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.* initial pose of object in manipulator) 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 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**

Fig. 1. **TossingBot** learns to grasp arbitrary objects from an unstructured bin and 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 predictions of throwing release velocities.

I. Introduction

Throwing is an excellent means of exploiting dynamics to increase the capabilities of a manipulator. In the case of pick-and-place for example, throwing enables a robot arm to rapidly place objects 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 pose of object in manipulator) 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 forces 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 on the handle (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 analytically model or measure [18] – hence prior studies are often confined to assuming homogeneous pre-throw conditions (*e.g.* object fixtured in gripper, 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 actively acquire its own pre-throw conditions (via grasping) and adapt its throws to account for varying properties and dynamics of arbitrary objects.

In this work, we propose ***TossingBot***, an end-to-end formulation that uses trial and error to learn how to predict 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:

- We jointly learn grasping and throwing policies with a deep neural network that maps from visual observations (of objects in a bin) to control parameters: the likelihood of grasping success for a dense pixel-wise sampling of end effector orientations and locations [27], 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 our dense predictions). As a result, our end-to-end policies learn to execute stable grasps that lead to predictable throws, as well as throwing velocities that can account for the variations in object-centric properties and dynamics that can be inferred from visual information.
- To make accurate predictions of throwing release velocities,

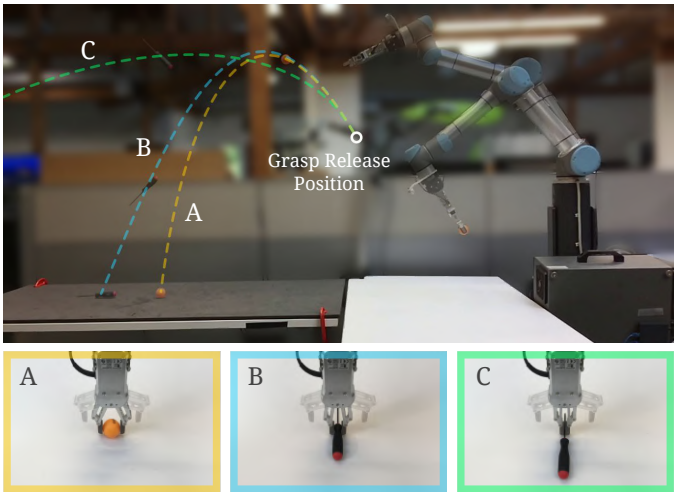


Fig. 2. **Projectile trajectories** of a thrown ping pong ball (a), screwdriver grasped and thrown by its handle (b), and the same screwdriver grasped and thrown by its shaft (c). The difference between (a) and (b) is largely due to aerodynamics, while the difference between (b) and (c) is largely due 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.

our throwing module learns a residual δ on top of an initial estimate $\hat{v}$ from a physics-based controller, and 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 reliably grasp and throw arbitrary objects 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 our 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 [7, 19, 24] rely on handcrafting or approximating dynamics based on mechanical analysis, 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 [18], accurately modeling these dynamics is challenging since 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 the objects and manipulators. As a result, these ad hoc systems often observe limited throwing accuracy (*e.g.* 40% success rate in [24]), 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 to compensate for the dynamics that are not explicitly accounted for in contact/ballistic models, and train our policies online via trial and error so that they can adapt to new situations (*e.g.* new object and manipulator dynamics) on the fly.

Learning models for throwing. More recently, learning-based systems for robotic throwing [1, 11, 14, 8] 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 shown to fare better than those which rely only on analytical models, but continue to be characterized by two primary drawbacks: 1) limited generalization to new object types (beyond balls or darts), and 2) the assumption that pre-throw conditions across all throws are kept the same (*e.g.* human operators are required to manually reset objects and manipulators to match the initial pre-throw state), which makes training from trial and error costly. Both drawbacks prevent the practical use of these throwing systems 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 proposed 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.

III. Method Overview

TossingBot consist 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. 3). The learning objective is to optimize our predictions of parameters ϕ_g and ϕ_t such that at each time-step, 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.

