TuneNet: One-Shot Residual Tuning for System Identification and Sim-to-Real Robot Task Transfer

Adam Allevato[*], **Elaine Schaertl Short**[†], **Mitch Pryor**[*], **and Andrea Thomaz**[†]
[*]Department of Mechanical Engineering
[†]Department of Electrical and Computer Engineering
The University of Texas at Austin
{allevato,mpryor}@utexas.edu, {eshort,athomaz}@ece.utexas.edu

Abstract: As researchers teach robots to perform more and more complex tasks, the need for realistic simulation environments is growing. Existing techniques for closing the reality gap by approximating real-world physics often require extensive real world data and/or thousands of simulation samples. This paper presents TuneNet, a new machine learning-based method to directly tune the parameters of one model to match another using an *iterative residual tuning* technique. TuneNet estimates the parameter difference between two models using a single observation from the target and minimal simulation, allowing rapid, accurate and sample-efficient parameter estimation. The system can be trained via supervised learning over an auto-generated simulated dataset. We show that TuneNet can perform system identification even when the true parameter values lie well outside the distribution seen during training, and demonstrate that simulators tuned with TuneNet outperform existing techniques for predicting rigid body motion. Finally, we show that our method can estimate real-world parameter values, allowing a robot to perform sim-to-real task transfer on a dynamic manipulation task unseen during training. Code and videos are available online at http://bit.ly/2lf1bAw.

1 Introduction

Recent research has validated simulators for real-world robot behaviors and control policies. Several studies have shown the ability to adapt simulation-learned policies to the real world based on observations [1, 2, 3, 4, 5, 6, 7], but these techniques require extensive real-world data collection. Even before collecting data in the real world, however, we can improve simulations by simply selecting more realistic simulator parameter values. While some approaches train over a large set of possible parameter values to create a robust policy [8, 9, 10, 11, 2], we are interested in the problem of creating a single "canonical" simulation that behaves as realistically as possible, which can then be used for robust physics prediction or better task planning.

In this paper, we propose TuneNet, a *residual tuning* technique that uses a neural network to modify the parameters of one physical model so it approximates another. TuneNet takes as input observations from two different models (i.e. a simulator and the real world), and estimates the difference in parameters between the models. By estimating the parameter gradient landscape, a small number of iterative tuning updates enable rapid convergence on improved parameters from a single observation from the target model. TuneNet is trained using supervised learning on a dataset of pairs of auto-generated simulated observations, which allows training to proceed without real-world data collection or labeling.

Our primary contribution is the concept of *residual tuning*, which allows fast and accurate one-shot system identification, and the development and analysis of the TuneNet neural network model for applying iterative parameter updates. We show that TuneNet is **fast** and **effective**: it tunes more efficiently than other gradient-free optimization methods, and that it outperforms state-of-the-art techniques for predicting rigid body motion. Finally, we validate that TuneNet can tune simulations to match real-world environments, and use those simulations to conduct sim-to-real task learning by teaching a robot to complete a task that requires accurate dynamics prediction.

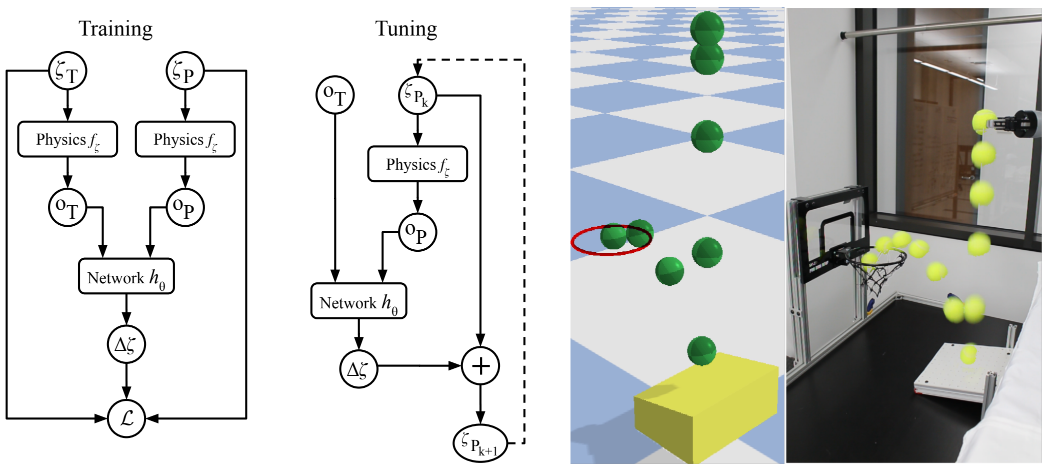


Figure 1: TuneNet uses observations to narrow the reality gap by tuning the physical parameters of a simulator so that it more closely approximates the real world (left), allowing sim-to-real robot skill transfer (right).

