TossingBot: Learning to Throw Arbitrary Objects with Residual Physics

Summary

They proposed a system that learns to predict control parameters of grasp and throw directly from visual observations jointly:

- Discovering grasps that enable accurate throws
- Learning throws that compensate for the dynamics of arbitrary objects.

Grasping is directly supervised by the accuracy of throws (grasp success = accurate throw). End-to-end policies learn

- to execute stable grasps that lead to predictable throws
- throwing velocities that can account for the variations in object-centric properties and dynamics.

- Jointly learning grasping and throwing → Since if robot grasp objects more stable, it get better throw accuracy(grasp is directly supervised by throw accuracy). It also allowed network to generalize its throws to different grasp locations of objects.
- Throwing module learns a residual δ on top of an initial estimate vˆ from a physics-based controller → Residual system learn to compensate for different grasp locations, and object dynamics. While physic based controller allow system to generalize well to different throw location.

Challenges

- 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).

- Learning directly on real robot. Creating a system that can learn from real data:
    - Resets were done by clever environment design.
    - Supervision of whether throw was success was designed.

Framework

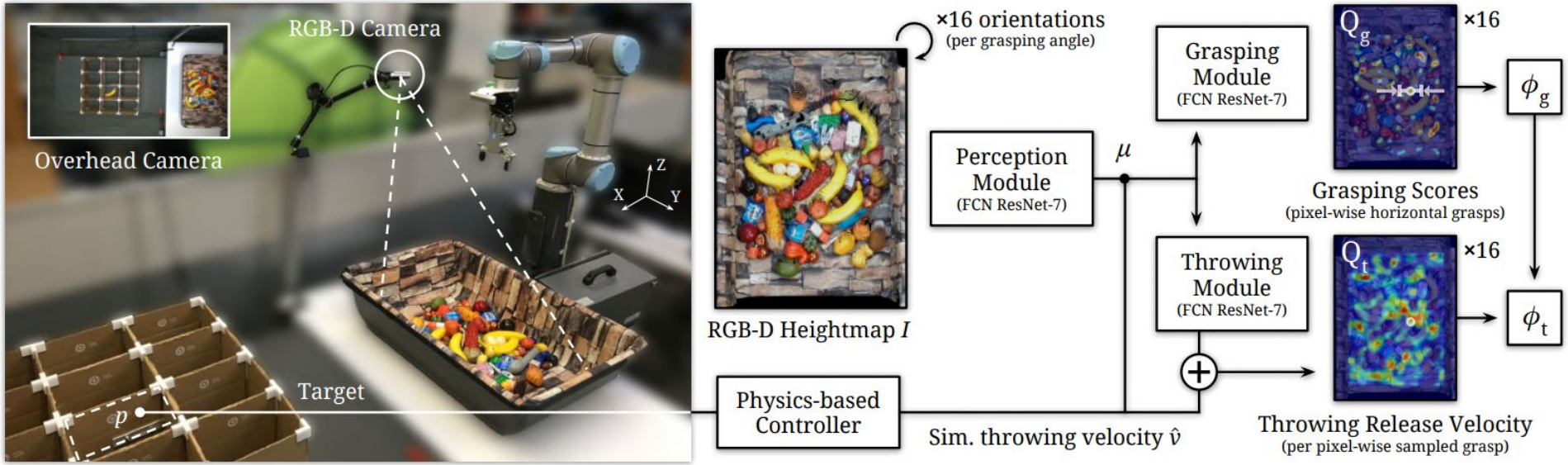


Fig. 3. **Overview.** An RGB-D heightmap of the scene is fed into a perception module to compute spatial features μ. In parallel, target location p is fed into a physics-based controller to provide an initial estimate of throwing release velocity $\hat{v}$, which is concatenated with μ then fed into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

Perception Module: Learning Visual Representations

Input : Topdown RGB-D image I with pixel resolution of 180×140.

- Each pixel $i \in I$ spatially represents a 5×5mm patch and thereby corresponds to a unique 3D location in the robot's workspace.

Network : 7-layer fully convolutional ResNet with pooling.

Output : Spatial feature representation μ of size 45×35×512

Grasping Module: Learning Parallel-jaw Grasps

Grasping Primitive:

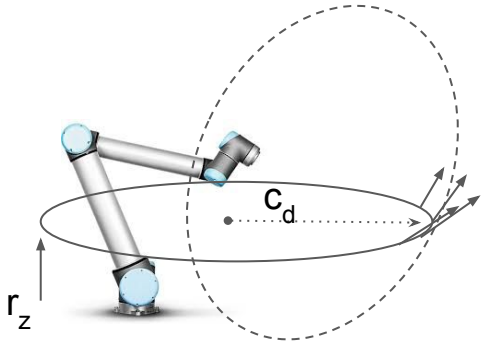
Input : Grasping angle and position. Position is acquired based on probability of success on grasp on each pixel. Angle is acquired by having 16 rotated images corresponding the different angles. This representation is selected because it provides sample efficiency. (Each pixel-wise prediction shares convolutional features for all grasping locations and orientations).

