TossingBot: Learning to Throw Arbitrary Objects With Residual Physics

Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser

***Abstract*—We investigate whether a robot arm can learn to pick and throw arbitrary rigid objects into selected boxes quickly and accurately. Throwing has the potential to increase the physical reachability and picking speed of a robot arm. However, precisely throwing *arbitrary objects* in unstructured settings presents many challenges: from acquiring objects in grasps suitable for reliable throwing, to handling varying object-centric properties (e.g., mass distribution, friction, shape) and complex aerodynamics. In this work, we propose an end-to-end formulation that jointly learns to infer control parameters for grasping and throwing motion primitives from visual observations (RGB-D images of arbitrary objects in a bin) through trial and error. Within this formulation, we investigate the synergies between grasping and throwing (i.e., learning grasps that enable more accurate throws) and between simulation and deep learning (i.e., using deep networks to predict residuals on top of control parameters predicted by a physics simulator). The resulting system, *TossingBot*, is able to grasp and successfully throw arbitrary objects into boxes located outside its maximum reach range at 500+ mean picks per hour (600+ grasps per hour with 85% throwing accuracy); and generalizes to new objects and target locations.**

***Index Terms*—Deep learning, perception, robotic manipulation.**

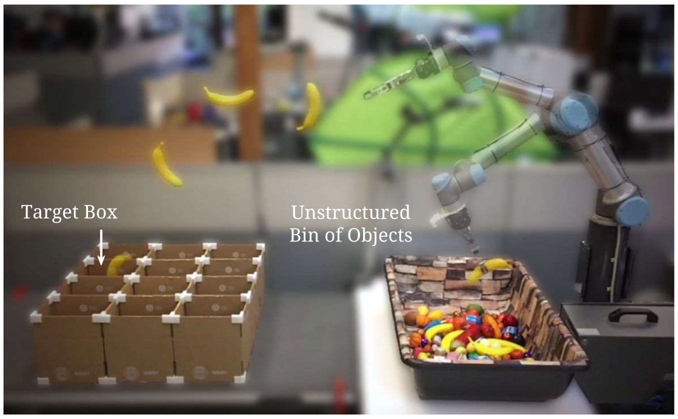

Fig. 1. *TossingBot* learns to grasp arbitrary objects from an unstructured bin and to throw them into target boxes located outside its maximum kinematic reach range. The aerial trajectory of different objects are controlled by jointly optimizing grasping policies and throwing release velocities.

I. Introduction

THROWING is a means to increase the capabilities of a manipulator by exploiting dynamics, a form of dynamic extrinsic dexterity [5]. In the case of pick-and-place, throwing enables a robot arm to place objects rapidly into boxes located outside its maximum kinematic range, which not only reduces the total physical space used by the robot, but also maximizes its picking efficiency. Rather than having to transport objects to their destination before executing the next pick, objects are instead immediately "passed to Newton" (see Fig. 1).

However, precisely throwing *arbitrary objects* in unstructured settings is challenging because it depends on many factors: from prethrow conditions (e.g., initial grasp of the object) to varying object-centric properties (e.g., mass distribution, friction, shape) and dynamics (e.g., aerodynamics). For example, grasping a screwdriver near the tip before throwing it can cause centripetal accelerations to swing it forward with significantly higher release velocities—resulting in drastically different projectile trajectories than if it were grasped closer to its center of mass (CoM) (see Fig. 2). Yet regardless of how it is grasped, its aerial trajectory would differ from that of a thrown ping pong ball, which can significantly decelerate after release due to air resistance. Many of these factors are notoriously difficult to model or measure analytically [22], [25]—hence, prior studies are often confined to assuming homogeneous prethrow conditions (e.g., object fixtured in gripper or manually reset after each throw) with predetermined, homogeneous objects (e.g., balls or darts). Such assumptions rarely hold in real unstructured settings, where a throwing system needs to acquire its own prethrow conditions (via grasping) and adapt its throws to account for varying properties and dynamics of arbitrary objects.

In this work, we present *TossingBot*, an end-to-end formulation that uses trial and error to learn how to plan control

Manuscript received September 27, 2019; accepted March 24, 2020. Date of publication June 1, 2020; date of current version August 5, 2020. This work was supported in part by the Google, in part by Amazon, in part by Intel, in part by NVIDIA, in part by ABB Robotics, and in part by Mathworks. This paper was presented in part at the Robotics: Science and Systems (RSS), Breisgau, Germany, 2019 [40]. This paper was recommended for publication by Associate Editor L. Natale and Editor P. Robuffo Giordano upon evaluation of the reviewers' comments. *(Corresponding author: Andy Zeng.)*

Andy Zeng and Thomas Funkhouser are with the Princeton University, Princeton, NJ 08540 USA, and also with the Google, Mountain View, CA 94043 USA (e-mail: andyz@cs.princeton.edu; funk@cs.princeton.edu).

Shuran Song is with the Princeton University, Princeton, NJ 08540 USA and with the Google, Mountain View, CA 94043 USA, and also with the Columbia University, New York, NY 10027 USA (e-mail: shurans@princeton.edu).

Johnny Lee is with the Google, Mountain View, CA 94043 USA (e-mail: jcl5m1@gmail.com).

Alberto Rodriguez is with the Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: albertor@mit.edu).

This article has supplementary downloadable material available at http://tossingbot.cs.princeton.edu. The video provides a summary of the paper's contributions, as well as qualitative videos of the robot in action. The size of the video is 18 MB.

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2020.2988642

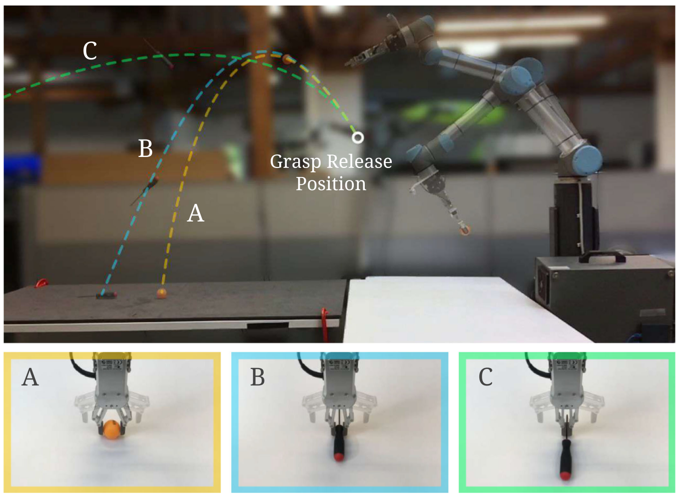

Fig. 2. (a) *Projectile trajectories* of a thrown ping pong ball, (b) screwdriver grasped and thrown by its handle, and (c) same screwdriver grasped and thrown by its shaft. The difference between (a) and (b) is largely due to aerodynamics, while the difference between (b) and (c) is largely due to grasping at different offsets from the object's CoM (near the handle). Our goal is to learn joint grasping and throwing policies that can compensate for these differences to achieve accurate targeted throws.

parameters for grasping and throwing from visual observations. The formulation learns grasping and throwing jointly—discovering grasps that enable accurate throws, while learning throws that compensate for the dynamics of arbitrary objects. There are the following two key aspects to our system.

1) *Joint learning of grasping and throwing policies:* With a deep neural network that maps from visual observations (of objects in a bin) to control grasping and throwing parameters: the likelihood of grasping success for a dense pixel-wise sampling of end effector orientations and locations [42], and the throwing release velocities for each sampled grasp. Grasping is directly supervised by the accuracy of throws (grasp success $=$ accurate throw), while throws are directly conditioned on specific grasps (via dense predictions). As a result, the end-to-end policy learns to execute stable grasps that lead to predictable throws, as well as throwing velocities that account for the variations in object-centric properties and dynamics that can be inferred from visual information.
2) *Residual learning of throw release velocities:* δ on top of velocities $\hat{v}$ predicted by a physics controller based on an ideal ballistic motion. The complete controller uses the superposition of the two predictions to obtain a final throwing release velocity $v = \hat{v} + \delta$. The physics-based controller uses ballistics to provide consistent estimates of $\hat{v}$ that generalize well to different landing locations, while the data-driven residuals learn to exploit those grasps, and compensate for object-centric properties and dynamics. Our experiments show that this hybrid data-driven method, *residual physics*, leads to significantly more accurate throws than baseline alternatives.

This formulation enables our system to grasp and throw arbitrary objects reliably into target boxes located outside its maximum reach range at 500+ mean picks per hour (MPPH), and generalizes to new objects and target landing locations.

The primary contribution of this paper is to provide new perspectives on throwing: in particular—its relationship to grasping, its efficient learning by combining physics with trial and error, and its potential to improve practical real-world picking systems. We provide several experiments and ablation studies in both simulated and real settings to evaluate the key components of our system. We observe that throwing performance strongly correlates with the quality of grasps, and experimental results show that our formulation is capable of learning synergistic grasping and throwing policies for arbitrary objects in real settings. An after-the-fact analysis of what the deep network learns, shows that the deep features internal to TossingBot effectively use visual appearance to cluster objects based on geometric and physical attributes—without any explicit supervision other than the goal to throw with accuracy. This journal paper is a revision of a conference paper appearing in Robotics: Science and Systems (RSS) 2019.

