TossingBot

Learning to Throw Arbitrary Objects with Residual Physics

https://tossingbot.cs.princeton.edu/

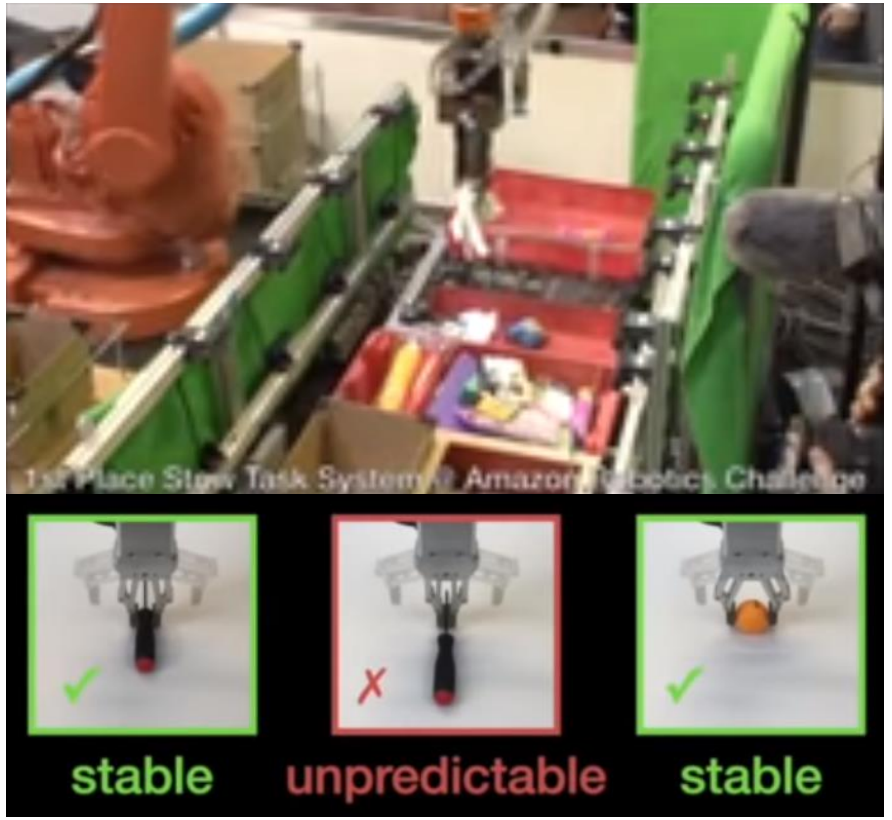
Weiwen Chen

Motivation

- Today's robo-picking: **slow, rigid**
- Human picking: **fast, involves tossing**

- The target is **outside** the range of robot

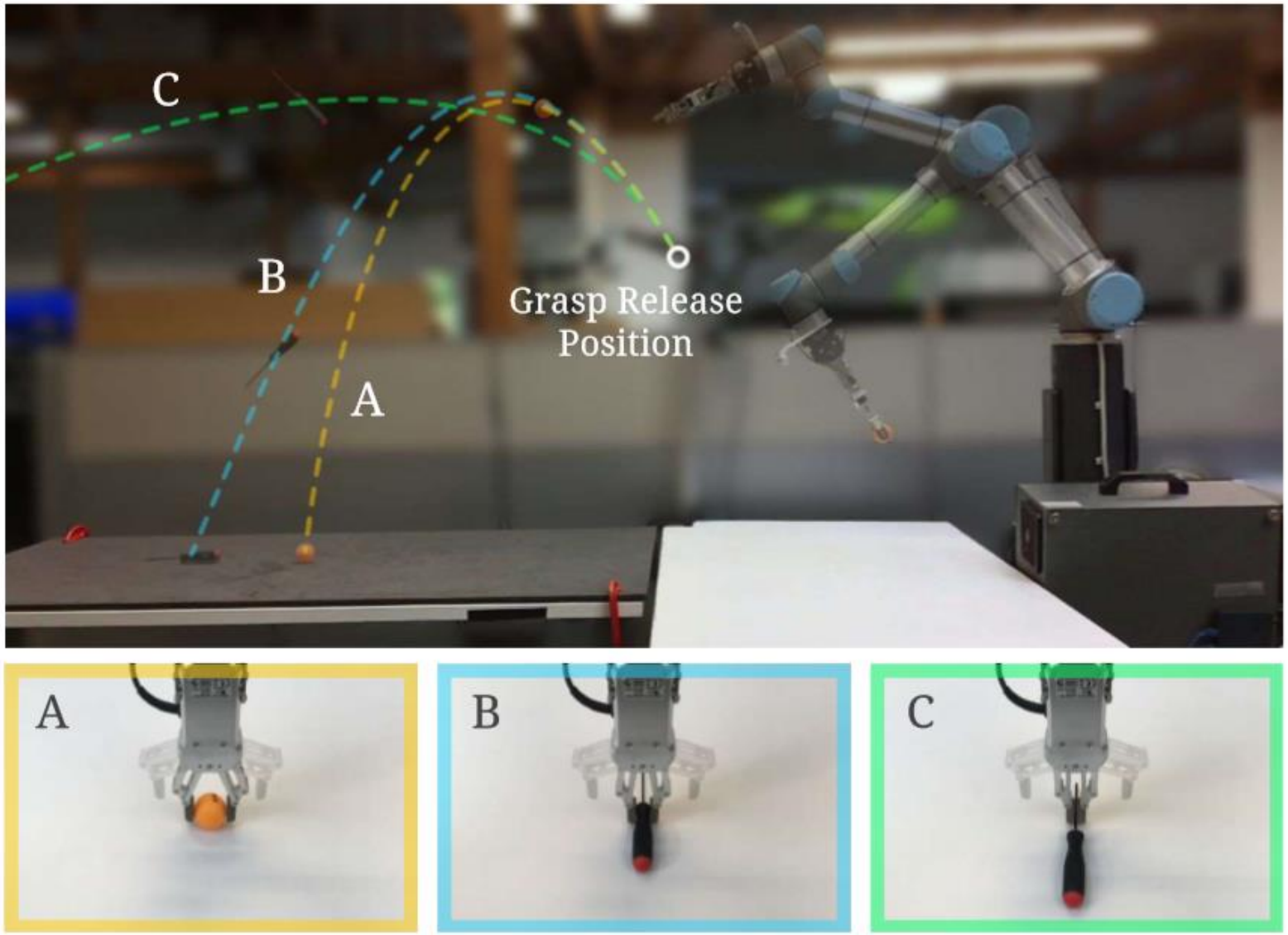
- Drawbacks of pre-work:
  - Limited object types(balls, cubes)
  - pre-throw conditions keeps all the same