Network : 7-layer fully convolutional ResNet with upsampling so that, output has same pixel resolution.

Throwing Module: Learning Throwing Velocities

Throwing primitive: Takes throw location and velocity as input. Gripper orientation was set in a way that throw angle will be orthogonal to grippers fingertips.

Throw location is provided depending on goal throw position(no network output).



45° throwing angle

$\|v_{x,y}\|=v_z$: magnitude of the final release velocity

- only one velocity value was enough with this formulation

r_x and r_y are calculated based on throw direction. Where $\text{sqrt}(r_x^2 + r_y^2)=c_d$

Estimating v_z

Throwing module learns to predict a residual δ on top of the estimated release velocity $\|v'_{x,y}\|$ from a physics-based controller:

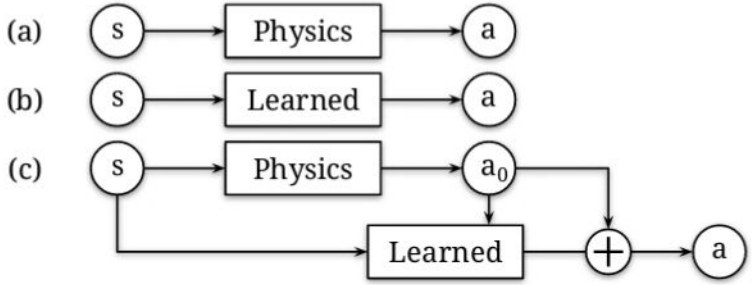


Fig. 4. Model variants: (a) analytical solutions that use physics and kinematics over state s to determine action a; (b) data-driven solutions that learn the direct mapping from states to actions; (c) hybrid solutions (like ours) use analytical solutions to obtain an initial action a_0, and combine it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

- $\|v_{x,y}\| = \|v'_{x,y}\| + \delta$

1) Analytical models provide initial estimates of control parameters (e.g. throwing release velocities)
2) Learned residuals compensates for unknown dynamics

Throwing network uses same structure as grasping network, it outputs a δ for each pixel location.

Jointly Learning Grasping and Throwing

Loss function: $L=L_g+y_iL_t$

L_g is binary cross entropy loss, y_i is binary mask for whether grasp is success. L_t is Huber loss(similar to MSE but more robust to outliers.)

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

δ_i' is ground truth residual:

- Calculated similar to HER: Even if the throw was fail for selected goal position, it was success for different goal throw location.
- Network is trained as if robot actually wanted to throw object at where object lands.(They used prioritized experience replay strategy.)

Evaluation

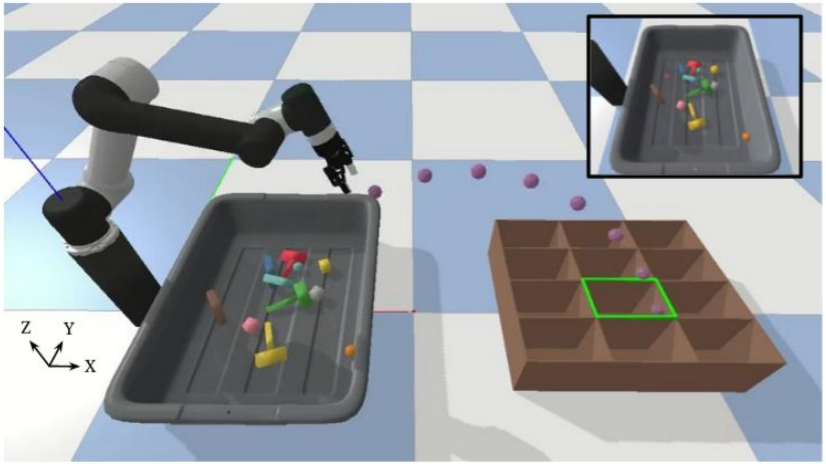


Fig. 5. **Simulation environment** in PyBullet [4]. This snapshot illustrates the aerial motion trajectory of a purple ball being thrown into the target landing box highlighted in green. The top right image depicts the view captured from the simulated RGB-D camera before the ball was grasped and thrown.