Most importantly, this version includes additional experiments in Section VI-H to visualize emerging visual deep features learned by TossingBot, demonstrating that it is possible to implicitly learn object-level semantics from physical interactions alone. Other additional changes include: more system details (i.e., on training and timing), algorithmic details on inferring throwing primitive parameters $\|v\|$ and r, details of human baseline experiments, visualizations and analysis of learned grasps, and an illustration of the simulation environment.

II. Related Work

A. Analytical Models for Throwing

Many previous systems built for throwing [10], [25], [26], [33], [35] rely on handcrafting or approximating dynamics based on frictional rigid body mechanics, and then optimizing control parameters to execute a throw such that the projectile (typically a ball) lands at a target location. However, as highlighted by Mason and Lynch in [25], accurately modeling throwing dynamics is challenging. It requires knowledge of physical properties that are difficult to estimate (e.g., aerodynamics, inertia, coefficients of restitution, friction, shape, mass distribution, etc.) for both objects and manipulators. As a result, these model-based systems often observe limited throwing accuracy (e.g., 40% success rate in [33]), and have difficulty generalizing to changing dynamics over time (e.g., deteriorating friction on gripper finger contact surfaces from repeated throwing). In our work, we leverage deep learning and self-supervision to compensate for the dynamics that are not explicitly accounted for in contact/ballistic models, and we train our policies online via trial and error so that they can adapt to new situations on the fly (e.g., new object and manipulator dynamics).

B. Learning Models for Throwing

More recently, learning-based systems for robotic throwing [2], [11], [16], [20] have also been proposed, which ignore low-level dynamics and directly optimize for task-level success

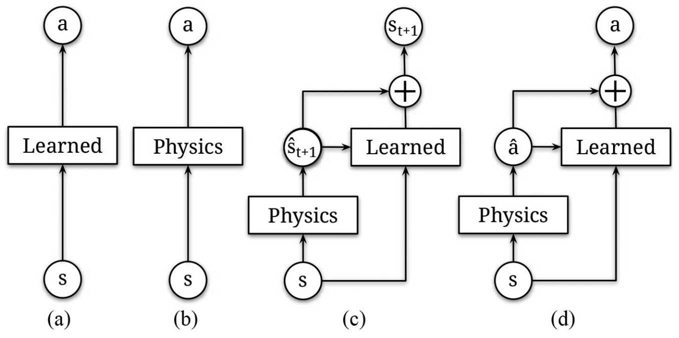

Fig. 3. Learning residual models and policies. (a) Analytical solutions that determine action a from state s. (b) Data-driven policies that learn the direct mapping from states to actions. (c) Hybrid models that combine analytical models with learning to predict future states s_{t+1}. (d) Hybrid policies (like ours) that combine analytical solutions with learning to determine action a.

signals (e.g., did the projectile land on the target?). These methods have demonstrated better accuracy than those which solely rely only on analytical models, but have two primary drawbacks: 1) limited generalization to new object types (beyond balls, blocks, or darts), and 2) limited prethrow conditions (e.g., human operators are required to manually reset objects and manipulators to match a prescribed initial state before every throw), which makes training from trial and error costly. Both drawbacks prevent their use in real unstructured settings.

In contrast to prior work, we make no assumptions on the physical properties of thrown objects, nor do we assume that the objects are at a fixed pose in the gripper before each throw. Instead, we propose an object-agnostic pick-and-throw formulation that jointly learns to acquire its own prethrow conditions (via grasping) while learning throwing control parameters that compensate for varying object properties and dynamics. The system learns through self-supervised trial and error, and resets it own training so that human intervention is kept at a minimum.

C. Learning Residual Models and Policies

Our approach to data-efficient learning, *residual physics*, falls under a broader category of hybrid controllers [1], [15], [29] that leverage both 1) analytical models to provide initial estimates of control parameters, and 2) learned residuals on top of those estimates to compensate for unknown dynamics [see Fig. 3(d)]. In contrast to prior work on learning residuals on predictions of future states for model-based control [3], [19] or data-augmented models [9], [17], [36], we instead directly learn the residuals on control parameters (i.e., action space) with deep networks. This approach provides a wider range of data-driven corrections that can compensate for noisy observations as well as dynamics that are not explicitly modeled. These benefits are also observed in concurrent work on residual reinforcement learning [18], [34] in block-assembly and object manipulation tasks.

III. Method Overview

TossingBot consists of a neural network $f(I, p)$ that takes as input a visual observation I of objects in a bin and the three-dimensional (3-D) position of a target landing location p, and outputs a prediction of parameters ϕ_g and ϕ_t used by two motion primitives for grasping and throwing, respectively, (see Fig. 4). The learning objective is to optimize the predictions of parameters ϕ_g and ϕ_t such that executing the grasping primitive using ϕ_g followed by the throwing primitive using ϕ_t results in an object (observed in I) landing on p at each time-step.

The network f consists of the following three parts:

1) a perception module that accepts visual input I and outputs a spatial feature representation μ; this is shared as input into;
2) a grasping module that predicts ϕ_g;
3) a throwing module that predicts ϕ_t.

f is trained end-to-end through self-supervision from trial and error by tracking the ground truth landing positions of thrown objects. The following sections provide an overview of these three modules, while the following two sections delve into details of the most novel aspects of the system.

A. Perception Module: Learning Visual Representations

We represent the visual input I as an RGB-D heightmap image of the workspace (i.e., a bin of objects). To compute this heightmap, we capture RGB-D images from a fixed-mount camera, project the data onto a 3-D point cloud, and orthographically back-project upward in the gravity direction to construct a heightmap image representation with both color (RGB) and height-from-bottom (D) channels. The RGB and D channels are normalized (mean-subtracted and divided by standard deviation from a prerecorded dataset of 100 images) so that learned convolutional filters can be shared across the two modalities.

The edges of the heightmaps are defined with respect to the boundaries of the robot's picking workspace. In our experiments, this area covers a 0.9×0.7 m tabletop surface, on top of which we place a bin of objects. Our heightmaps have a pixel resolution of 180×140, hence, each pixel $i \in I$ represents a 5×5 mm vertical column of 3-D space in the robot's workspace. Using its height-from-bottom value, each pixel thereby corresponds to a unique 3-D location in the robot's workspace. The input I is fed into the perception network, a seven-layer fully convolutional residual network [4], [14], [21] (interleaved with two layers of spatial 2×2 max-pooling), which outputs a spatial feature representation μ of size $45 \times 35 \times 512$ that is then fed into the grasping and throwing modules.

B. Grasping Module: Learning Parallel-Jaw Grasps

The grasping module consists of a grasping network that predicts the probability of grasping success for a predefined grasping primitive across a dense pixel-wise sampling of end effector locations and orientations in I.

1) Grasping Primitive: The grasping primitive takes as input parameters $\phi_g = (x, \theta)$ and executes a top-down parallel-jaw grasp centered at a 3-D location $x = (x_x, x_y, x_z)$ oriented θ° around the gravity direction. During execution, the open gripper approaches x along the gravity direction until the 3-D position of the middle point between the gripper fingertips meets x, at which point the gripper closes, and lifts upward 10 cm. This primitive

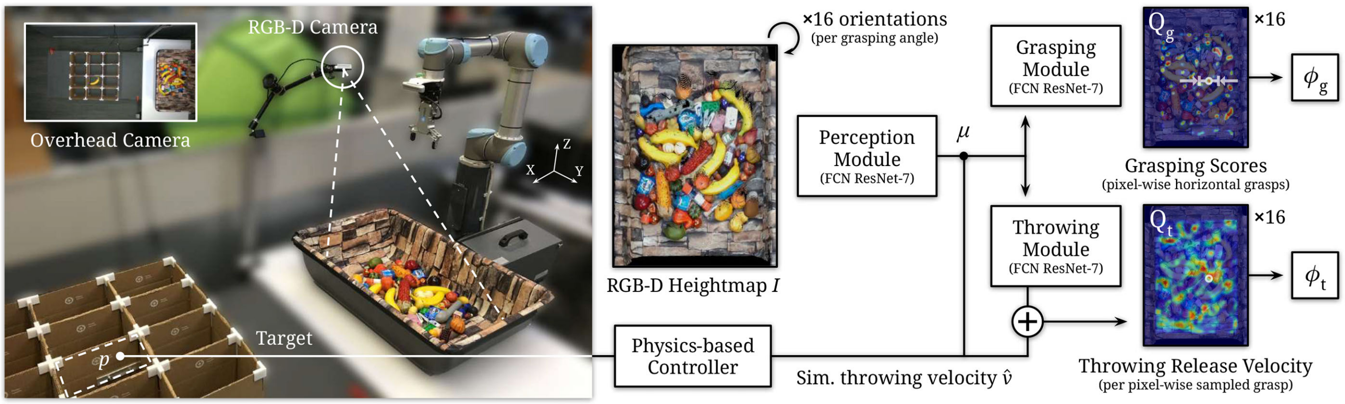

Fig. 4. *Overview.* An RGB-D heightmap of the scene is fed into a perception module to compute spatial features μ. In parallel, target location p is fed into a physics-based controller to provide an initial estimate of throwing release velocity $\hat{v}$, which is concatenated with μ then fed into grasping and throwing modules. Grasping module predicts probability of grasp success for a dense pixel-wise sampling of horizontal grasps, while throwing module outputs dense prediction of residuals (per sampled grasp), which are added to $\hat{v}$ to get final predictions of throwing release velocities. We rotate input heightmaps by 16 orientations to account for 16 grasping angles. Robot executes the grasp with the highest score, followed by a throw using its corresponding predicted velocity.

is open-loop, with robot arm motion planning executed using a stable, collision-free IK solver [8].