The network f consists of three parts: 1) a perception module that accepts visual input I and outputs a spatial feature representation μ, which is then 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 using an additional overhead camera to track ground truth landing positions of thrown

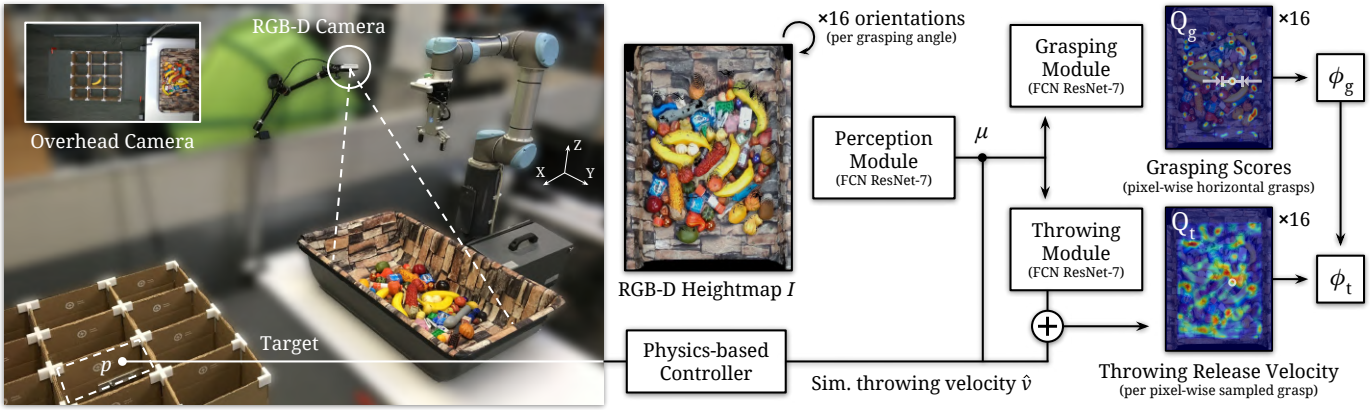


Fig. 3. **Overview.** An RGB-D heightmap of the scene is fed into a perception module to compute spatial features μ. In parallel, target location p is fed into a physics-based controller to provide an initial estimate of throwing release velocity $\hat{v}$, which is concatenated with μ then fed into 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.

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 predefined with respect to the boundaries of the robot's workspace for picking. In our experiments, this area covers a 0.9×0.7m tabletop surface, on top of which a bin of objects can be placed. Since our heightmaps have a pixel resolution of 180×140, each pixel $i \in I$ spatially represents a 5×5mm vertical column of 3D space in the robot's workspace. Using its height-from-bottom value, each pixel i 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 [3, 10, 15] (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 stable, collision-free IK solves [6].

Grasping network. The grasping network is a 7-layer fully convolutional residual network [3, 10, 15] (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 [27], 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 pixel i determines grasping position x and the orientation of the heightmap determines grasping angle θ. We choose this visual state and action representation as it has been shown to provide sample efficiency when used in conjunction with fully-convolutional action-value functions for grasping and pushing in prior work [27, 28], since 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 middle point between the gripper fingertips

reach 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 projectile. During execution, the robot's 6DOF arm curls inwards while grasping onto an object, then uncurls outward at high speeds, releasing the projectile as soon as it meets the desired position and velocity. Throughout this motion, the gripper is oriented such that the antipodal line between the fingertips remains orthogonal to the intended aerial trajectory of the projectile. 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 grasp attempt for which an object is detected in the gripper (by thresholding on the distance between fingertips).

Estimating 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-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 $v_{x,y}$, for which our system is still aware of and will compensate for through learning. 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.

