "Space-time neural irradiance fields for free-viewpoint video," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 9421–9431.

[18] Y. Du, Y. Zhang, H.-X. Yu, J. B. Tenenbaum, and J. Wu, "Neural radiance flow for 4D view synthesis and video processing," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 14304–14314.

[19] S. Peng et al., "Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 9054–9063.

[20] C. Gao, A. Saraf, J. Kopf, and J.-B. Huang, "Dynamic view synthesis from dynamic monocular video," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 5712–5721.

[21] D. Driess, Z. Huang, Y. Li, R. Tedrake, and M. Toussaint, "Learning multi-object dynamics with compositional neural radiance fields," in *Proc. Conf. Robot Learn.*, 2022, pp. 1755–1768.

[22] Y. Li, S. Li, V. Sitzmann, P. Agrawal, and A. Torralba, "3D neural scene representations for visuomotor control," in *Proc. Conf. Robot Learn.*, 2022, pp. 112–123.

[23] B. Shen et al., "ACID: Action-conditional implicit visual dynamics for deformable object manipulation," 2022, *arXiv:2203.06856*.

[24] Q. Le Lidec, I. Kalevatykh, I. Laptev, C. Schmid, and J. Carpentier, "Differentiable simulation for physical system identification," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 3413–3420, Apr. 2021.

[25] P. Sundaresan, R. Antonova, and J. Bohgl, "Diffcloud: Real-to-sim from point clouds with differentiable simulation and rendering of deformable objects," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 10828–10835.

[26] S. Pfrommer, M. Halm, and M. Posa, "ContactNets: Learning of discontinuous contact dynamics with smooth, implicit representations," in *Proc. Conf. Robot Learn.*, 2020, pp. 2279–2291.

[27] J. K. Murthy et al., "GradSim: Differentiable simulation for system identification and visuomotor control," in *Proc. Int. Conf. Learn. Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=c_E8kFWfhp0

[28] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 3400–3407.

[29] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, vol. 31, pp. 7178–7189.

[30] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and feature-complete differentiable physics for articulated rigid bodies with contact," in *Proc. Robot.: Sci. Syst. Conf.*, 2021.

[31] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, "BRAX–A differentiable physics engine for large scale rigid body simulation," 2021, *arXiv:2106.13281*.

[32] E. Heiden, D. Millard, E. Coumans, Y. Sheng, and G. S. Sukhatme, "NeuralSim: Augmenting differentiable simulators with neural networks," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021.

[33] J. Lee et al., "DART: Dynamic animation and robotics toolkit," *J. Open Source Softw.*, vol. 3, no. 22, 2018, Art. no. 500.

[34] R. Elandt, E. Drumwright, M. Sherman, and A. Ruina, "A pressure field model for fast, robust approximation of net contact force and moment between nominally rigid objects," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 8238–8245.

[35] E. Todorov, T. Erez, and Y. Tassa, "MujoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026–5033.

[36] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Berlin, Germany: Springer, 2006.

[37] X. Qin, Z. Zhang, C. Huang, M. Dehghan, O. R. Zaiane, and M. Jagersand, "U2-Net: Going deeper with nested U-structure for salient object detection," *Pattern Recognit.*, vol. 106, 2020, Art. no. 107404.

[38] J. Bohg et al., "Interactive perception: Leveraging action in perception and perception in action," *IEEE Trans. Robot.*, vol. 33, no. 6, pp. 1273–1291, Dec. 2017.

[39] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *Proc. Int. Conf. Informat. Control Automat. Robot.*, 2004, pp. 222–229.

[40] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A fast solver for constrained trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 7674–7679.

[41] T. S. Newman and H. Yi, "A survey of the marching cubes algorithm," *Comput. Graph.*, vol. 30, no. 5, pp. 854–879, 2006.

[42] K. Mamou, E. Lengyel, and A. Peters, "Volumetric hierarchical approximate convex decomposition," in *Game Engine Gems 3*. Natick, MA, USA: AK Peters, 2016, pp. 141–158.

[43] X. Wei, M. Liu, Z. Ling, and H. Su, "Approximate convex decomposition for 3D meshes with collision-aware concavity and tree search," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 1–18, 2022.