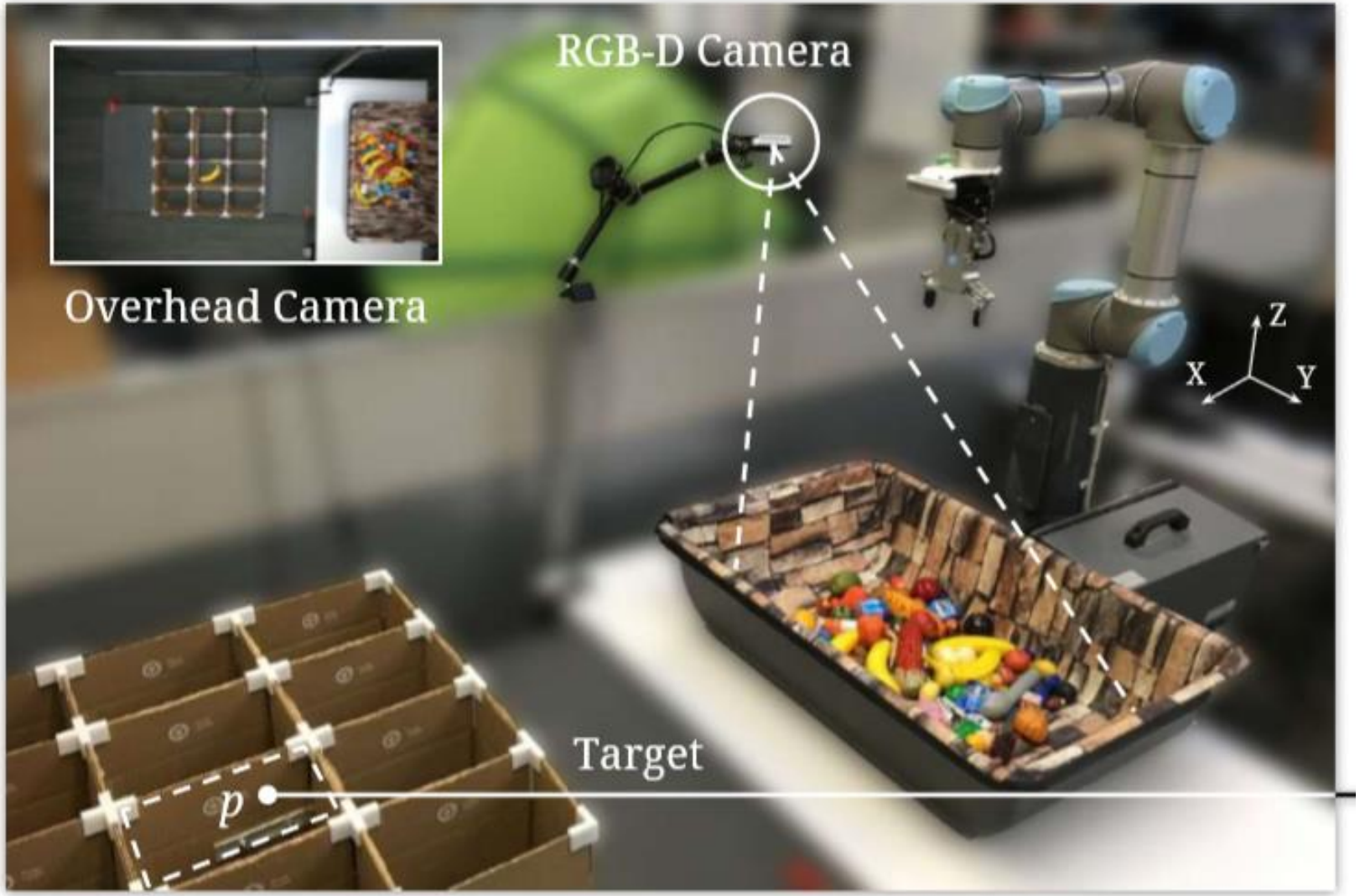
Overview

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.