2) Grasping Network: The grasping network is a seven-layer fully convolutional residual network [4], [14], [21] (interleaved with two layers of spatial bilinear $2\times$ upsampling). This accepts the visual feature representation μ as input, and outputs a probability map Q_g with the same image size and resolution as that of the input heightmap I. Each value of a pixel $q_i \in Q_g$ represents the predicted probability of grasping success (i.e., grasping affordance) when executing a top-down parallel-jaw grasp centered at the 3-D location of $i \in I$ with the gripper oriented horizontally with respect to the heightmap I.

As in [38], [39], [42], and [43], we account for different grasping angles by rotating the input heightmap by 16 orientations (multiples of 22.5°) before feeding into the network. Rotations are done with nearest-neighbor sampling to avoid blurring artifacts, as well as 0-padding to maintain fixed image sizes. The pixel with the highest predicted probability among all 16 maps determines the parameters $\phi_g = (x, \theta)$ for the grasping primitive to be executed: the 3-D location of a pixel determines the grasping position x, and the orientation of the heightmap determines grasping angle θ. This visual state and action representation has been shown to provide sample efficiency when used in conjunction with fully convolutional action-value functions for grasping and pushing [42], [43]. Each pixel-wise prediction shares convolutional features for all grasping locations and orientations (i.e., translation and rotation equivariance).

C. Throwing Module: Learning Throwing Velocities

The goal of the throwing module is to predict the release position and velocity of a predefined throwing primitive for each possible grasp (over the dense pixel-wise sampling of end effector locations and orientations in I).

1) Throwing Primitive: The throwing primitive takes as input parameters $\phi_t = (r, v)$ and executes an end effector trajectory such that the mid-point between the gripper fingertips reaches a desired release position $r = (r_x, r_y, r_z)$ and velocity $v = (v_x, v_y, v_z)$, at which point the gripper opens and releases the object. During execution, the robot arm curls inward while grasping onto an object, then uncurls outward at high speed, releasing the object at the desired position and velocity. Throughout this motion, the gripper is oriented such that the axis between the fingertips is orthogonal to the plane of the intended aerial trajectory. In our system, the direction of curling/uncurling aligns with (v_x, v_y). Fig. 2 visualizes this motion primitive and its end effector trajectory. The throwing primitive is executed after each successful grasp attempt (checked by thresholding the distance between fingertips).

2) Planning the Release Position: In most real-world settings, only a handful of release positions are physically accessible by the robot for throwing. So for simplicity in our system, we directly constrain the release position r from the given target landing location p using two assumptions: 1) the aerial trajectory of a projectile is linear on the xy-horizontal-plane and in the same direction as $v_{x,y} = (v_x, v_y)$. In other words, we assume that the forces of aerodynamic drag *orthogonal* to $v_{x,y}$ are negligible. This is not to be confused with the primary forces of drag that exist in *parallel* to $v_{x,y}$, which our system will learn to compensate. We also assume 2) that the release distance $\sqrt{r_x^2 + r_y^2}$ is at a fixed value c_d from the robot base origin, and that r_z is at a fixed constant height c_h. Formally, these constraints can be written as: $(r_{x,y} - p_{t_{x,y}}) \times v_{x,y} = 0$ and $\sqrt{r_x^2 + r_y^2} = c_d$ and $r_z = c_h$. In our experiments, we select constant values of c_h and c_d such that all release positions are accessible by the robot: $c_h = 0.04$ m and $c_d = 0.7$ m in simulation, and $c_h = 0.02$ m and $c_d = 0.76$ m in real settings.

3) Planning the Release Velocity: Given a target landing location p and release position r, there could be multiple solutions of the release velocity v for which the object lands on p. To remove this ambiguity, we further constrain the direction of v to be angled 45° upward in the direction of p. Formally, this constraint can be defined as $\|v_{x,y}\| = v_z$. Under all the aforementioned constraints, the only unknown variable for throwing is $\|v_{x,y}\|$, which represents the magnitude of the final release velocity. Specifically, assuming a fixed throwing release height r_z, fixed release distance c_d from robot base origin, and release velocity direction angled 45° upward: for any given target landing location $p = (p_x, p_y, p_z)$, we can derive a release position r and release velocity magnitude $\|v\|$ that achieves the target landing location p assuming equations of linear projectile motion

$$\begin{aligned}\theta &= \arctan\left(\frac{p_y}{p_x}\right)\\ r_x &= c_d \sin(\theta)\\ r_y &= c_d \cos(\theta)\end{aligned} \tag{1}$$

$$\|v\| = \sqrt{\frac{a(p_x^2 + p_y^2)}{(r_z - p_z - \sqrt{p_x^2 + p_y^2})}} \tag{2}$$

where a is acceleration from gravity.

These equations are valid for any given target landing location p, as long as both $\|v\|$ and r are within robot physical limits. Hence assuming no aerial obstacles, varying only the velocity magnitude $\|v\|$ is sufficient to cover the space of all possible projectile landing locations. In the following section, we describe how the throwing module predicts $\|v_{x,y}\|$.

IV. Learning Residual Physics for Throwing

A key aspect of TossingBot's throwing module is that it learns to predict a residual δ on top of the estimated release velocity $\|\hat{v}_{x,y}\|$ from a physics-based controller (i.e., ballistic equations of projectile motion), then uses the superposition of the two predictions to compute a final release velocity $\|v_{x,y}\| = \|\hat{v}_{x,y}\| + \delta$ for the throwing primitive. Conceptually, this enables our models to leverage the advantages of physics-based controllers (e.g., generalization via analytical models), while still maintaining the capacity (via data-driven residual δ) to account for aerodynamic drag and offsets to the real-world projectile velocity (conditioned on the grasp), which are otherwise not analytically modeled. Our experiments in Section VI show that this approach, a.k.a. *residual physics*, even when using a simple ballistics model, yields significant improvements in both accuracy and generalization of throwing arbitrary objects compared to baseline alternatives: e.g., using only the physics-based controller [see Fig. 3(a)], or a fully data-driven training of f to regress $\|v_{x,y}\|$ [see Fig. 3(b)].

1) Physics-Based Controller: The physics-based controller uses the standard equations of linear projectile motion, by assuming a grasp on the CoM of the object, to analytically solve back for the release velocity $\hat{v}$ given the target landing location p and release position r of the throwing primitive: $p = r + \hat{v}t + \frac{1}{2}at^2$. This controller assumes that the object is a point particle and that the aerial trajectory of the projectile moves along a ballistic path affected only by gravity, which imparts a downward acceleration $a_z = -9.8$ m/s^2.

We also provide the estimated physics-based release velocity $\hat{v}$ as input into both the grasping and throwing networks by concatenating the visual feature representation μ with a k-channel image ($k = 128$) where each pixel holds the value of $\hat{v}$, repeated across channels. Providing $\hat{v}$ as input enables our grasping and throwing predictions to be conditioned on $\hat{v}$ – i.e., larger values of $\hat{v}$ for farther target locations can lead to a different set of effective grasps.

This physics-based controller has several advantages in that it provides a closed-form solution, generalizes well to new landing locations p, and serves as a consistent approximation for v. However, it also relies on several assumptions that do not generally hold. First, it assumes that the effects of aerodynamic drag are negligible. However, as we show in our experiments in Fig. 2, the aerial trajectory for lightweight objects like ping pong balls can be substantially influenced by drag. Second, it assumes that the gripper release velocity v directly determines the velocity of the projectile. This is not true since the object is often not grasped at the CoM, nor is the object completely immobilized by the grasp prior to release. For example, as illustrated in Fig. 2, a screwdriver picked up by the shaft can be flung forward with a significantly higher velocity than the gripper release velocity due to centripetal forces, resulting in a farther aerial trajectory.

2) Residual Physics-Based Controller: To compensate for the shortcomings of the physics-based controller, the throwing module includes a throwing network that predicts a residual δ on top of the estimated release velocity $\|\hat{v}_{x,y}\|$ for each possible grasp. The throwing network is a seven-layer fully convolutional residual network [14] interleaved with two layers of spatial bilinear 2× upsampling that accepts the visual feature representation μ as input, and outputs an image Q_t with the same size and resolution as that of the input heightmap I. Q_t has a pixel-wise one-to-one spatial correspondence with I, thus, each pixel in Q_t also corresponds one-to-one with the pixel-wise probability predictions of grasping success $q_i \in Q_g$ (for all possible grasps using rotating input I). Each pixel in Q_t holds a prediction of the residual value δ_i added on top of the estimated release velocity $\|\hat{v}_{x,y}\|$ from a physics-based controller, to compute the final release velocity v_i of the throwing primitive after executing the grasp at pixel i. The better the prediction of δ_i, the more likely the grasped and thrown object will land on the target location p.