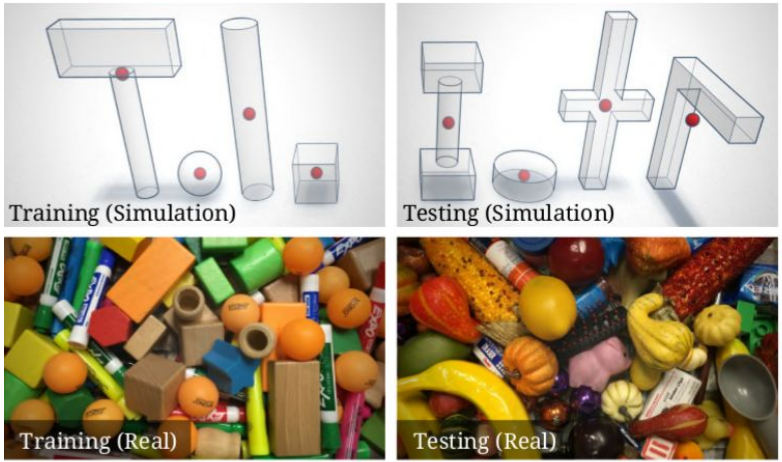


Fig. 6. **Objects** used in simulated (top) and real (bottom) experiments, split by training objects (left) and unseen testing objects (right). The center of mass for each simulation object is indicated with a red sphere (visualization only).

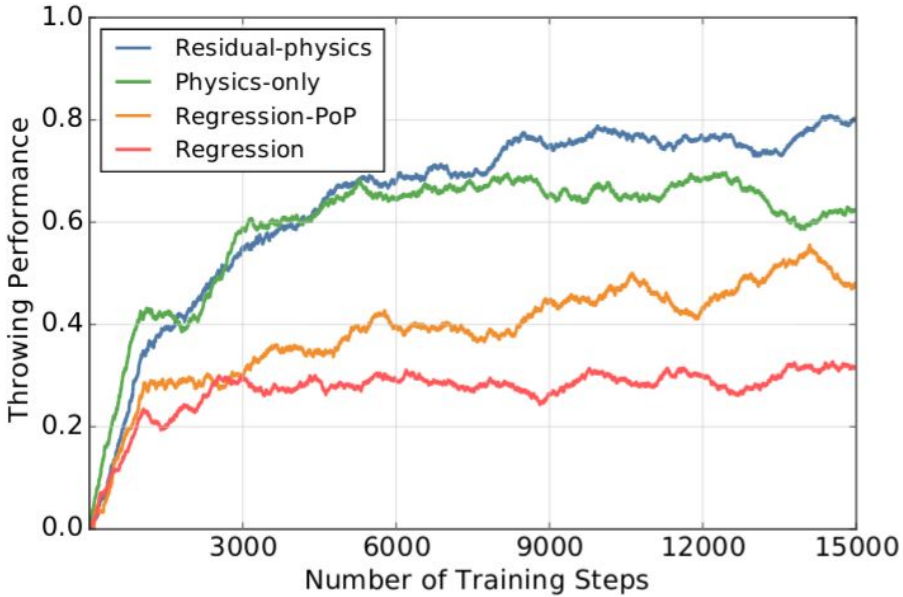


Fig. 7. Our method (Residual-physics) outperforms baseline alternatives in terms of throwing success rates in simulation on the Hammers object set.

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

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**

Stability of Grasp

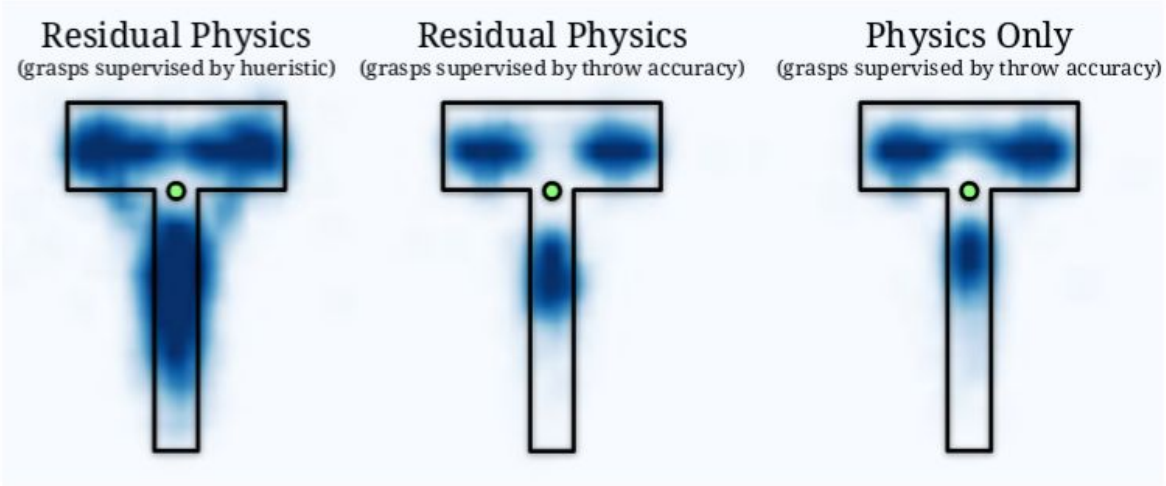


Fig. 9. Projected 2D histograms of successful grasping positions on hammers in simulation: show that 1) leveraging accuracy of throws as supervision enables the grasping policy to learn a more restricted but stable set of grasps, while 2) learning throwing in general helps to relax this constraint.

