TossingBot: Learning to Throw Arbitrary Objects with Residual Physics

Andy Zeng[1,2], Shuran Song[1,2,3], Johnny Lee[2], Alberto Rodriguez[4], Thomas Funkhouser[1,2]
[1]Princeton University [2]Google [3]Columbia University [4]Massachusetts Institute of Technology
http://tossingbot.cs.princeton.edu

***Abstract*—We investigate whether a robot arm can learn to pick and throw arbitrary 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 reliable pre-throw conditions (*e.g.* grasp of the object) to handling varying object-centric properties (*e.g.* mass distribution, friction, shape) and dynamics (*e.g.* 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 (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. Videos are available at http://tossingbot.cs.princeton.edu**

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 pre-throw 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 (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] – hence prior studies are often confined to assuming homogeneous pre-throw 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 pre-throw conditions (via grasping) and adapt its throws to account for varying properties and dynamics of arbitrary objects.

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.

In this work, we present ***TossingBot***, an end-to-end formulation that uses trial and error to learn how to plan control 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 two key aspects to our system:

- **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 [33], 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.

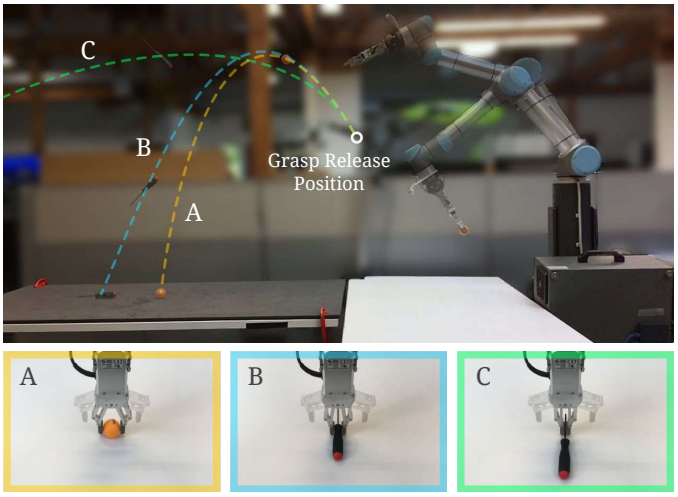

Fig. 2. **Projectile trajectories** of a thrown ping pong ball (a), screwdriver grasped and thrown by its handle (b), and the same screwdriver grasped and thrown by its shaft (c). The difference between (a) and (b) is largely due to aerodynamics, while the difference between (b) and (c) is largely due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

- **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 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. Qualitative results (videos) are available at http://tossingbot.cs.princeton.edu

II. Related Work

Analytical models for throwing. Many previous systems built for throwing [22, 10, 23, 29, 31] 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 in Mason and Lynch [22], 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 [29]), 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).

Learning models for throwing. More recently, learning-based systems for robotic throwing [2, 14, 18, 11] have also been proposed, which ignore low-level dynamics and directly optimize for task-level success 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 pre-throw 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 pre-throw conditions (via grasping) while learning throwing control parameters that compensate for varying object properties and dynamics. The system learns from scratch through self-supervised trial and error, and resets it own training so that human intervention is kept at a minimum.

Learning residual models and policies. Our approach to data-efficient learning, *Residual Physics*, falls under a broader category of hybrid controllers [1, 26] 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. 3d). In contrast to prior work on learning residuals on predictions of future states for model-based control [3, 17] or data-augmented models [9, 15], 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 [16, 30] in block-assembly and object manipulation tasks.

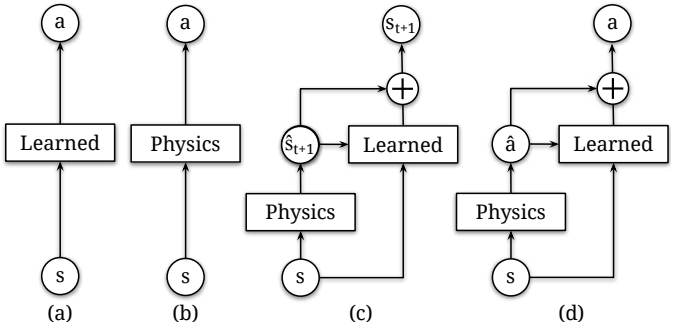

Fig. 3. Learning residual models and policies: (a) analytical solutions that determine action a from state s; (b) data-driven policies that learn the direct mapping from states to actions; (c) hybrid models that combine analytical models with learning to predict future states s_{t+1}; (d) hybrid policies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

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 3D 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 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 and 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 subsections provide an overview of these three modules, while the next 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 3D point cloud, and orthographically back-project upwards 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) 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 0.9×0.7m 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 5×5mm vertical column of 3D space in the robot's workspace. Using its height-from-bottom value, each pixel thereby corresponds to a unique 3D location in the robot's workspace. The input I is fed into the perception network, a 7-layer fully convolutional residual network [4, 13, 19] (interleaved with 2 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.