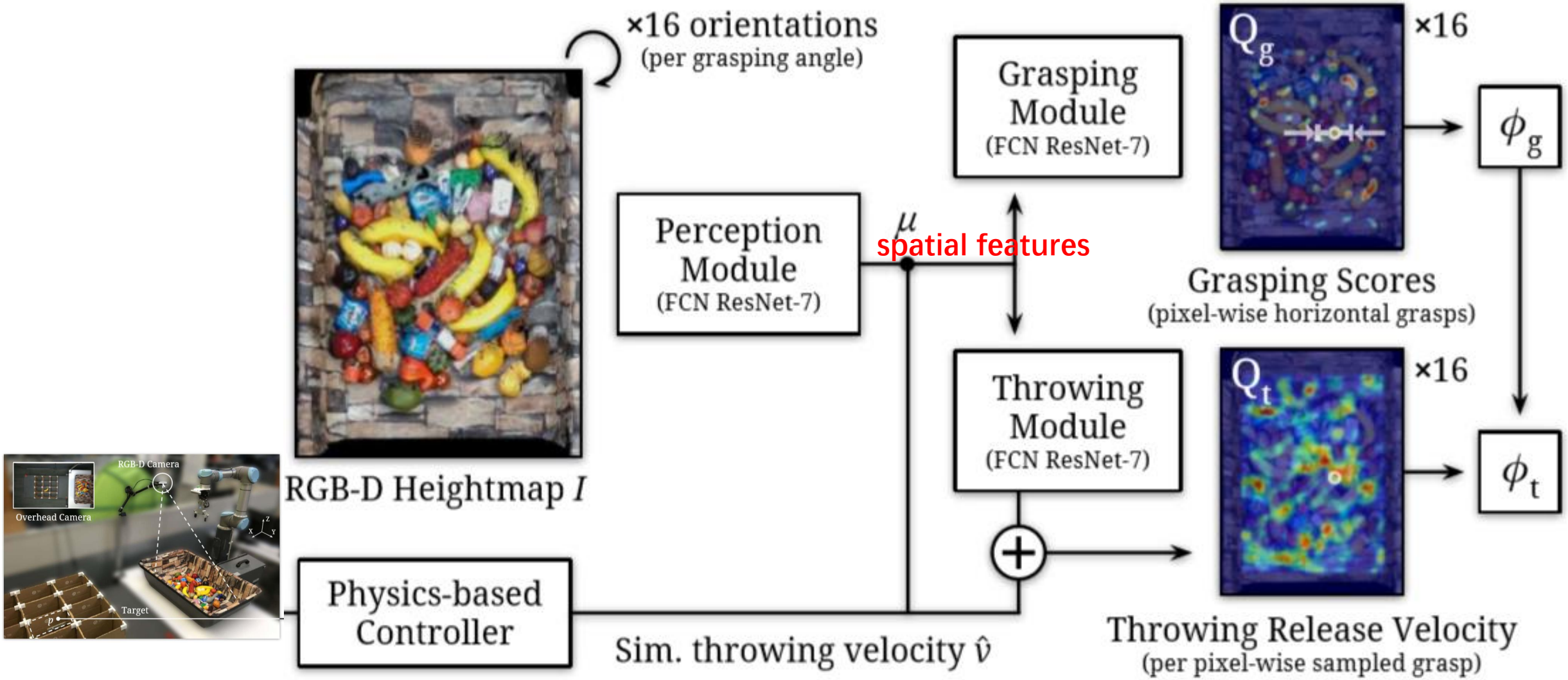
Projectile trajectories



Overview



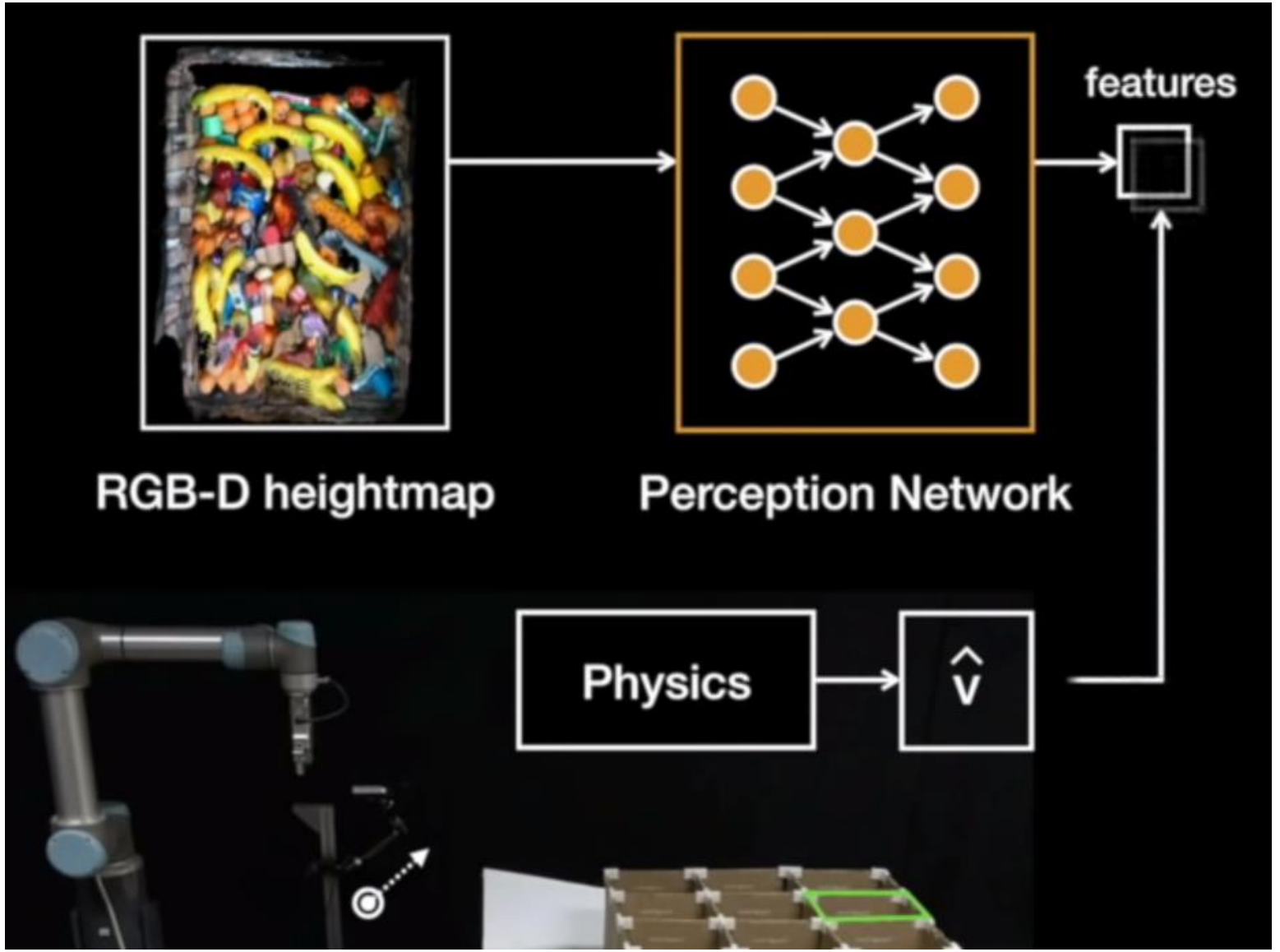
Overview


×16 orientations
(per grasping angle)
Grasping Module
(FCN ResNet-7)
Perception Module
(FCN ResNet-7)
μ
spatial features
Q_g
×16
ϕ_g
Grasping Scores
(pixel-wise horizontal grasps)
Throwing Module
(FCN ResNet-7)
Q_t
×16
ϕ_t
RGB-D Heightmap I
RGB-D Camera
Overhead Camera
Target
Physics-based Controller
Sim. throwing velocity $\hat{v}$
Throwing Release Velocity
(per pixel-wise sampled grasp)