2 Related Work

This paper focuses on tuning simulation parameters as a way to help rapidly close the reality gap. This places it at the intersection of system identification and sim-to-real learning.

Classical system identification [12, 13] used maximum likelihood methods to estimate the parameters of a fixed model given observed behavior. For estimating parameters of complex non-differentiable models (such as physics engines), robotics researchers have used a variety of gradient-free [2, 14, 15, 16] or gradient-approximating [17] optimization techniques. These techniques require sampling the model (e.g. performing simulation rollouts) many times with different parameter values, which is a computationally expensive process. TuneNet seeks to minimize the number of simulation rollouts during system identification. Research has shown the ability to perform robot system identification from a few observations using a neural network [18], but cannot take advantage of previous parameter estimations, and the results have not been validated in a sim-to-real setting.

In contrast to system identification, recent work ([10, 2]) has explored data-driven methods for simulation improvement by leveraging a combination of real-world data and domain randomization [8]. These adaptive approaches require huge numbers of randomized simulations to be repeated once the robot is in the target environment.

TuneNet is also similar to techniques that learn a transformation from a simulated environment to a target environment. These techniques leverage physics simulators as priors, while also accounting for simulator inaccuracies. This may involve a neural network-based transformation over states [4] or actions [3], learning a new real-world policy based on one learned in simulation [7], or using a physics model and a data-driven *residual* to improve physics prediction [19, 20, 5] or policy learning [1, 5, 6]. All of these approaches require a substantial amount of real-world training examples from the target environment to learn a transformation or develop a policy. TuneNet learns to account for errors in parameter space rather than action or state space, and is pretrained in simulation, allowing a model to be adapted using a single observation from the target environment.

A recent related work is James et al. [21], which trained a network to convert from randomized simulation images to non-randomized simulations, and showed that this network could also convert real-world pictures to simulations. Zhang et al. [22] took a similar approach, using adversarial domain adaptation [23] and unlabeled data from the target environment to convert real images to simulated ones. Both of these works operate in the visual domain and focus on learning the arrangement of objects in an environment, whereas our approach focuses on learning better simulator physical parameters by observing the world state over time. Finally, Xu et al. [24] discovered physics parameters through robot interaction, but the possible parameter values are fixed and the approach does not take advantage of information that could be gleaned from an existing simulation.

3 One-Shot Residual Tuning

In this section, we describe our approach for tuning a simulator to match real-world observations. We present the underlying theory first, followed by our algorithm and implementation.

3.1 Tuning a Parameterized Model

Our goal is to generate a *proposed model*, $f_{\zeta_P}(s_t, a_t) = s_{t+1}$, so that it approximates another (fixed) *target model*, $g_{\zeta_T}(s_t, a_t) = s_{t+1}$, where ζ_P and ζ_T are physics parameters of those models. One way to develop this approximation would be to minimize the prediction error (some distance function $d(\cdot, \cdot)$) over a dataset of N examples (Eq. (1)), where L is the length of each example.

$$\arg\min_{\zeta_P} \sum_{n=1}^{N} \sum_{t=1}^{L} d(f_{\zeta_P}(s_{n_t}, a_{n_t}), g_{\zeta_T}(s_{n_t}, a_{n_t})) \tag{1}$$

Unfortunately, we usually do not have access to the true system dynamics g_{ζ_T}. In addition, if the proposed model is an off-the-shelf simulator, it is not differentiable, and so we cannot easily calculate a gradient for Eq. (1). While we can directly estimate the parameters using a neural network (as in [18]), the approach would not generalize well outside the training set, and would be unable to iteratively improve a parameter estimate without additional data since it would encode a single mapping from states to parameter values.

In this work, we instead assume that there exists some set of parameters ζ'_T that will cause the proposed model to closely match the target model, $f_{\zeta'_T} \approx g_{\zeta_T}$, and seek to approximate it. This assumption takes advantage of the fact that f_ζ encodes at least a reasonable approximation of of g_ζ's dynamics, and allows us to use the proposed model as a prior during optimization.

3.2 TuneNet

Our model, TuneNet, estimates the *parameter residual* $\Delta\zeta = \zeta'_T - \zeta_P \approx \zeta_T - \zeta_P$ between a single proposed model and a single target model, and then iteratively updates the proposed model parameters accordingly to match those of the target model. By using a network to estimate parameter deltas, we transform Eq. (1) into a differentiable loss that can easily be used as a supervised learning objective. We can still calculate the quality of the world state estimation, but it does not appear in the loss.