Grasping primitive. The grasping primitive takes as input parameters $\phi_g = (x, \theta)$ and executes a top-down parallel-jaw grasp centered at a 3D 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 3D position of the middle point between the gripper fingertips meets x, at which point the gripper closes, and lifts upwards 10cm. This primitive is open-loop, with robot arm motion planning executed using a stable, collision-free IK solver [8].

Grasping network. The grasping network is a 7-layer fully convolutional residual network [4, 13, 19] (interleaved with 2 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 3D location of $i \in I$ with the gripper oriented horizontally with respect to the heightmap I.

As in [34, 33], we account for different grasping angles by rotating the input heightmap by 16 orientations (multiples of 22.5°) before feeding into the network. 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 3D 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 [33, 34]. 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).

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 inwards while grasping onto an object, then uncurls outward at high speed, releasing the object at the desired position and velocity.

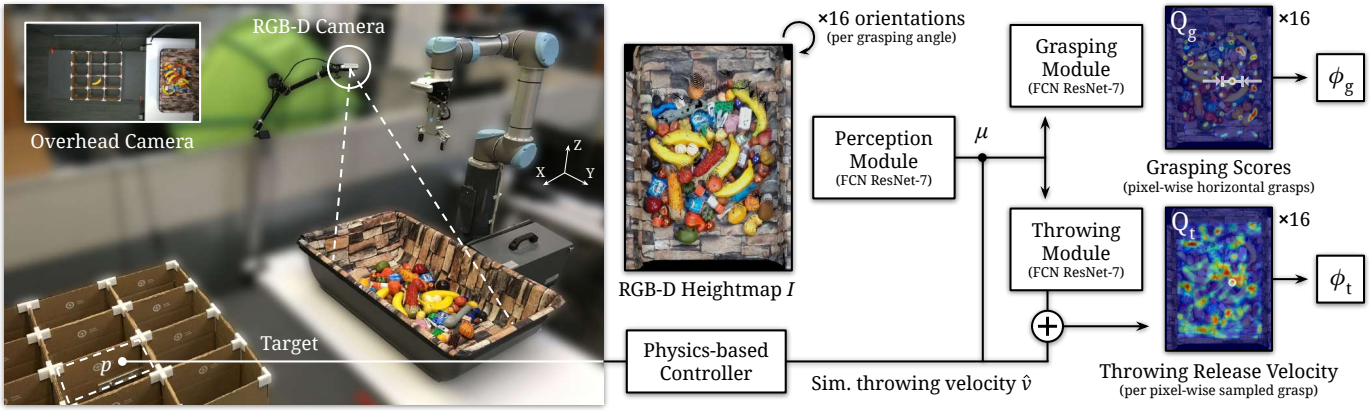

Fig. 4. **Overview.** An RGB-D heightmap of the scene is fed into a perception module to compute spatial features μ. In parallel, target location p is fed into a physics-based controller to provide an initial estimate of throwing release velocity $\hat{v}$, which is concatenated with μ then fed into 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.

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).

Planning the release position. In most real-world settings, only a handful of release positions are accessible by the robot for throwing. So for simplicity in our system, we directly derive 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 $\sqrt{r_x^2 + r_y^2}$ is at a fixed distance 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.

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° upwards 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. As we show in Sec. VIII-C of the appendix, changing $\|v_{x,y}\|$ and r 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 Sec. VI show that this approach, a.k.a. *Residual Physics*, yields significant improvements in both accuracy and generalization of throwing arbitrary objects compared to baseline alternatives: *e.g.* using only the physics-based controller (Fig. 3a), or directly training f to regress $\|v_{x,y}\|$ (Fig. 3b).