Overview

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

Perception Module

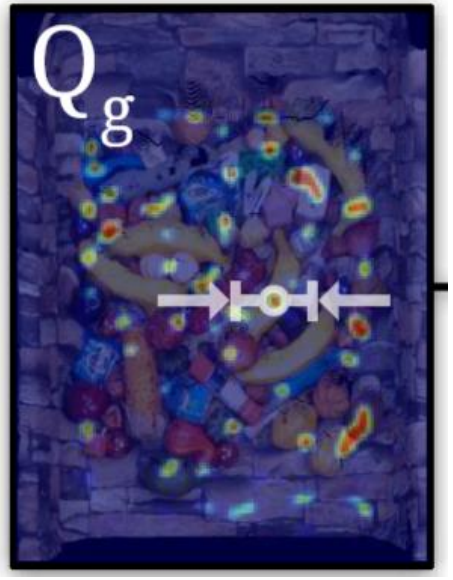


Learning Visual Representations

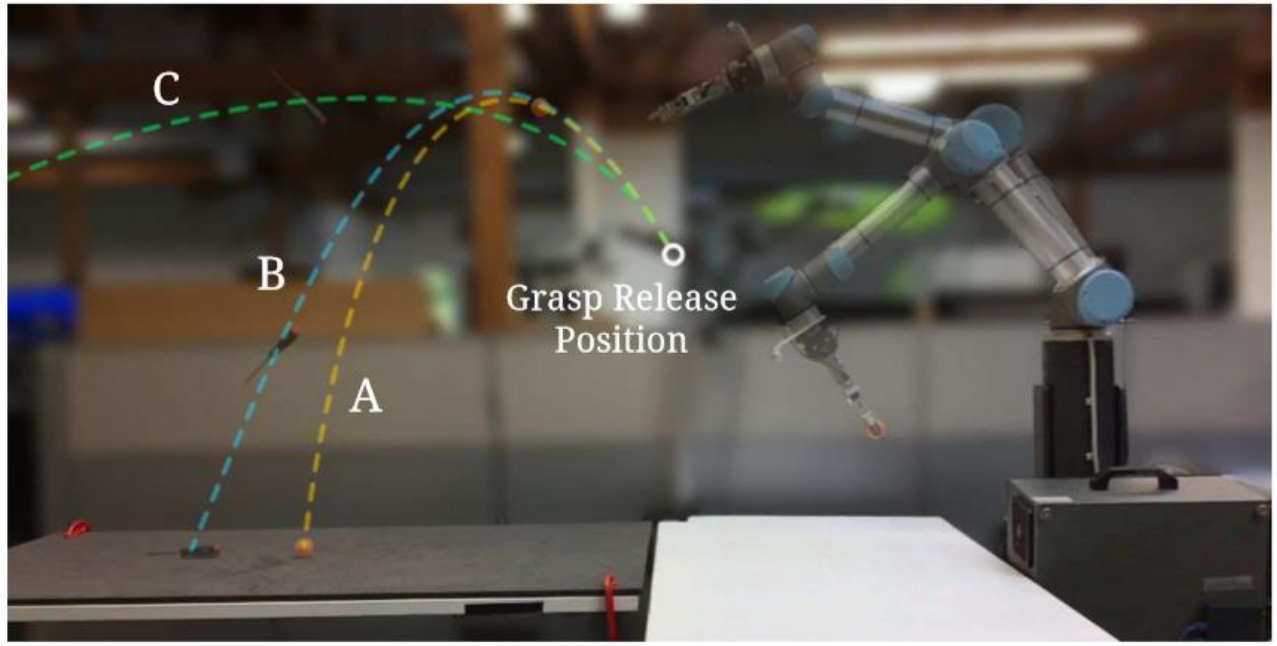
workspace for picking. In our experiments, this area covers a 0.9×0.7m tabletop surface, on top of which a bin of objects can be placed. Since our heightmaps have a pixel resolution of 180×140, each pixel $i \in I$ spatially represents a 5×5mm vertical column of 3D space in the robot's workspace. Using 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.

Grasping Network

Grasping network. The grasping network is a 7-layer fully convolutional residual network [3, 10, 15] (interleaved with 2 layers of spatial bilinear 2× 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



Throwing Module



Throwing primitive. The th
parameters $\phi_t = (r, v)$ and e
such that the middle point

reach a desired release position $r = (r_x, r_y, r_z)$ a
$v = (v_x, v_y, v_z)$, at which point the gripper opens a

Estimating release position

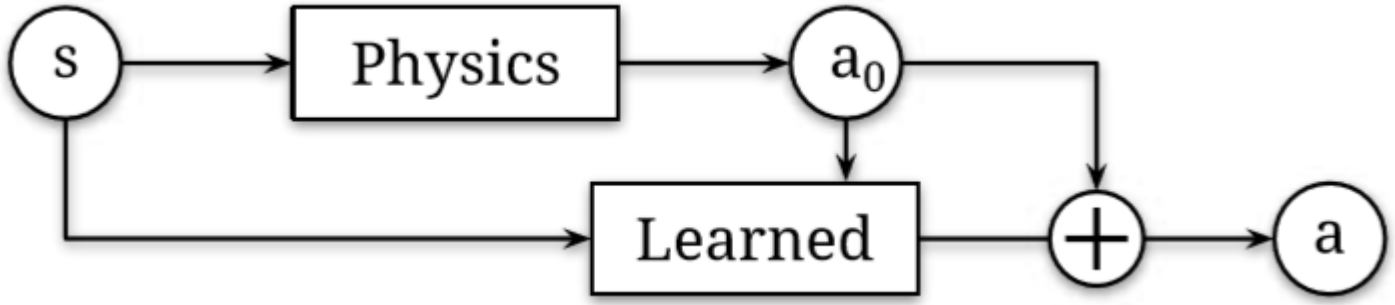
location p using two assumptions: 1) the aerial trajectory of a projectile is linear on the xy-plane and in the same direction as $v_{x,y} = (v_x, v_y)$. In other words, we assume that the forces of aerodynamic drag *orthogonal* to $v_{x,y}$ are negligible. This is not to be confused with the primary 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