V. Jointly Learning Grasping and Throwing

Our full network f (including the perception, grasping, and residual throwing modules) is trained end-to-end using the following loss function: $\mathcal{L} = \mathcal{L}_g + y_i\mathcal{L}_t$, where $\mathcal{L}_g$ is the binary cross-entropy error from predictions of grasping success classification

$$\mathcal{L}_g = -(y_i \log q_i + (1 - y_i)\log(1 - q_i))$$

and $\mathcal{L}_t$ is the continuous Huber loss from regressing the residual throwing velocity δ_i

$$\mathcal{L}_t = \begin{cases} \frac{1}{2}(\delta_i - \bar{\delta}_i)^2, & \text{for } |\delta_i - \bar{\delta}_i| < 1 \\ |\delta_i - \bar{\delta}_i| - \frac{1}{2}, & \text{otherwise}\end{cases}$$

where y_i is the binary ground truth grasp success label and $\bar{\delta}_i$ is the ground truth residual label. We use a Huber loss [12] instead of an MSE loss for regression since we find that it is less sensitive to inaccurate outlier labels. We pass gradients only through the single pixel i on which the grasping primitive was executed. All other pixels backpropagate with 0 loss.

We train our network f by stochastic gradient descent with momentum, using fixed learning rates of 10^{-4}, momentum of 0.9, and weight decay 2^{-5}. Our models are trained from scratch (i.e., random Xavier initialization [13], without any image-based pretraining [37]) in PyTorch with an NVIDIA Titan X on an Intel Xeon CPU E5-2699 v3 clocked at 2.30GHz. We train with prioritized experience replay [32] using stochastic rank-based prioritization, approximated with a power-law distribution. Our exploration strategy is ϵ-greedy, with ϵ initialized at 0.5 then annealed over training to 0.1. Specifically, when executing a grasp, the robot has an ϵ chance to sample a random grasp within the robot's workspace for picking; likewise when executing a throw, the robot has an ϵ chance to explore a random positive release velocity.

1) Network Architecture Details: Our network architecture consists of the following layers for each module:

1) Perception: C(3,64)-MP-RB(128)-MP-RB(256)-RB(512);
2) Grasping: RB(256)-RB(128)-UP-RB(64)-UP-C(1,2);
3) Throwing: RB(256)-RB(128)-UP-RB(64)-UP-C(1,1).

where C(k, c) denotes a convolutional layer with $k \times k$ filters and c channels, RB(c) denotes a residual block [14] with two convolutional layers using 3×3 filters and c channels, MP denotes a 3×3 max pooling layer with stride $= 2$, and UP denotes a bilinear $2\times$ upsampling layer.

2) Training Via Self-Supervision: We obtain our ground truth training labels y_i and $\bar{\delta}_i$ through trial and error. At each training step, the robot captures RGB-D images to construct visual input I, performs a forward pass of $f(I, p)$ to make a prediction of primitive parameters ϕ_g and ϕ_t, executes the grasping primitive using ϕ_g, then executes the throwing primitive using ϕ_t. We obtain ground truth grasp success labels y_i by one of the following two ways.

1) Success after grasping, by checking the distance between gripper fingertips after the grasping primitive.
2) Success after throwing, by checking the binary signal of whether or not a throw lands in the correct box.

As we show in Section VI-F, supervising grasps by the accuracy of throws eventually leads to more stable grasps and better overall throwing performance. The grasping policy learns to favor grasps that lead to successful throws, which is a stronger requirement than simple grasp success. After each throw, we approximate the object's actual landing location $\hat{p}$ using a calibrated overhead RGB-D camera to detect, which bin it landed in (i.e., detect visual changes in the landing zone before and after the throw). Regardless of where the object lands, its actual landing location $\hat{p}$ and the executed release velocity v is recorded and saved to the experience replay buffer as a training sample, with which we obtain the ground truth residual label $\bar{\delta}_i = \|v_{x,y}\| - \|\hat{v}_{x,y}\|_{\hat{p}}$.

In experiments in Section VI, we train our models by self-supervision with the same procedure: n objects are randomly dropped into the 0.9×0.7 m workspace in front of the robot. The robot performs data collection until the workspace is void of objects, at which point n objects are again randomly dropped into the workspace. In simulation $n = 12$, while in real-world experiments $n = 80$+. In our real-world setup, the landing zone (on which target boxes are positioned) is slightly tilted at a 15° angle adjacent to the bin to simplify resetting. When the workspace is void of objects, the robot lifts the bottomless boxes such that the objects slide back into the bin. In this way, human intervention is kept at a minimum during the training process.

VI. Evaluation

We execute a series of experiments in simulated and real settings to evaluate the learned grasping and throwing policies. The goal of the experiments are four-fold:

1) to evaluate the overall accuracy and efficiency of our pick-and-throw system on arbitrary objects,
2) to test its generalization to new objects and target locations unseen during training,
3) to investigate how learned grasps can improve the accuracy of subsequent throws, and
4) to compare our proposed method based on *residual physics* to other baseline alternatives.

Evaluation Metrics: Evaluation metrics are defined in the following points: 1) grasping success: the percent rate, which an object remains in the gripper after executing the grasping primitive (by measuring distance between fingertips), and 2) throwing success: the percent rate, which a thrown object lands in the intended target box (tracked by an overhead camera).

A. Experimental Setup

We evaluate each policy on its ability to grasp and throw various objects into 12 boxes located outside a UR5 robot arm's maximum reach range (as shown in Fig. 1). Specifically, the task is to pick objects from a cluttered bin and stow them uniformly into the boxes such that all boxes have the same number of objects, regardless of object type. Since boxes are *outside* the robot's reach range, throwing is necessary to succeed in the task. Each box is 20 cm tall with a 25×15 cm opening. The middle of the top opening of each box is used as the input target landing position p to the formulation $f(I, p)$.

1) Simulation Setup: The simulation environment (shown in Fig. 5) is built using PyBullet [6]. We use eight different objects: four seen during training and four unseen for testing. Training objects are chosen in order of increasing difficulty: 4 cm-diameter ball, $4 \times 4 \times 4$ cm cube, 3 cm-diameter 16 cm-long rod, and a 16 cm-long hammer (union of 2 cm-diameter 12 cm-long rod with $10 \times 4 \times 2.5$ cm block). Throwing difficulty is determined by how much an object's projectile trajectory changes depending on its initial grasp and CoM. For example, the trajectory of the ball is mostly agnostic to grasp location and orientation, while both rod (CoM in middle) and hammer objects (CoM between handle and shaft) can have drastically different projectile trajectories depending on the grasping point. Objects are illustrated in Fig. 6—their CoMs indicated with a red sphere. Multiple copies of each object (12 in total) are randomly colored and dropped into the bin during training and testing.

Although simulation provides a consistent and controlled environment for fair ablative analyses, the simulated environment

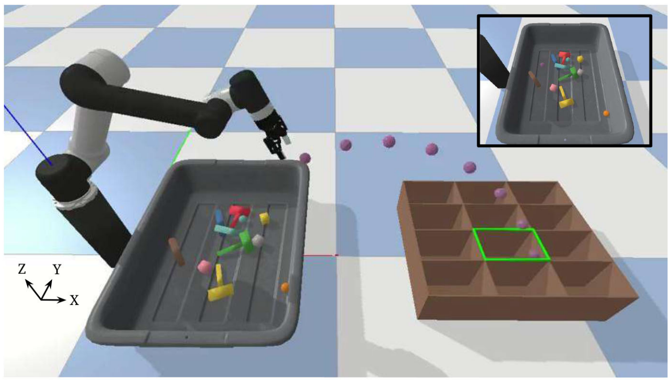

Fig. 5. *Simulation environment* in PyBullet [6]. This snapshot illustrates the aerial motion trajectory of a purple ball being thrown into the target landing box highlighted in green. The top right image depicts the view captured from the simulated RGB-D camera before the ball was grasped and thrown.

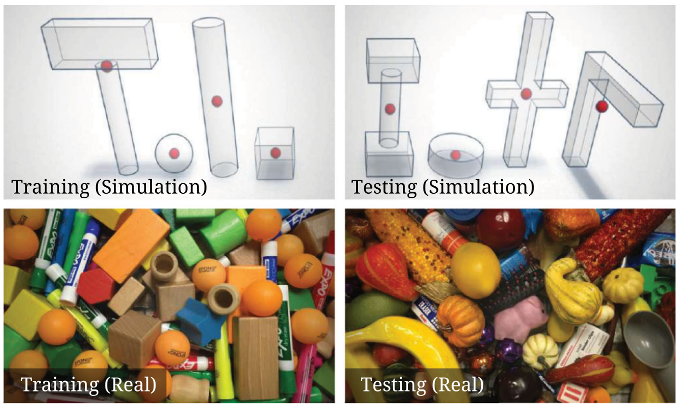

Fig. 6. *Objects* used in simulated (top) and real (bottom) experiments, split by seen objects (left), and unseen objects (right). The CoM of each simulated object is indicated with a red sphere (for illustration).