Physics-based controller. The physics-based controller uses the standard equations of linear projectile motion, by assuming a grasp on the center of mass 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 aerial trajectory of the projectile moves along a ballistic path affected only by gravity, which imparts a downward acceleration $a_z = -9.8\text{m/s}^2$.

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 center of mass, 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.

Estimating residual release velocity. 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 7-layer fully convolutional residual network [13] interleaved with 2 layers of spatial bilinear $2\times$ 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:

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

and $\mathcal{L}_t$ is the Huber loss from its regression of δ_i for throwing:

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

where y_i is the binary ground truth grasp success label and $\bar{\delta}_i$ is the ground truth residual label. We use an 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. More training details in Sec. VIII-A of the appendix.

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 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 Sec. VI-E, supervising grasps by the accuracy of throws eventually leads to more stable grasps and better overall throwing performance, since the grasping policy learns to favor grasps that lead to successful throws. After each throw, we measure the object's actual landing location $\bar{p}$ using a calibrated overhead RGB-D camera to detect changes in the landing zone before and after the throw. Regardless of where the object lands, its actual landing location $\bar{p}$ 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 $\bar{\delta}_i = \|v_{x,y}\| - \|\hat{v}_{x,y}\|_{\bar{p}}$.

In experiments in Sec. VI, we train our models by self-supervision with the same procedure: n objects are randomly dropped into the 0.9×0.7m 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 titled at a $15°$ angle adjacent to the bin. 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 are 1) grasping success: the % rate which an object remains in the gripper after executing the grasping primitive (by measuring distance between fingertips), and 2) throwing success: the % 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

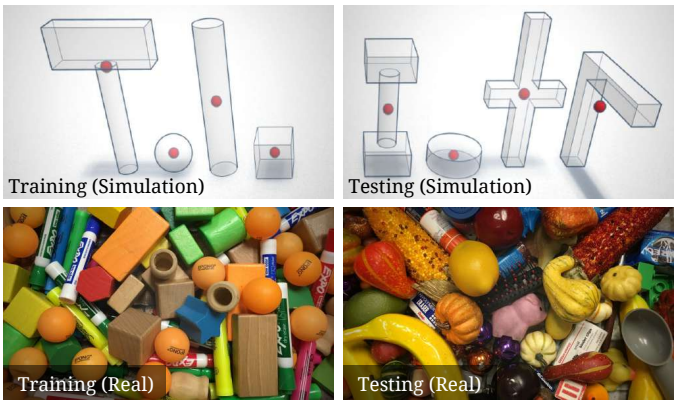

Fig. 5. **Objects** used in simulated (top) and real (bottom) experiments, split by training objects (left) and unseen testing objects (right). The center of mass of each simulated object is indicated with a red sphere (for illustration).

number of objects, regardless of object type. Since boxes are located *outside* the robot's reach range, throwing is necessary to succeed in the task. Each box is 20cm tall with a 25×15cm opening. The middle of the top opening of each box is used as the input target landing position p to the formulation $f(I, p)$.

Simulation setup. The simulation environment is built using PyBullet [6] (appendix Fig. 9). We use 8 different objects: 4 seen during training and 4 unseen for testing. Training objects are chosen in order of increasing difficulty: 4cm-diameter ball, $4 \times 4 \times 4$cm cube, 3cm-diameter 16cm-long rod, and a 16cm-long hammer (union of 2cm-diameter 12cm-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 center of mass (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. 5 – 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 does not account for aerodynamics, and as a result, performance in simulation does not necessarily reflect the performance in the real world. Therefore we also provide quantitative experiments on a real system.

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. 5). 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 [33]. A second RealSense D415 is mounted above the boxes looking downwards to track landing locations of thrown objects by measuring changes between images captured before and after executed throws.

B. Baseline Methods

Residual-physics denotes our approach described in Sec. 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:

Regression 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.

Physics-only 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).

Regression-pretrained-on-physics is a version of ***Regression*** that is pre-trained on release velocity predictions $\hat{v}$ made by the physics-based controller described in Sec. III-C. The shorthand name for this method is ***Regression-PoP***.

We also provide a ***Human-baseline***, which measures the performance of 15 untrained participants performing the task. More details in Sec. VIII-D of the appendix.

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 1,000 steps and report their average grasping and throwing success rates.