Estimating release velocity

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

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

RESIDUAL PHYSICS



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

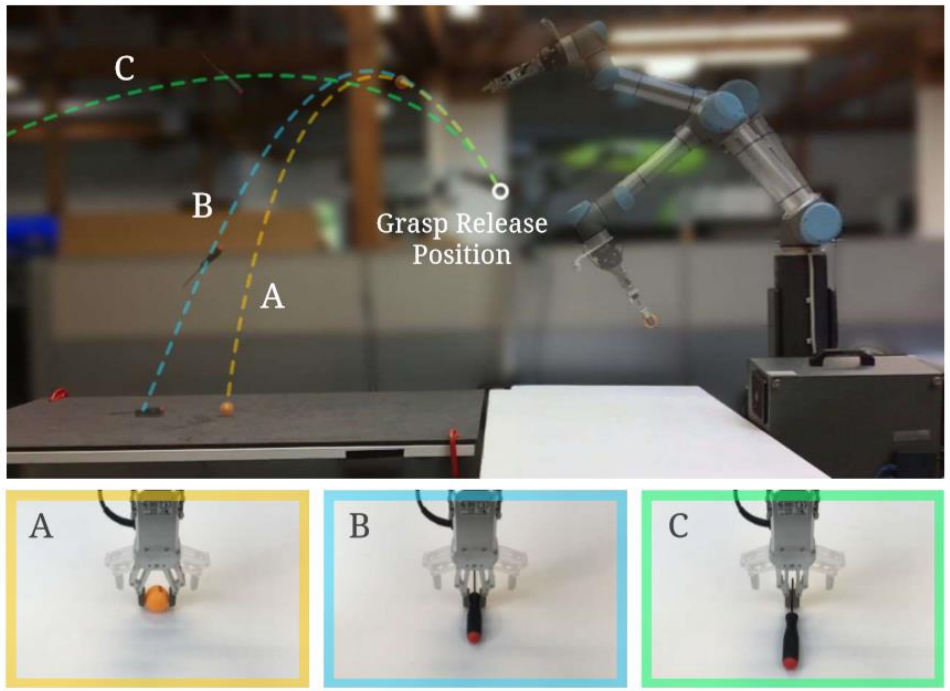
Physics-based controller

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 $a_z = -9.8\text{m/s}^2$.

However

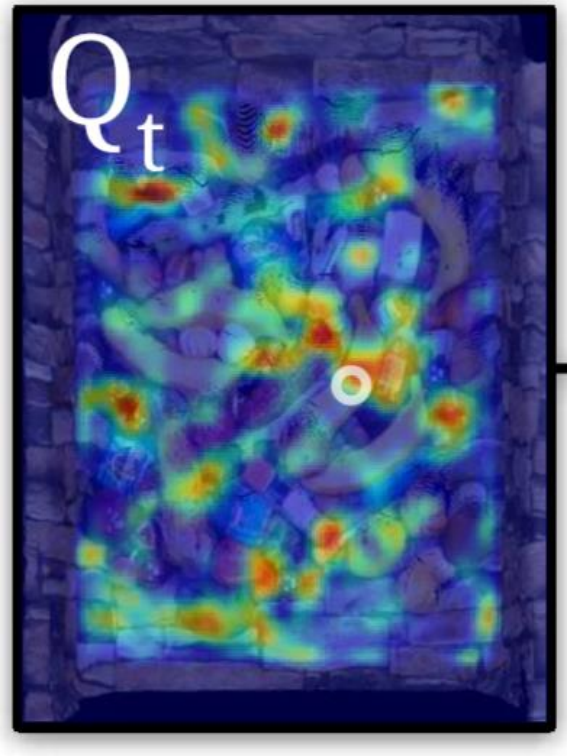


for v. However, it also strictly relies on several assum
that generally do not hold in the real world. First, it as
that the effects of aerodynamic drag are completely negl
However, as we show in our experiments in Fig. 2, the
trajectory for lightweight objects like ping pong balls
substantially influenced 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,

Residual Release Velocity

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

(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.