TABLE I
THROWING PERFORMANCE IN SIMULATION (MEAN %)

Method	Balls	Cubes	Rods	Hammers	Seen	Unseen
Regression	70.9	48.8	37.5	32.8	41.8	28.4
Regression-PoP	96.1	73.5	52.8	47.8	56.2	35.0
Physics-only	98.6	83.5	77.2	70.4	82.6	50.0
Residual-physics	**99.6**	**86.3**	**86.4**	**81.2**	**88.6**	**66.5**

The bold numbers indicate the best performance.

TABLE II
GRASPING PERFORMANCE IN SIMULATION (MEAN %)

Method	Balls	Cubes	Rods	Hammers	Seen	Unseen
Regression	99.4	99.2	89.0	87.8	95.6	69.4
Regression-PoP	99.2	98.0	89.8	87.0	96.4	70.6
Physics-only	99.4	99.2	87.6	85.2	96.6	64.0
Residual-physics	98.8	99.2	89.2	84.8	96.0	74.6

The bold numbers indicate the best performance.

does not model aerodynamics and only approximates frictional interactions. As a result, the performance in simulation does not necessarily reflect performance in the real world. Therefore, we also provide quantitative experiments on a real system.

2) Real-World Setup: We use a UR5 arm with an RG2 gripper to pick and throw a collection of 80+ different toy blocks, fake fruit, decorative items, and office objects (see Fig. 6). For perception data, we capture 640×480 RGB-D images using a calibrated Intel RealSense D415 statically mounted overlooking the bin of objects from the side. The camera is localized with respect to the robot base using an automatic calibration procedure from [42]. A second RealSense D415 is mounted above the boxes looking downward to track landing locations of thrown objects by measuring changes between images captured before and after executed throws.

B. Baseline Methods

1) Residual-Physics: Denotes our approach described in Section III. Since there are no comparable available algorithms that can learn joint grasping and throwing policies, we compare our approach to three baselines based on variations of the proposed method.

2) Regression: It is a variant of our approach where the throwing network is trained to directly regress the final release velocity v, instead of the residual δ. Specifically, each pixel in the output Q_t of the throwing network holds a prediction of the final release velocity $\|v_{x,y}\|$ for the throwing primitive. The physics-based controller is removed from this baseline, but in order to ensure a fair comparison, we concatenate the visual features μ with the xy-distance d between the target landing location and release position (i.e., $d = \|r_{x,y} - p_{t_{x,y}}\|$) before feeding into the grasping and throwing networks. Conceptually, this variant of our approach is forced to learn physics from scratch instead of bootstrapping on physics-based control.

3) Physics-Only: It is also a variant of our approach where the throwing network is removed and completely replaced by velocity predictions made by the physics-based controller. In other words, this variant only learns grasping and uses physics for throwing (without learning a residual).

4) Regression-Pretrained-on-Physics: It is a version of *Regression* that is pretrained on release velocity predictions $\hat{v}$ made by the physics-based controller described in Section III-C. The shorthand name for this method is *Regression-PoP*.

C. Baseline Comparisons

In simulated and real settings, we train our models via trial and error for 15 000 steps, then test each model for 1000 steps and report their average grasping and throwing success rates.

1) Simulation Results: They are reported in Tables I and II. Each column of the table represents a different set of test objects, e.g., "Hammers" is a set of n hammers, "Seen" is a mixed set of objects seen during training, "Unseen" is a mixed set of objects not seen during training.

The throwing results in Table I indicate that learning residuals (Residual-physics) on top of a physics-based controller provides the most accurate throws. Physics-only performs competitively in simulation, where the environment is void of aerodynamics and unstable contact dynamics, but falls short of performance in comparison to Residual-physics—particularly for difficult objects like rods or hammers of which the grasping offsets from

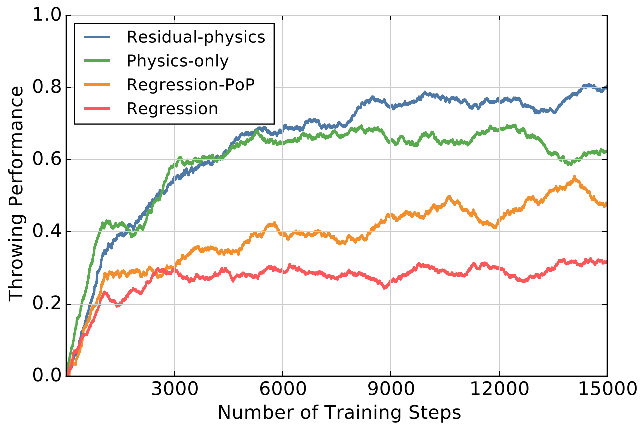

Fig. 7. Our method (Residual-physics) outperforms baseline alternatives in terms of throwing success rates in simulation on the Hammers object set.

TABLE III
GRASPING AND THROWING PERFORMANCE IN REAL (MEAN %)

	Grasping		Throwing	
Method	Seen	Unseen	Seen	Unseen
Human-baseline	–	–	–	80.1±10.8
Regression-PoP	83.4	75.6	54.2	52.0
Physics-only	85.7	76.4	61.3	58.5
Residual-physics	86.9	73.2	**84.7**	**82.3**

Bold signifies numbers are the best performing numbers (for clarity).

CoM can significantly change projectile trajectories. We also observe that regression pretrained on physics (Regression-PoP) always consistently outperforms regression alone. On the other hand, the results in Table II show that the grasping performance remains roughly the same across all methods. All policies experience moderately lower grasping and throwing success rates for unseen testing objects.

Fig. 7 plots the average throwing performance of all baseline methods over training steps on the hardest seen object set: hammers. Throwing performance is measured by throwing success rates over the last $j = 1000$ attempts. Numbers reported at earlier training steps (i.e., iteration $i < j$) in Fig. 7 are weighted by $\frac{i}{j}$. The plot shows that as soon as the performance of physics-only begins to asymptote, residual-physics starts to outperform physics-only by learning residual throwing velocities that compensate for grasping offsets from the object CoM.

2) Real-World Results: They are reported in Table III on seen and unseen object sets. The results show that residual-physics continues to provide more accurate throws than the baseline methods. Most notably, in contrast to simulation, physics-only does not perform as competitively to residual-physics in the real world. This is likely because the ballistic model used by physics-only does not account for the unmodeled and uncertain contact- and aero-dynamics in the real world. Residual-physics can compensate for them in one of two ways: either improving the model (learning good residuals), or avoiding regions of the model that are not predictable (avoiding complex grasps). This

TABLE IV
PICKING SPEED VERSUS STATE-OF-THE-ART SYSTEMS

System	Mean Picks Per Hour (MPPH)
Cartman [27]	120
Dex-Net 2.0 [23]	250
FC-GQ-CNN [31]	296
Dex-Net 4.0 [24]	312
TossingBot (w/ Placing)	432
TossingBot (w/ Throwing)	**514**

The bold numbers indicate the best performance.

allows TossingBot to maintain a throwing accuracy above 80% for both seen and unseen objects.

Interestingly, our system also seems to match or even slightly exceed the average performance of an untrained human (i.e., with no training time provided beforehand). To measure human throwing performance, we asked 15 willing participants (average height: 174.0 ± 8.3 cm, most of whom were engineers) to stand in place of the robot in the real-world setup and then grasp and throw 80 objects from the bin into the target boxes round-robin. Objects came from the collection of unseen test objects used in the robot experiments, and were kept consistent across runs. Participants were asked to pick-and-throw at whichever speed felt most comfortable (i.e., we did not compare picking speeds).

Surprisingly, the human performance was lower than we expected. The largest contributor to the poor performance was fatigue—the accuracy of throws deteriorates over time, particularly after around the 20th object regardless of picking speed. The second largest contributor to the performance was the physical height of the participant (taller performed better)—this may be due to differences in throwing distance (measured from grasp release to object landing locations, which is smaller for taller participants with longer arms) and the throwing strategies (taller participants more often preferred overhand throws to underhand ones). Other common throwing strategies included the following points:

1) largely relying tactile feedback to grasp objects in the bin to maintain visual attention on target boxes,
2) grasping objects with one hand and throwing with the other so that the throwing arm can make more repeatable movements,
3) grouping objects by weight, then correspondingly changing to different grasping and throwing strategies.

These additional strategies were interesting, but did not seem to correlate with the better performance. Also, most strategies seem designed to overcome human limitations in terms of restricted attention spans, limited viewpoints, limited motor control calibration, or fatigue, which do not hinder robotic systems.