Estimating 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, 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

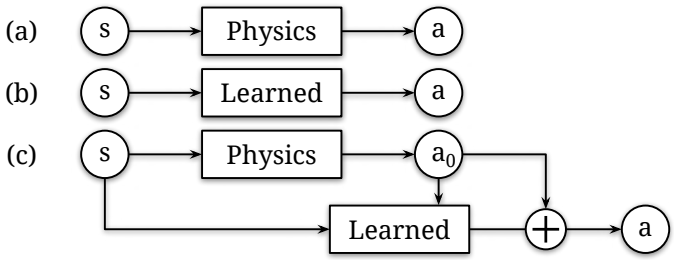


Fig. 4. Model variants: (a) analytical solutions that use physics and kinematics over state s to determine action a; (b) data-driven solutions that learn the direct mapping from states to actions; (c) hybrid solutions (like ours) use analytical solutions to obtain an initial action a_0, and combine it with a predicted residual from a learning model to obtain the final action a.

offsets to the real-world projectile velocity (conditioned on the grasp), which are otherwise not analytically modeled.

This approach to throwing, which we refer to as *Residual Physics*, falls under a broader category of hybrid controllers that leverage both 1) analytical models to provide initial estimates of control parameters (*e.g.* throwing release velocities), and 2) learned residuals on top of those estimates to compensate for unknown dynamics (see Fig. 4c). In contrast to prior work on learning residuals on predictions of future states for model-based control [2, 13], we instead directly learn the residuals on control parameters. This provides a wider range of data-driven corrections that can compensate for noisy observations as well as dynamics that are not explicitly modeled. Concurrent work on residual reinforcement learning [12, 25] investigates the benefits of residual control under variation in control and sensor noise, partial observability, and transfer from sim-to-real. Our experiments in Sec. VI show that learning residuals on top of a simple physics-based controller (using the ballistic equations of projectile motion) can yield substantial improvements in the both accuracy and generalization ability of throwing arbitrary objects than baseline alternatives: *e.g.* using only the physics-based controller (Fig. 4a), or directly training f to regress $\|v_{x,y}\|$ (Fig. 4b). This finding suggests a new general approach to learning control parameters, which may generalize to other systems.

Physics-based controller. The physics-based controller uses the standard equations of linear projectile motion (which are object-agnostic) 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 where each pixel holds the value of $\hat{v}$. 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 strictly relies on several assumptions that generally do not hold in the real world. First, it assumes that the effects of aerodynamic drag are completely 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 in real-world environments. Second, it assumes that the gripper release velocity v directly determines the velocity of the projectile. This is largely not true since the object may not necessarily be grasped at the center of mass, nor is the object completely immobilized by the grasp in all motion freedoms 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 consists of a throwing network that predicts the 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 [10] 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 entire 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 [9] 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) thresholding on the antipodal distance between gripper fingers after the grasping primitive, or 2) using the binary signal of whether or not the thrown object lands in the correct box. As we show in our experiments in Sec. VI-E, the second way of supervising grasps with 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 can obtain the ground truth residual label $\bar{\delta}_i = \|v_{x,y}\| - \|\hat{v}_{x,y}\|_{\bar{p}}$.

In our 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 settings $n = 80$+. 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 executed a series of experiments in simulated and real settings to evaluate our 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

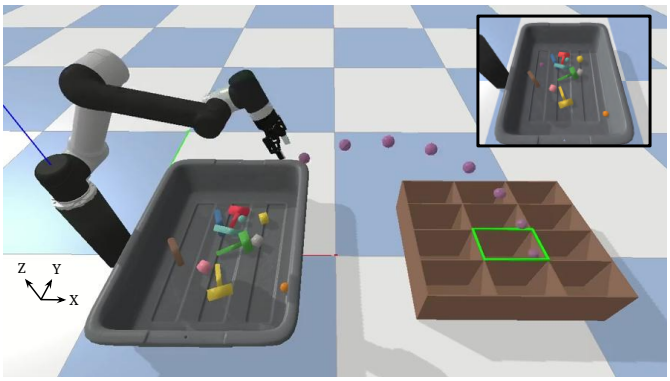


Fig. 5. **Simulation environment** in PyBullet [4]. This snapshot illustrates the aerial motion trajectory of a purple ball being thrown into the target landing box highlighted in green. The top right image depicts the view captured from the simulated RGB-D camera before the ball was grasped and thrown.