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

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**

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

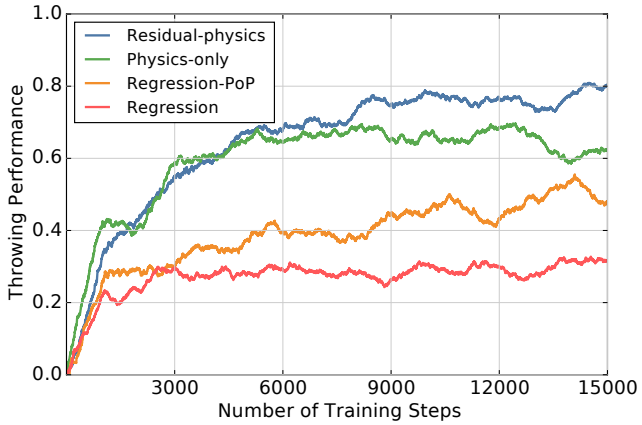

Fig. 6. Our method (Residual-physics) outperforms baseline alternatives in terms of throwing success rates in simulation on the Hammers object set.

Fig. 6 plots the average throwing performance of all baseline methods over training steps on the hardest object set: hammers. Throwing performance is measured by throwing success rates over the last $j = 1{,}000$ attempts. Numbers reported at earlier training steps (*i.e.*, iteration $i < j$) in Fig. 6 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.

Real-world results 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 unmodelled 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, or avoiding regions of the model that are not predictable. This allows to maintain a throwing accuracy above 80% for both seen and unseen objects. Interestingly, our system also seems to exceed the average performance of an untrained human.

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**

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

System	Mean Picks Per Hour (MPPH)
Cartman [24]	120
Dex-Net 2.0 [20]	250
FC-GQ-CNN [27]	296
Dex-Net 4.0 [21]	312
TossingBot (w/ Placing)	432
TossingBot (w/ Throwing)	**514**

D. Pick-and-Place Efficiency

Throwing enables our system (TossingBot) to achieve picking speeds of 514 mean picks per hour (MPPH), where 1 pick = successful grasp and accurate throw. Specifically, the system performs 608 grasps per hour, and achieves 84.7% throwing accuracy, yielding 514 MPPH. In Table IV, we compare against other state-of-the-art picking systems found in literature: Cartman [24], Dex-Net 2.0 [20], FC-GQ-CNN [27], Dex-Net 4.0 [21], and a variant of TossingBot that places objects into a box 0.8m 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 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.

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. 7 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 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 2D histograms of successful grasps, mapped directly on the hammer object in Fig. 8. To create this visualization from simulation, we record each grasping position by saving the 3D location (with respect to the hammer) of the middle point

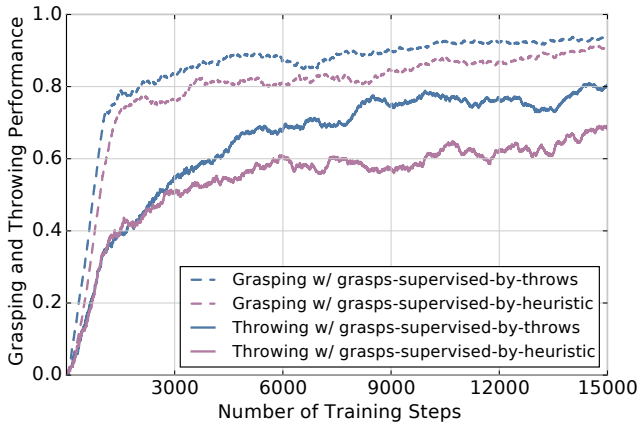

Fig. 7. Both grasping and throwing success rates of Residual-physics policies improve when grasps are supervised by the accuracy of throws (blue), versus when grasps are supervised by a heuristic that checks grasp width (purple).

between gripper fingertips after each successful grasp. We then project the grasping positions recorded over 15,000 training steps onto a 2D 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 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 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 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. More examples and analysis in Sec. VIII-E of the appendix.

F. Generalizing to New Target Locations

To explore how the trained 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 4 training and 4 testing boxes (limited by physical setup). We record each model's throwing performance on seen objects over these new box locations across 1,000 testing steps in Table V.

We observe that in both simulated and 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.