D. Pick-and-Place Efficiency

Throwing enables our system (TossingBot) to achieve picking speeds of 514 MPPH, where 1 pick = successful grasp and accurate throw. Specifically, the system performs 608 grasps per hour (measured over two hours), and achieves 84.7% throwing accuracy, yielding 514 MPPH. In Table IV, we compare against

Algorithm 1: System Pipeline.

```
1:  Initialize robot.
2:  Initialize policy with model $f$.
3:  Initialize replay buffer.
4:  while step $i < N$ and not terminate do
5:      $I^i$ = robot.CaptureState()
6:      $p^i$ = robot.SelectTarget()
7:      $\phi_g^i, \phi_t^i = f$.Inference($I^i, p^i$)
8:      while robot.is_grasping do
9:          $f$.Train(buffer)
10:     $y^{i-1}$ = robot.CheckGraspSuccess()
11:     robot.ExecuteThrow($\phi_t^{i-1}, p^{i-1}$)    ▷ asynchronous
12:     while robot.is_throwing do
13:         $f$.Train(buffer)
14:     robot.ExecuteGrasp($\phi_g^i$)    ▷ asynchronous
15:     $\hat{p}^{i-1}$ = robot.TrackLanding()
16:     buffer.SaveData($I^{i-1}, p^{i-1}, \phi_g^{i-1}, \phi_t^{i-1}, y^{i-1}, \hat{p}^{i-1}$)
17:     $i = i + 1$
```

other state-of-the-art picking systems found in the literature: Cartman [27], Dex-Net 2.0 [23], FC-GQ-CNN [31], Dex-Net 4.0 [24], and a variant of TossingBot that places objects into a box 0.8 m away from the bin without throwing. This is not a like-for-like comparison, since throwing is only practical for certain types of objects (e.g., not eggs) and hardware, and placing is only practical for limited distance ranges. Yet, the results suggest that throwing may be useful to improve the overall MPPH in some applications.

In addition to throwing, there are the following three other aspects that enable our system's picking speeds:

1) fast algorithmic run-time speeds (220 ms for inference),
2) real-time TSDF fusion [7], [28], [30], [41] of RGB-D data, which enables us to capture and aggregate observed 3-D data of the scene simultaneously as the robot moves around within the field-of-view, and
3) online training and inference in parallel to robot actions (described in Algorithm 1).

E. Learning Stable Grasps for Throwing

We next investigate the importance of supervising grasps with the accuracy of throws. To this end, we train two variants of residual-physics: 1) grasping network supervised by accuracy of throws (i.e., grasp success = object landed on target), and 2) grasping network supervised by checking grasp width after grasping primitive (i.e., grasp success = object in gripper). We plot their grasping and throwing success rates over training steps in Fig. 8 on the hammer object set.

The results indicate that throwing performance significantly improves when grasping is supervised by the accuracy of throws. This not only suggests that the grasping policies are capable of learning to execute the subset of grasps that lead to more predictable throws, but also indirectly that throwing accuracy is strongly influenced by the quality of grasps. Interestingly, the

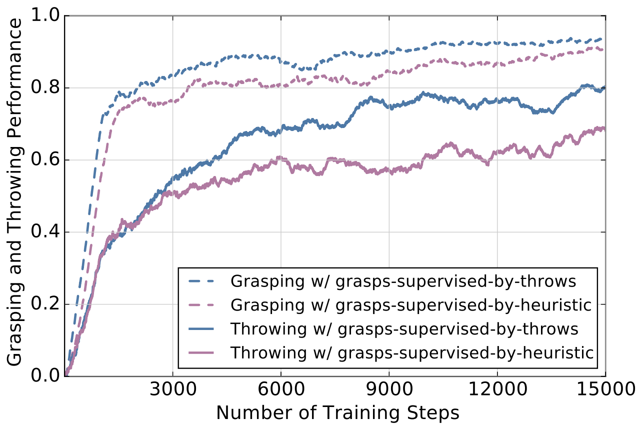

Fig. 8. Both grasping and throwing success rates of residual-physics policies in simulation improve when grasps are supervised by the accuracy of throws (blue), versus when grasps are supervised by a heuristic that checks gripper width (purple).

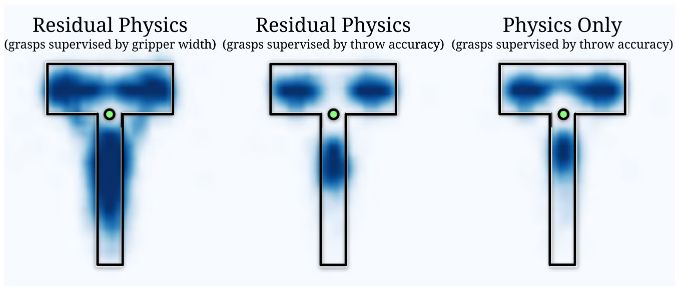

Fig. 9. Projected 2-D histograms of successful grasping positions on hammers in simulation: show that 1) leveraging accuracy of throws as supervision enables the grasping policy to learn a more restricted but stable set of grasps, while 2) learning throwing in general helps to relax this constraint.

results also show that grasping performance slightly increases when supervised by the accuracy of throws.

We also investigate the quality of learned grasps by visualizing 2-D histograms of successful grasps, mapped directly on the hammer object in Fig. 9. To create this visualization from simulation, we record each grasping position by saving the 3-D location (with respect to the hammer) of the middle point between gripper fingertips after each successful grasp. We then project the grasping positions recorded over 15 000 training steps onto a 2-D histogram, where darker regions indicate more grasps. The silhouette of the hammer is outlined in black, with a green dot indicating its CoM. We illustrate the grasp histograms of three policies: Residual-physics with grasping supervised by heuristic that checks grasp width after grasping primitive (left), residual-physics with grasping supervised by accuracy of throws (middle), and physics-only with grasping supervised by accuracy of throws (right).

The differences between left and middle histograms indicate that leveraging accurate throws as a supervisory signal encourages the grasping policy to learn a more restricted but stable and homogeneous set of grasps: slightly further from the CoM to avoid unintentional collisions between the fingers and rest of the object at the moment of release, but also further from the ends

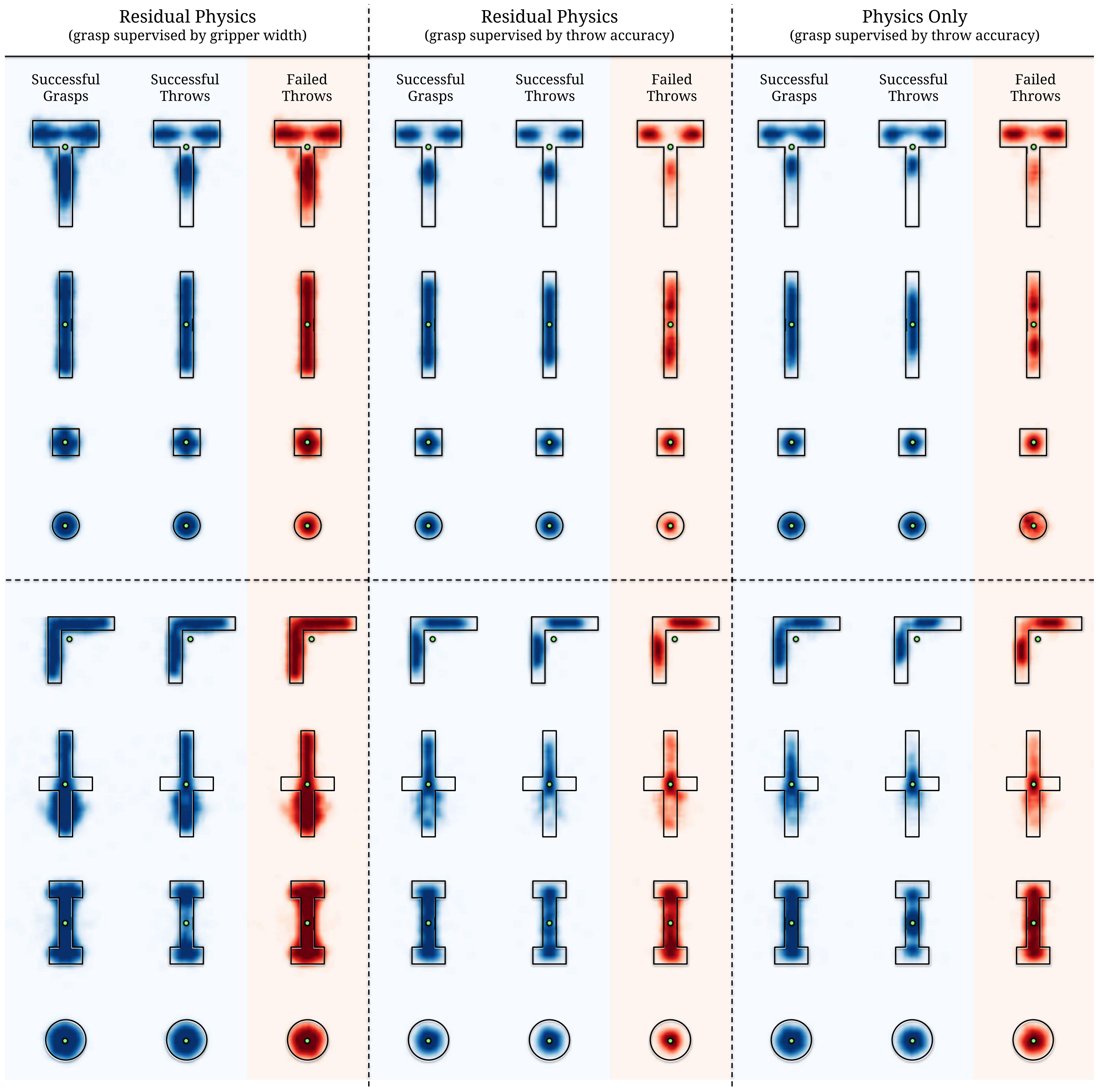

Fig. 10. *Additional grasping histograms* of all simulation objects. Histograms are generated for successful grasps, grasps that lead to successful throws, and grasps that lead to failed throws—recorded over 15 000 training steps. Darker regions indicate more grasps. The silhouette of each object is outlined in black, with a green dot indicating its CoM.

of the handle to avoid less predictable throws. The differences between middle and right histograms show that when using only ballistics for the throwing module (i.e., without learning throwing), the grasping policy seems to further optimize for grasps that are closer to the CoM. This leads to a more restricted set of grasps in contrast to Residual-physics, where the throwing can learn to compensate, respectively.