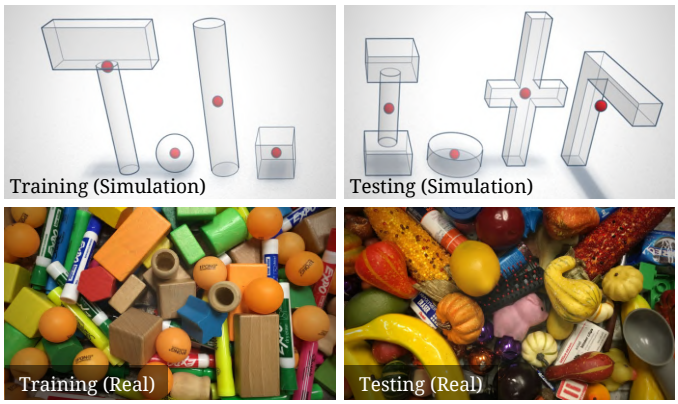


Fig. 6. **Objects** used in simulated (top) and real (bottom) experiments, split by training objects (left) and unseen testing objects (right). The center of mass for each simulation object is indicated with a red sphere (visualization only).

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 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 simulator is built using PyBullet [4] (Fig. 5). We use in total 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. 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 does not account for aerodynamics, and as a result, performance in simulation does not necessarily equate to performance in the real world. Therefore we also provide quantitative experiments on real systems.

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 on a fixed tripod overlooking the bin of objects from the side. The camera is localized with respect to the robot base using an automatic calibration procedure from [27]. 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 full 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 methods:

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***.

Human-baseline reports the average throwing accuracy and standard deviation across 15 participants (average height: 174.0±8.3cm). 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.* ,

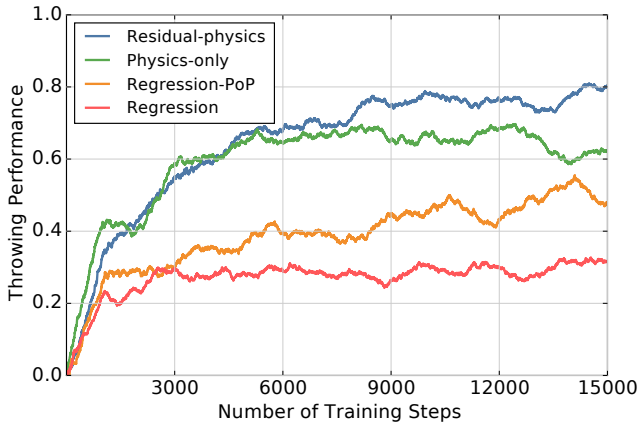


Fig. 7. Our method (Residual-physics) outperforms baseline alternatives in terms of throwing success rates in simulation on the Hammers object set.

"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 because 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. 7 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. 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.

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 contact and aerodynamics that exist in the real world, which Residual-physics is able to compensate for and still maintain a throwing accuracy above 80% for both seen and unseen objects. Interestingly, our system exceeds average untrained human-level performance on the task.

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 [20]	120
Dex-Net 2.0 [16]	250
FC-GQ-CNN [22]	296
Dex-Net 4.0 [17]	312
TossingBot (w/ Placing)	432
TossingBot (w/ Throwing)	**514**

D. Pick-and-Place Efficiency

Throwing enables our real 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 our MPPH against other state-of-the-art picking systems found in literature: Cartman [20], Dex-Net 2.0 [16], FC-GQ-CNN [22], Dex-Net 4.0 [17], 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, they suggest throwing may be useful to improve the overall MPPH for some real-world 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

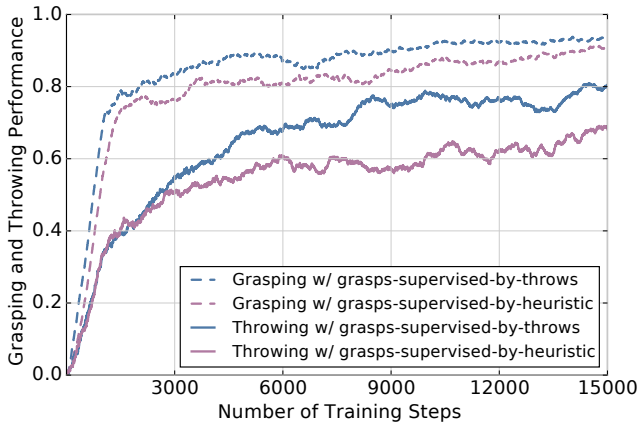


Fig. 8. Both grasping and throwing success rates of Residual-physics policies improve when grasps are supervised by the accuracy of throws (blue), versus when grasps are supervised by a heuristic that checks grasp width (purple).