Generalizing to New Target Locations

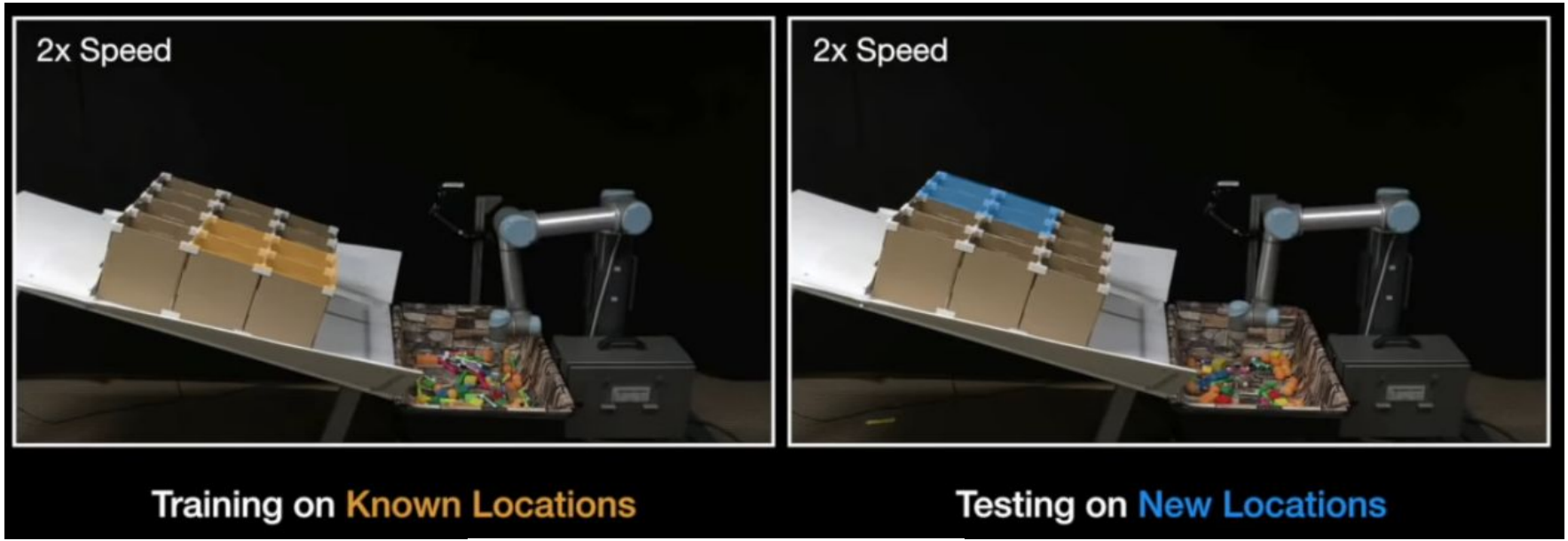
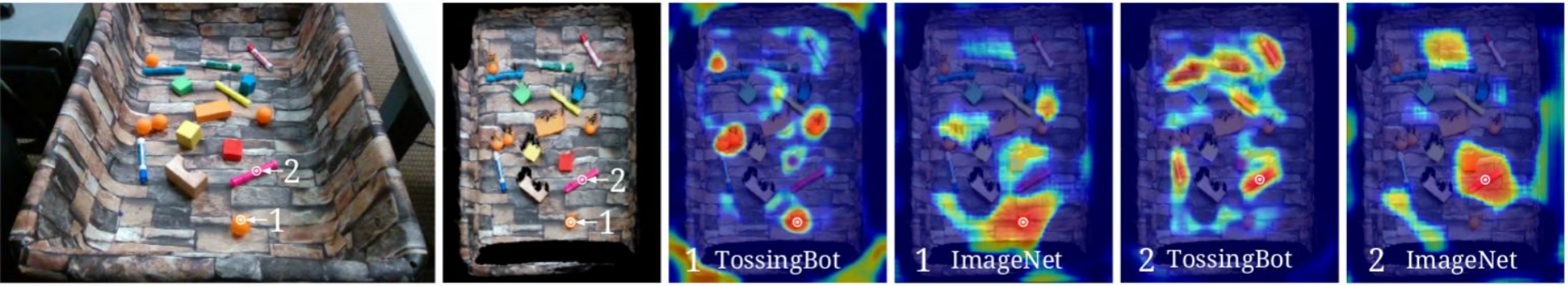


TABLE V
THROWING TO UNSEEN LOCATIONS (MEAN %)

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



(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).