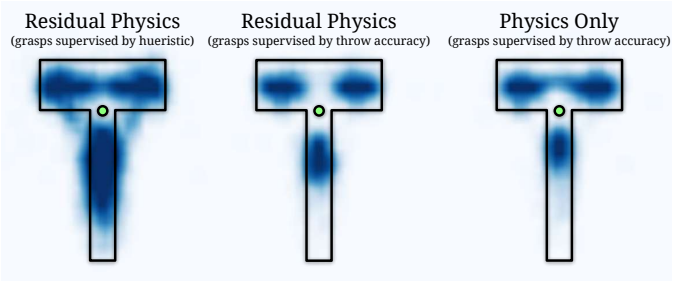

Fig. 8. Projected 2D histograms of successful grasping positions on hammers in simulation: show that 1) leveraging accuracy of throws as supervision enables the grasping policy to learn a more restricted but stable set of grasps, while 2) learning throwing in general helps to relax this constraint.

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**

VII. DISCUSSION AND FUTURE WORK

This paper 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 robust enough to withstand forces encountered when thrown – further work is required to train networks to predict motions that account for fragile 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. 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.

ACKNOWLEDGMENTS

We would like to thank the team members of Robotics at Google for insightful discussions and technical support. We are also grateful for hardware and financial support from Amazon, Intel, NVIDIA, ABB Robotics, and Mathworks.

REFERENCES

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[2] Eric W Aboaf, Christopher G Atkeson, and David J Reinkensmeyer. Task-level robot learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 1988.

[3] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2017.

[5] Nikhil Chaval-Dafle, Alberto Rodriguez, R. Paolini, B. Tang, S. S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge. Extrinsic dexterity: In-hand manipulation with external forces. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[6] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2018.

[7] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Special Interest Group on Computer GRAPHics and Interactive Techniques (SIGGRAPH)*, 1996.

[8] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute.

[9] Nima Fazeli, Samuel Zapolsky, Evan Drumwright, and Alberto Rodriguez. Learning data-efficient rigid-body contact models: Case study of planar impact. *Conference on Robot Learning (CoRL)*, 2017.

[10] Yizhi Gai, Yukinori Kobayashi, Yohei Hoshino, and Takanori Emaru. Motion control of a ball throwing robot with a flexible robotic arm. *World Academy of Science, Engineering and Technology (WASET)*, 2013.

[11] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[12] Ross Girshick. Fast r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] Jwu-Sheng Hu, Ming-Chih Chien, Yung-Jung Chang, Shyh-Haur Su, and Chen-Yu Kai. A ball-throwing robot with visual feedback. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[15] Yifeng Jiang, Jiazheng Sun, and C Karen Liu. Data-augmented contact model for rigid body simulation. *arXiv*, 2018.

[16] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. *arXiv*, 2018.

[17] Alina Kloss, Stefan Schaal, and Jeannette Bohg. Combining learned and analytical models for predicting action effects. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[18] Jens Kober, Erhan Öztop, and Jan Peters. Reinforcement learning to adjust robot movements to new situations. *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2011.

[19] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[20] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *Robotics: Science and Systems (RSS)*, 2017.

[21] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 2019.

[22] Matthew T Mason and Kevin M Lynch. Dynamic manipulation. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1993.

[23] Wataru Mori, Jun Ueda, and Tsukasa Ogasawara. 1-dof dynamic pitching robot that independently controls velocity, angular velocity, and direction of a ball: Contact models and motion planning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[24] Douglas Morrison, Adam W Tow, M McTaggart, R Smith, N Kelly-Boxall, S Wade-McCue, J Erskine, R Grinover, A Gurman, T Hunn, et al. Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[25] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2011.

[26] Peter Pastor, Mrinal Kalakrishnan, Jonathan Binney, Jonathan Kelly, Ludovic Righetti, Gaurav Sukhatme, and Stefan Schaal. Learning task error models for manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[27] Vishal Satish, Jeffrey Mahler, and Ken Goldberg. On-policy dataset synthesis for learning deep robot grasping policies based on fully-convolutional grasp quality neural networks. *IEEE Robotics and Automation Letters (RA-L)*.

[28] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *International Conference on Learning Representations*, 2016.

[29] Taku Senoo, Akio Namiki, and Masatoshi Ishikawa. High-speed throwing motion based on kinetic chain approach. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[30] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv*, 2018.

[31] Orion Taylor and Alberto Rodriguez. Optimal shape and motion planning for dynamic planar manipulation. *Autonomous Robots*, 43(2): 327–344, 2019.

[32] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[33] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[34] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.