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 shows that throwing accuracy is strongly influenced by the quality of grasps. Moreover, 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. 9. 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 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 enables 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. Meanwhile, 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 over-optimizes for grasping as close to the CoM as possible (without collisions). 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.

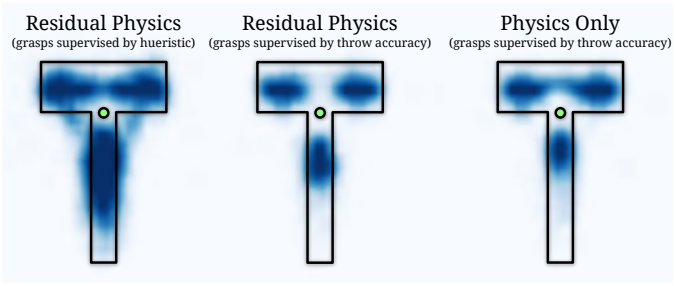


Fig. 9. Projected 2D histograms of successful grasping positions on hammers in simulation: show that 1) leveraging accuracy of throws as supervision enables the grasping policy to learn a more restricted but stable set of grasps, while 2) learning throwing in general helps to relax this constraint.

F. Generalizing to New Target Locations

To explore how well our trained policies generalize to new target locations, we shift the locations of the boxes diagonally in both the x and y axes from where they were during training, such that there is no overlap between training and testing locations. In simulation, there are 12 training and 12 testing boxes; while in real settings, there are 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.

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**

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.

VII. DISCUSSION AND FUTURE WORK

This paper presents a framework for jointly learning grasping and throwing policies that enable TossingBot, a real UR5 picking system, to pick-and-throw arbitrary objects from an unstructured bin into boxes located outside its maximum reach range at 500+ MPPH. A key research contribution of the framework is *Residual Physics*, a hybrid controller that leverages deep learning to predict residuals on top of initial estimates of control parameters from physics. This combination enables the data-driven predictions to focus on learning the aspects of dynamics that are difficult to analytically model. 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 Ryan Hickman for valuable managerial support, Ivan Krasin and Stefan Welker for fruitful technical discussions, Brandon Hurd and Julian Salazar and Sean Snyder for hardware support, Chad Richards and Jason Freidenfelds for feedback on the paper, Erwin Coumans for advice on PyBullet, Laura Graesser for video narration, and Regina Hickman for photography and videos. We are also grateful for hardware and financial support from Google, Amazon, Intel, NVIDIA, ABB Robotics, and Mathworks.

REFERENCES

[1] Eric W Aboaf, Christopher G Atkeson, and David J Reinkensmeyer. Task-level robot learning. *ICRA*, 1988. 2

[2] 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. *IROS*, 2018. 4

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *PAMI*, 2017. 3

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

[5] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996. 10

[6] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute. 3

[7] Yizhi Gai, Yukinori Kobayashi, Yohei Hoshino, and Takanori Emaru. Motion control of a ball throwing robot with a flexible robotic arm. *WASET*, 2013. 2

[8] Ali Ghadirzadeh, Atsuto Maki, Danica Kragic, and Mårten Björkman. Deep predictive policy training using reinforcement learning. *IROS*, 2017. 2

[9] Ross Girshick. Fast r-cnn. In *ICCV*, 2015. 5

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016. 3, 5

[11] Jwu-Sheng Hu, Ming-Chih Chien, Yung-Jung Chang, Shyh-Haur Su, and Chen-Yu Kai. A ball-throwing robot with visual feedback. *IROS*, 2010. 2