We also provide similar 2-D grasp histogram visualizations in Fig. 10 for all simulation objects. Across all policies, the histograms visualizing grasps, which lead to successful throws (columns 2, 5, 8) share large overlaps with the grasps that lead to failed throws (red columns 3, 6, 9). This suggests grasping and throwing might have been learned simultaneously, rather than one after the other—likely because the way the robot throws is conditioned on how it grasps in a nontrivial manner.

F. *Generalizing to New Target Locations*

One of the key benefits from any residual physics approach is that the physics-based part of the controller naturally generalizes

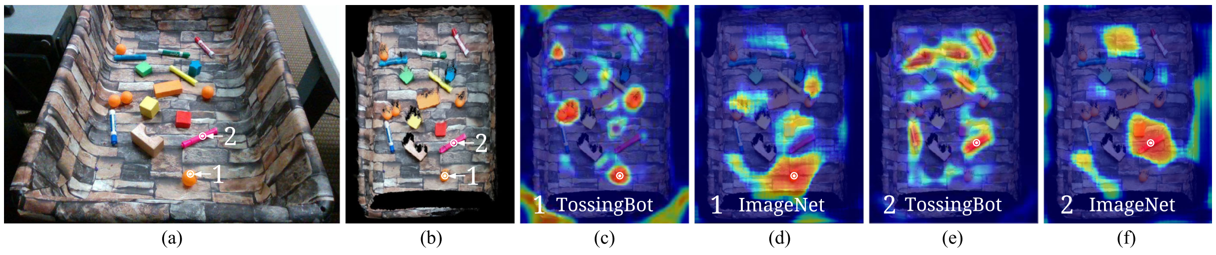

Fig. 11. *Emerging semantics from interaction.* Visualizing pixel-wise deep features μ learned by TossingBot (c) and (e) overlaid on the input heightmap image (b) generated from an RGB-D side-view (a) of a bin of objects. (c) Heatmap of pixel-wise feature distances (hotter = smaller distance) from the feature vector of a query pixel on a ping pong ball (labeled 1). Likewise, (e) Heatmap of pixel-wise feature distances from the feature vector of a query pixel on a pink marker pen (labeled 2). These visualizations show that TossingBot learns features that distinguish object categories from each other without explicit supervision (i.e., only task-level grasping and throwing). (d) and (f) For reference, the same visualization technique is used on deep features generated by a ResNet-18 pre-trained on ImageNet.

TABLE V
THROWING TO UNSEEN LOCATIONS (MEAN %)

Method	Simulation	Real
Regression-PoP	26.5	32.7
Physics-only	79.6	62.2
Residual-physics	**87.2**	**83.9**

The bold numbers indicate the best performance.

to conditions outside the collected data, for example, to new target locations. To explore how well our trained TossingBot policies generalize to new target locations, we displace the locations of the boxes in both the horizontal plane from where they were during training, such that there is no overlap between training and testing locations. For this experiment, we set in simulation 12 training boxes and 12 testing boxes; while in real settings, we set four training and four testing boxes (limited by physical setup). We record each model's throwing performance on seen objects over these new box locations across 1000 testing steps in Table V.

We observe that both in simulated and in real experiments, residual-physics significantly outperforms the regression baseline. The performance margin in this scenario illustrates how Residual-physics leverages the generalization of the ballistic equations to adapt to new target locations.

G. Deep Object Semantics Emerging From Task Training

In this section, we explore the deep features being learned by the neural network f and answer the questions: "What does TossingBot learn from grasping and throwing arbitrary objects?" and "Do they convey any meaningful structure or representation?" We do this by analyzing how similarly or differently the learned network handles different objects.

To this end, we place several training objects in the bin (well-isolated from each other for visualization purposes), capture RGB-D images to construct a heightmap I of the scene, and feed it through the network f (already trained for 15 000 steps from real experiments). The training objects include marker pens, ping pong balls, and wooden toy blocks of different shapes (see Fig. 11). We then extract the intermediate spatial feature representation of the network μ (described in Section III-A), which contains a 512-dimensional feature vector for each pixel of the heightmap I (after $4\times$ upsampling to the same resolution). We then extract the feature vector from a query pixel belonging to one of the objects in the bin (ping pong ball in this case), and visualize its distance to all other pixel-wise features as a heatmap in Fig. 11(a) (where hotter regions indicate smaller distances), overlaid on the original input heightmap. More specifically, we rank each pixel based on its ℓ_2 feature distance to the query pixel feature, then colorize it based on its rank (i.e., higher rank = closer feature distance = hotter color).

The procedure creates a form of similarity map between pixels. Interestingly, when choosing the pixel from a ping-pong ball, the visualization immediately localizes all other ping-pong balls in the scene—presumably because they share similar deep features. It is also interesting to note that the orange wooden block, despite sharing a similar color, does not get picked up by the query. Similarly, Fig. 11(b) illustrates the feature distances between a query pixel on a pink marker pen to all other pixels of the scene. The visualization immediately localizes all other marker pens, which share similar shape and mass, but do not necessarily share color textures.

These interesting results suggest that the deep network is learning to bias the features (i.e., learning a prior) based on the objects' shapes more so than their visual textures or color. The network likely learns that geometric cues are more useful for learning grasping and throwing policies—i.e., provides more information related to grasping interactions and projectile behaviors. In addition to shape, one could also argue that the learned deep features reflect the second-order (beyond visual or geometric) physical attributes of objects, which influence their aerial behaviors when thrown. This perspective is also plausible, since the throwing policies are effectively learning to compensate for these physical attributes, respectively. For comparison, these visualizations generated by features from TossingBot are more informative in this setting than those generated using deep features from a 18-layer ResNet pretrained on ImageNet (also shown in Fig. 11).

These emerging features were learned implicitly from scratch without any explicit supervision beyond task-level grasping and throwing. Yet, they seem to be sufficient for enabling the system to distinguish between ping-pong balls and markers. As such, this experiment speaks out to a broader concept related to machine vision: how should robots learn the semantics of the visual world? From the perspective of classic computer vision, semantics are often predefined using human-fabricated image datasets and manually constructed class categories (i.e., this is a "hammer," and this is a "pen"). However, our experiment suggests that it is possible to implicitly learn such object-level semantics from physical interactions alone (as long as they matter for the task at hand). The more complex these interactions, the higher the resolution of the semantics. Toward more generally intelligent robots—perhaps it is sufficient for them to develop their own notion of semantics through interaction [36], without human guidance.

VII. Discussion and Future Work

This article presents a framework for jointly learning grasping and throwing policies that enable TossingBot to pick-and-throw arbitrary objects from an unstructured bin into boxes located outside its maximum reach range at 500+ MPPH. We show that a key is the use of *residual physics*, a hybrid controller that leverages deep learning to predict residuals on top of control parameters estimated with physics. The combination enables the data-driven predictions to focus on learning the aspects of dynamics that are difficult to model analytically. Our experiments in both simulation and real settings show that the system: 1) learns to improve grasps for throwing through joint training from trial and error, and 2) performs significantly better with residual physics than comparable alternatives.

The proposed system is a prototype with several limitations that suggest directions for future work. First, it assumes that objects are rigid and robust enough to withstand forces encountered when thrown—further work is required to train networks to predict motions that account for fragile, articulated, or deformable objects. Second, it infers object-centric properties and dynamics only from visual data (an RGB-D image of the bin)—exploring additional sensing modalities such as force-torque or tactile may enable the system to better react to new objects and better adapt its throwing velocities. Third, it is only able to infer the parameters needed to get an object to land in a target location—it would be interesting to explore how to achieve more fine-grained control of the pose (including orientation) of an object in flight, potentially to reach a target landing pose while avoiding or leveraging external obstacles. Finally, we have so far demonstrated the benefits of residual physics only in the context of throwing—investigating how the idea generalizes to other tasks is a promising direction for future research.

Acknowledgment

The authors would like to thank R. Hickman for managerial support, I. Krasin, and S. Welker for technical discussions, B. Hurd, J. Salazar, and S. Snyder for hardware support, C. Richards and J. Freidenfelds for feedback on writing, E. Coumans for advice on PyBullet, L. Graesser for video narration, and R. Hickman for photography. The authors are also grateful for hardware and financial support from Google, Amazon, Intel, NVIDIA, ABB Robotics, and Mathworks.