Loss

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

Simulation

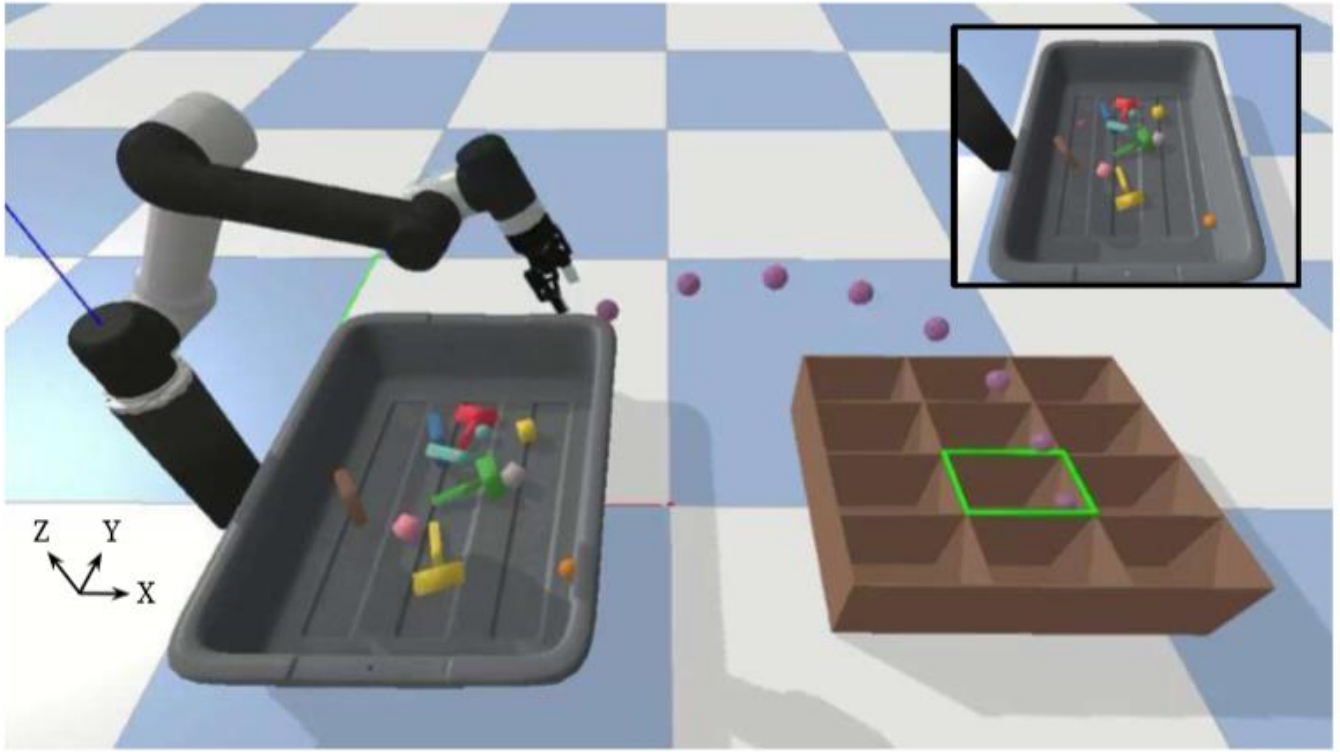


Fig. 5. **Simulation environment** in PyBullet [4]. This snapshot illustrates the aerial motion trajectory of a purple ball being thrown into the target landing box highlighted in green. The top right image depicts the view captured from the simulated RGB-D camera before the ball was grasped and thrown.