[12] 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. 4

[13] Alina Kloss, Stefan Schaal, and Jeannette Bohg. Combining learned and analytical models for predicting action effects. *ICRA*, 2018. 4

[14] Jens Kober, Erhan Öztop, and Jan Peters. Reinforcement learning to adjust robot movements to new situations. *IJCAI*, 2011. 2

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015. 3

[16] 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. *RSS*, 2017. 7

[17] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 2019. 7

[18] Matthew T Mason and Kevin M Lynch. Dynamic manipulation. *IROS*, 1993. 1, 2

[19] 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. *ICRA*, 2009. 2

[20] 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. *ICRA*, 2018. 7

[21] 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 *ISMAR*, 2011. 10

[22] 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. 2018. 7

[23] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *ICLR*, 2016. 10

[24] Taku Senoo, Akio Namiki, and Masatoshi Ishikawa. High-speed throwing motion based on kinetic chain approach. *IROS*, 2008. 2

[25] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv*, 2018. 4

[26] Andy Zeng, Shuran Song, Matthias Nießner, Matthew Fisher, Jianxiong Xiao, and Thomas Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *CVPR*, 2017. 10

[27] 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. *IROS*, 2018. 1, 3, 6

[28] 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. *ICRA*, 2018. 3

VIII. Appendix

The appendix consists of additional system details, analysis, and experimental results.

A. Additional Training Details

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 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 [23] 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.

B. Additional Timing Details

Our average cycle time is 5-6 seconds per successful grasp-then-throw and 3-4 seconds per grasp retry. The average cycle time of TossingBot without throwing is on average 7-8 seconds per successful grasp-then-place.

In addition to throwing, there are 3 other aspects that enable our system's picking speeds: 1) fast algorithmic runtime speeds (220ms for inference), 2) real-time TSDF fusion [5, 21, 26] of RGB-D data, which enables us to capture and aggregate observed 3D 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:

Algorithm 1 System Pipeline

Initialize *robot*.
Initialize policy with model f.
Initialize replay *buffer*.
while step $i < N$ and not *terminate* **do**
  I^i = *robot*.CaptureState()
  p^i = *robot*.SelectTarget()
  ϕ_g^i,ϕ_t^i = f.Inference(I^i,p^i)
  while *robot*.is_grasping **do**
    f.ExperienceReplay(*buffer*)
  y^{i-1} = *robot*.CheckGraspSuccess()
  robot.ExecuteThrow(ϕ_t^{i-1},p^{i-1}) ▷ asynchronous
  while *robot*.is_throwing **do**
    f.ExperienceReplay(*buffer*)
  robot.ExecuteGrasp(ϕ_g^i) ▷ asynchronous
  $\bar{p}^{i-1}$ = *robot*.TrackLanding()
  buffer.SaveData(I^{i-1},p^{i-1},ϕ_g^{i-1},ϕ_t^{i-1},y^{i-1},$\bar{p}^{i-1}$)
  $i = i + 1$

C. Additional Details of Inferring $\|v\|$ and r

Assuming a fixed throwing release height r_z, fixed release distance c_d from robot base origin, and release velocity direction angled 45° upwards: for any given target landing location $p = (p_x, p_y, p_z)$, we can derive the 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.

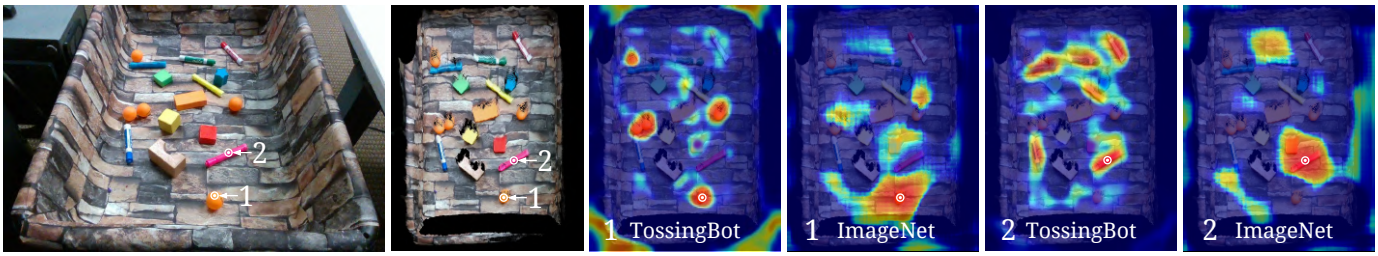
D. Additional Details of Human Baseline Experiments