References

[1] P. Abbeel, M. Quigley, and A. Y. Ng, "Using inaccurate models in reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2006, pp. 1–8.
[2] E. W. Aboaf, C. G. Atkeson, and D. J. Reinkensmeyer, "Task-level robot learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1988, pp. 1309–1310.
[3] A. Ajay *et al.*, "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 3066–3073.
[4] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," in *Proc. IEEE Trans. Pattern Anal. Mach. Intell.*, 2017, pp. 2481–2495.
[5] N. Chaval-Dafle *et al.*, "Extrinsic dexterity: In-hand manipulation with external forces," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 1578–1585.
[6] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," pp. 2016–2018. [Online]. Available: http://pybullet.org
[7] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. Special Interest Group Comput. Graph. Interactive Techn.*, 1996, pp. 303–312.
[8] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Robot. Inst., Carnegie Mellon University, Pittsburgh, PA, USA, 2010.
[9] N. Fazeli, S. Zapolsky, E. Drumwright, and A. Rodriguez, "Learning data-efficient rigid-body contact models: Case study of planar impact," in *Proc. Conf. Robot Learn.*, 2017, pp. 388–397.
[10] Y. Gai, Y. Kobayashi, Y. Hoshino, and T. Emaru, "Motion control of a ball throwing robot with a flexible robotic arm," in *Proc. World Acad. Sci., Eng. Technol.*, no. 79, p. 622, 2013.
[11] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, 2351–2358.
[12] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1440–1448.
[13] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.
[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[15] J. C. G. Higuera, D. Meger, and G. Dudek, "Adapting learned robotics behaviours through policy adjustment," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 5837–5843.
[16] J.-S. Hu, M.-C. Chien, Y.-J. Chang, S.-H. Su, and C.-Y. Kai, "A ball-throwing robot with visual feedback," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 2511–2512.
[17] Y. Jiang, J. Sun, and C. K. Liu, "Data-augmented contact model for rigid body simulation," 2018, *arXiv:1803.04019*.
[18] T. Johannink *et al.*, "Residual reinforcement learning for robot control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6023–6029.
[19] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018.
[20] J. Kober, E. Öztop, and J. Peters, "Reinforcement learning to adjust robot movements to new situations," in *Proc. Int. Joint Conf. Artif. Intell.*, 2011, pp. 2650–2655.
[21] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440.
[22] K. M. Lynch and M. T. Mason, "Dynamic nonprehensile manipulation: Controllability, planning, and experiments," *Int. J. Robot. Res.*, vol. 18, pp. 64–92, 1999.
[23] J. Mahler *et al.*, "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics," *Robot. Sci. Syst.*, 2017.
[24] J. Mahler *et al.*, "Learning ambidextrous robot grasping policies," *Sci. Robot.*, vol. 4, no. 26, 2019.
[25] M. T. Mason and K. M. Lynch, "Dynamic manipulation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 1993, pp. 152–159.
[26] W. Mori, J. Ueda, and T. Ogasawara, "1-dof dynamic pitching robot that independently controls velocity, angular velocity, and direction of a ball: Contact models and motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 1655–1661.
[27] D. Morrison *et al.*, "Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 7757–7764.

[28] R. A. Newcombe *et al.*, "Kinectfusion: Real-time dense surface mapping and tracking," in *Proc. IEEE Int. Symp. Mixed Augmented Reality*, 2011, pp. 127–136.

[29] P. Pastor *et al.*, "Learning task error models for manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 2612–2618.

[30] S. S. Sajjan *et al.*, "Cleargrasp: 3D shape estimation of transparent objects for manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020.

[31] V. Satish, J. Mahler, and K. Goldberg, "On-policy dataset synthesis for learning deep robot grasping policies based on fully-convolutional grasp quality neural networks," *IEEE Robot. Autom. Lett. (RA-L)*, vol. 4, no. 2, pp. 1357–1364, Apr. 2019.

[32] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Representations*, 2016.

[33] T. Senoo, A. Namiki, and M. Ishikawa, "High-speed throwing motion based on kinetic chain approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2008, pp. 3206–3211.

[34] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," 2018, *arXiv:1812.06298*.

[35] O. Taylor and A. Rodriguez, "Optimal shape and motion planning for dynamic planar manipulation," *Auton. Robots*, vol. 43, no. 2, pp. 327–344, 2019.

[36] Z. Xu, J. Wu, A. Zeng, J. B. Tenenbaum, and S. Song, "Densephysnet: Learning dense physical object representations via multi-step dynamic interactions," *Robot.: Sci. Syst.*, 2019.

[37] L. Yen-Chen, S. Zeng, Andy Song, P. Isola, and T.-Y. Lin, "Learning to see before learning to act: Visual pre-training for manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, to be published.

[38] K. Zakka, A. Zeng, J. Lee, and S. Song, "Form2fit: Learning shape priors for generalizable assembly from disassembly," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020.

[39] A. Zeng, "Learning visual affordances for robotic manipulation," Ph.D. dissertation, Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, 2019.

[40] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *Robot.: Sci. Syst.*, 2019.

[41] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3dmatch: Learning local geometric descriptors from rgb-d reconstructions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 199–208.

[42] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 4238–4245.

[43] A. Zeng *et al.*, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 3750–3757.

Andy Zeng received the bachelor's in computer science and mathematics with University of California, Berkeley, Berkeley, CA, USA, in 2015, and the Ph.D. degree in computer science with Princeton University, Princeton, NJ, USA, in 2019.

He is currently a Research Scientist in Robotics with Google AI, Mountain View, CA, USA. His research interests lie in vision for robotics developing learning algorithms that enable machines to intelligently interact with the physical world and improve themselves over time.

Dr. Zeng was the recipient of the Best System Paper Award at RSS in 2019, Best Manipulation System Paper Award from Amazon in 2018, and finalist for Best Cognitive Robotics Paper Award at IROS in 2018. His research has been recognized through the Gordon Y.S. Wu Fellowship in Engineering and Wu Prize in 2016, NVIDIA Fellowship in 2018, and Princeton SEAS Award for Excellence in 2018.

Shuran Song received the B.Eng. degree in computer engineering from Hong Kong University of Science and Technology, Hong Kong, in 2013 and the Ph.D. degree in computer science from Princeton University, Princeton, NJ, USA, in 2018.

She is currently an Assistant Professor with the Department of Computer Science, Columbia University, New York, NY, USA. During her Ph.D., she spent time working at Microsoft Research, Redmond, WA, USA, and Google Daydream, Mountain View, CA, USA. Her research interests lie at the intersection of computer vision, robotics, and computer graphics.

Dr. Song was the recipient of the RSS Best System Paper Award in 2019, Facebook Fellowship in 2014, Siebel Scholar in 2016, Wallace Fellowship in 2017, and Princeton SEAS Award for Excellence in 2017.

Johnny Lee received the Ph.D. degree in human–computer interaction from Carnegie Mellon University, Pittsburgh, PA, USA, in 2008.

He is an Engineering Director in Robotics with Google, Mountain View, CA, USA, which is a division focused on applying machine learning to the field of robotics. His areas of interest include machine learning, computer vision, sensors, robotics, and human–computer interaction. Previously, he led the development of hardware and software technologies for AR/VR such as real-time motion tracking and environment mapping, and early stage RD efforts in Tango/Daydream. Previously, he helped Google X explore new projects as a Rapid Evaluator, and was a core algorithms contributor to the original Xbox Kinect. His YouTube videos demonstrating Wii remote hacks have surpassed over 15 million views and became one of the most popular TED talk videos. He has been recognized in MIT Technology Reviews TR35.

Alberto Rodriguez received the M.Sc. degree in mathematics in 2005 and M.Sc. degree in telecommunication engineering in 2006 from the Universitat Politecnica de Catalunya, Barcelona, Spain, and the Ph.D. degree from the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, in 2013.

He is currently an Associate Professor with the Mechanical Engineering Department, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. He leads the Manipulation and Mechanisms Lab, MIT (MCube) researching autonomous dexterous manipulation, robot automation, and endeffector design.

Dr. Rodriguez was the recipient of Best Paper Awards at conferences RSS11, ICRA13, RSS18, IROS18, and RSS19, the 2018 Best Manipulation System Paper Award from Amazon, and has been finalist for best paper awards at conferences IROS16 and IROS18. He has lead Team MIT-Princeton in the Amazon Robotics Challenge between 2015 and 2017, and Faculty Research Awards from Amazon in 2018, 2019, and 2020, and from Google in 2020. He is also the recipient of the 2020 IEEE Early Academic Career Award in Robotics and Automation.

Thomas Funkhouser received the Ph.D. degree in computer science from University of California, Berkeley, Berkeley, CA, USA, in 1993.

He is currently a Senior Research Scientist with Google, New York, NY, USA, and the David M. Siegel Professor of Computer Science, Emeritus with the Princeton University (Princeton), Princeton, NJ, USA. He was a member of the technical staff at Bell Labs, New Providence, NJ, USA, for 4 years, and then was with the Princeton for 20 years. His research interests include computer graphics, computer vision, and robotics.