Objects

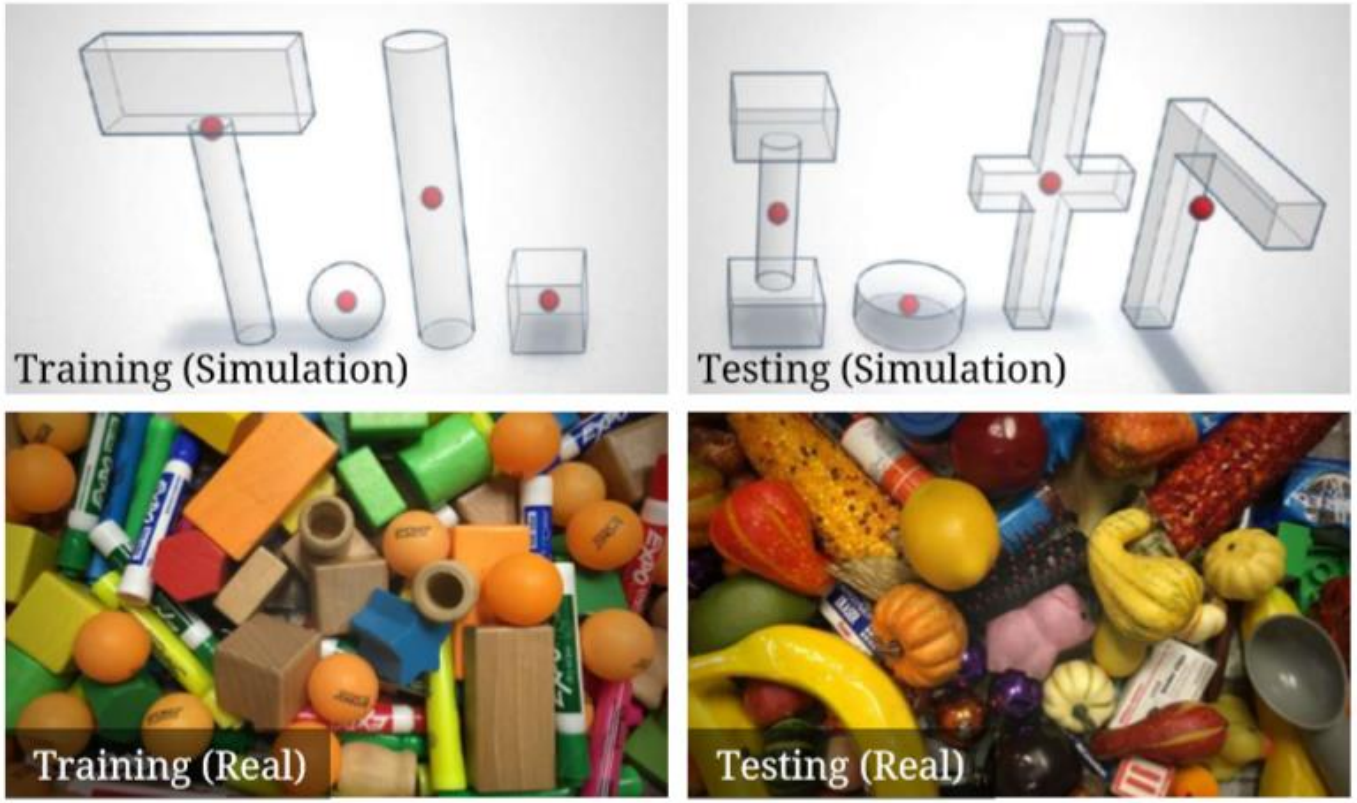


Fig. 6. **Objects** used in simulated (top) and real (bottom) experiments, split by training objects (left) and unseen testing objects (right). The center of mass for each simulation object is indicated with a red sphere (visualization only).

Newbee

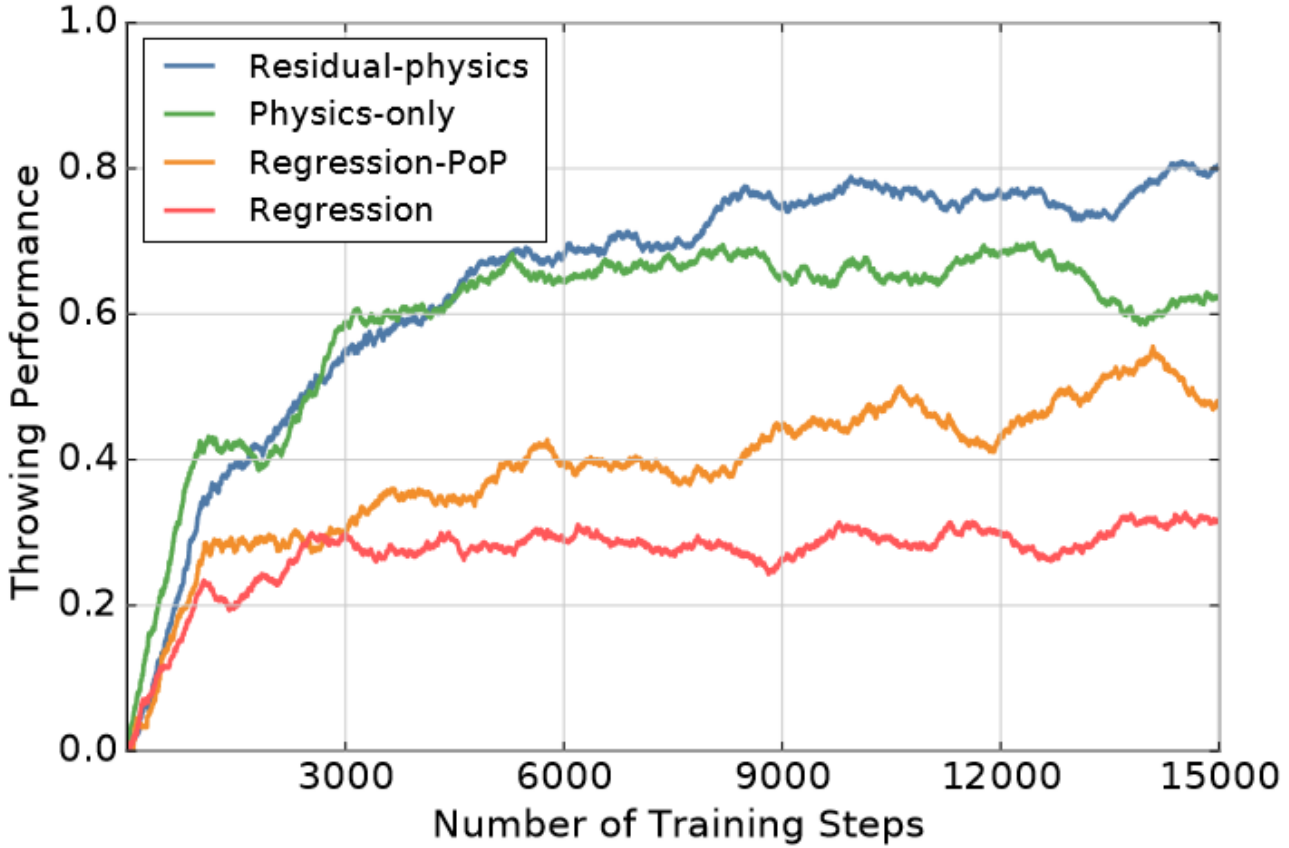


Fig. 7. Our method (Residual-physics) outperforms baseline alternatives in terms of throwing success rates in simulation on the Hammers object set.

Newbee

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

Method	Grasping		Throwing	
	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**

ummm

- Two arms?
- Fake
- HAO DAI A

QA