To measure human throwing performance, 15 willing participants were asked to stand in place of the robot in the real-world setup, 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 are 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).

Interestingly, human performance was lower than we had expected. The largest contributor to poor performance was fatigue – the accuracy of throws deteriorates over time, particularly after around the 20th object regardless of picking speeds. The second largest contributor to performance was the physical height of the participant, which determines both throwing distance (measured from grasp release to object landing locations, which is smaller for taller participants with longer arms), as well as throwing strategy (taller participants performed better and often preferred overhand over underhand throws). Other throwing strategies that participants adopted include: 1) largely using tactile feedback to grasp objects in the bin so that visual field of view remains in focus 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) and grouping objects by weight, then correspondingly changing to different grasping and throwing strategies. These additional strategies were interesting, but did not seem to strongly correlate with better performance.

E. Additional Visualizations of Learned Grasps

In this section, we further explore the interaction between learned grasps and throws. Towards this end, we provide additional 2D grasp histogram visualizations in Fig. 11 for all simulation objects. The histograms are generated using the procedure described in Sec. VI-E for successful grasps, grasps that lead to successful throws, and grasps that lead to failed throws – recorded over 15,000 training steps. Darker regions



(a) (b) (c) (d) (e) (f)

Fig. 10. **Emerging semantics from interaction.** Visualizing pixel-wise deep features μ learned by TossingBot (c,e) overlaid on the input heightmap image (b) generated from an RGB-D side-view (a) of a bin of objects. (c) shows a 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) shows a 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). For reference, the same visualization technique is used on deep features generated by a ResNet-18 pre-trained on ImageNet (d,f).

indicate more grasps. The silhouette of each object is outlined in black, with a green dot indicating its CoM.

In line with the observations drawn in the main paper, the differences between columns 1 and 4 indicate that leveraging accurate throws as a supervisory signal for the grasping policy enables it to learn a more restricted but stable set of grasps: slightly further from CoM to avoid unintentional collisions between fingers and the rest of the object at the moment of release, but also further from the ends of the handle to avoid less predictable throws. Furthermore, the differences between columns 4 and 7 continue to show that when using only physics for the throwing module, the grasping policy over-optimizes for grasping as close to the CoM as possible (without collisions). This leads to a more restricted set of grasps in contrast to column 4, where the throwing can learn to compensate respectively.

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 not trivially conditioned on how it grasps.

F. Emerging Object Semantics from Interaction

In this section, we explore the deep features being learned by the neural network f – i.e., "What does TossingBot learn from grasping and throwing arbitrary objects?" and "Do they convey any meaningful structure or representation?" 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 heightmap I, and feed it through the network f (already trained for 15,000 steps from the real experiments). Training objects include marker pens, ping pong balls, and wooden toy blocks of different shapes (see Fig. 10). We then extract the intermediate spatial feature representation of the network μ (described in Sec. III-A of the main paper), which essentially holds 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 ping pong balls in the bin, and visualize its distance to all other pixel-wise features as a heatmap in Fig. 10a (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).

Interestingly, 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. 10b 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. 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 pre-trained on ImageNet (also shown in Fig. 10).

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 pre-defined using human-fabricated image datasets and manually constructed class categories (*i.e.*, this is a "hammer", and this is a

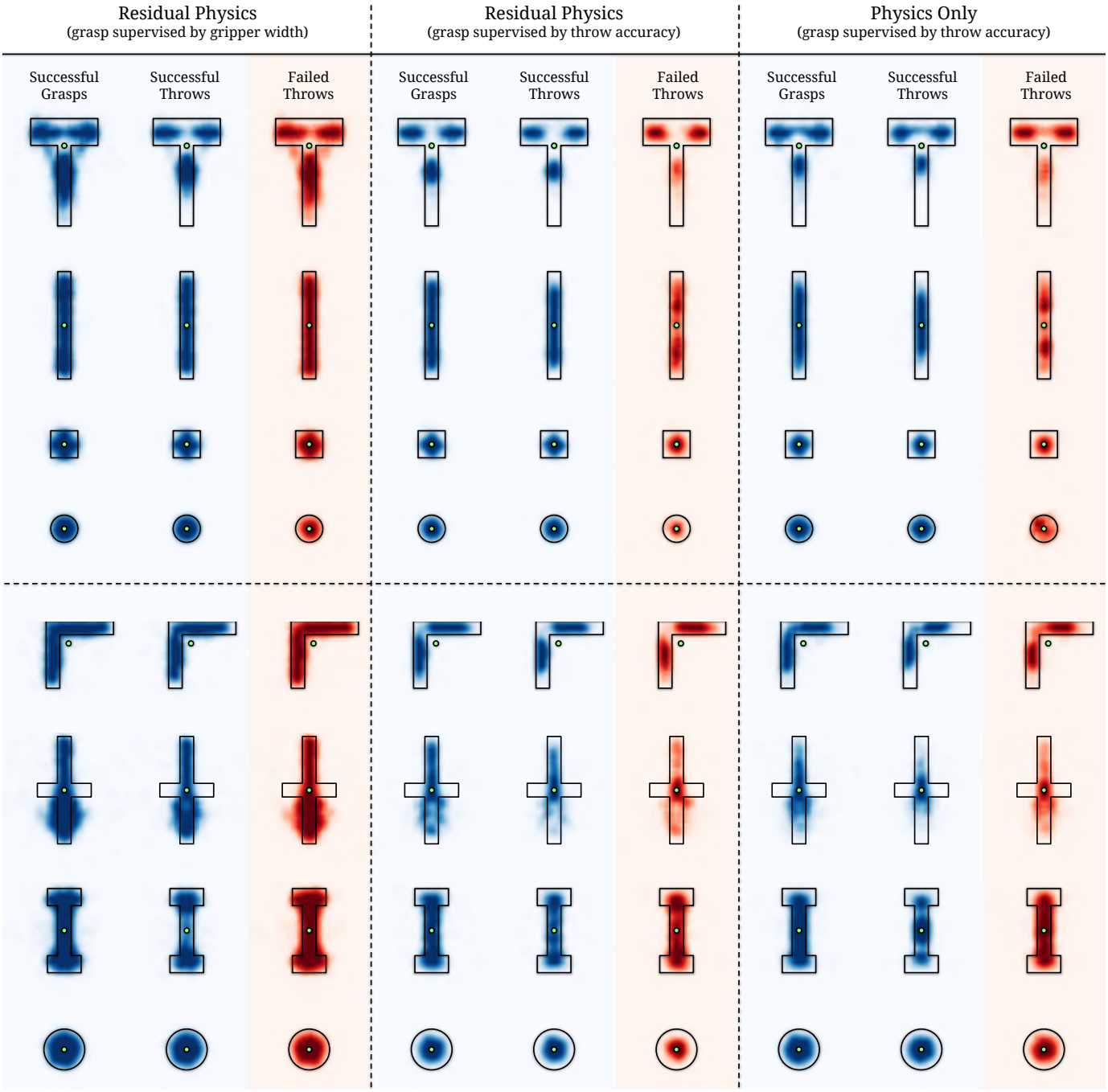


Fig. 11. **Additional grasping histograms** of all simulation objects. Histograms are generated for successful grasps, grasps that lead to successful throws, and grasps that lead to failed throws – recorded over 15,000 training steps. Darker regions indicate more grasps. The silhouette of each object is outlined in black, with a green dot indicating its CoM.

"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. Towards more generally intelligent robots – perhaps it is sufficient for them to develop their own notion of semantics through interaction, without requiring any human intervention.