We may be unable to directly observe the state of each model, and represent this with observation functions z_P and z_T applied to proposed and target models, respectively. Observations of length L are generated from each model using the current proposed parameter ζ_P and the true target parameter ζ_T (Eq. (2) and Eq. (3)).

$$o_P = z_P(f_{\zeta_P}(s_t, a_t)), t = 1 \dots L \tag{2}$$

$$o_T = z_T(g_{\zeta_T}(s_t, a_t)), t = 1 \dots L \tag{3}$$

TuneNet estimates the parameter error directly based on the two observations and approximates a function h, which is parameterized by some θ (in this work, the weights of a neural network): $h_\theta(o_P, o_T) = \Delta\zeta$. The new optimization problem for training (over a dataset of length n) is given in Eq. (4), where λ is a weight regularization constant.

$$\arg\min_{\theta} \sum_{n=1}^{N} \|(\zeta_{P_n} + h_\theta(o_{P_n}, o_{T_n})) - \zeta_{T_n}\| + \lambda\|\theta\|^2 \tag{4}$$

Our network consists of one independent fully connected layer for each observation, the outputs of which are concatenated and passed to additional connected layers which estimate the parameter error. We found that concatenating the two intermediate outputs performed better that other combination methods, such as calculating the difference. In this work, we use a physics-based simulator for the proposed dynamics model f_ζ, and the target model g_ζ is either another simulator or the real world.

Algorithm 1 TuneNet's parameter tuning procedure.

```
procedure TUNE($o_T$, simulator $f_\zeta$, initial guess $\zeta_{P_0}$, Network $h_\theta$, $K$)
    for $k = 1 \ldots K$ do
        $o_P \leftarrow z_P(f_{\zeta_{P_{k-1}}}(s_t, a_t)), t = 1 \ldots L$          ▷ Eq. (2)
        $o_T \leftarrow$ RESAMPLE($o_T, L$)
        $\zeta_{P_k} \leftarrow \zeta_{P_{k-1}} + h_\theta(o_P, o_T)$          ▷ Eq. (5)
    return $\zeta_{P_K}$
```

We train TuneNet using a dataset where each datapoint consists of a *pair* of models with a known difference in physics parameters (see Fig. 1). The datasets are auto-generated purely in simulation, which removes the need for hand-labeling or real-world data and makes training faster, safer, and easier. Also, because pairs are generated from the same dynamics model (differing only by ζ), we know that during training, $\zeta_T = \zeta'_T$ and the best-case reality gap during training is 0.

After training TuneNet, we apply it to a set of observations from a new target model and observations from an initial proposed model having parameters ζ_P (Fig. 1). As discussed below in Section 3.3, TuneNet excels at estimating smaller differences in ζ, so we tune iteratively in K steps to achieve the final $\zeta_P = \zeta_{P_{k=K}}$ given a single target observation. The process can be described and implemented using a recursive approach, or an iterative one as listed in Algorithm 1. The RESAMPLE function interpolates the values in o_T, if necessary, so that the length and sampling rate match those of o_P. The two observations are then passed to TuneNet to estimate the parameter difference between them. The algorithm adds the resulting estimate to the previous best guess for ζ_P, bringing it into closer agreement with ζ_T:

$$\zeta_{P_k} = \zeta_{P_{k-1}} + h_\theta(o_{P_{k-1}}, o_T), k = 1 \ldots K \tag{5}$$

Importantly, we are able to do this without collecting any additional observations from the target model, and only sampling the proposed model once per iteration (Eq. (5)). After tuning, the tuned simulator f_{ζ_P} is ready to be used for other tasks such as object motion prediction and robot task planning.

3.3 Residual Tuning

By learning to estimate $\Delta\zeta$ instead of ζ, an estimator naturally becomes better at estimating small deltas due to the nature of the difference transformation. Given uniformly distributed training data over some interval $[d, u]$, the probability densities of parameters ζ_P and ζ_T are equal to some constant $c = \frac{1}{u-d}$ over the interval. However, the difference, $\Delta\zeta$, will be distributed according to $P(\Delta\zeta = x) = -\frac{2}{(u-d)^2}x + \frac{2}{u-d}$, making small delta values more common. As the parameter estimate becomes closer and closer to the true value, the network is able to leverage more and more training data. This behavior allows us to conduct *iterative residual tuning* by collecting observations from the proposed model using successively better ζ_P estimates. Estimating parameter deltas also allows TuneNet to tune to values that lie outside the ranges seen in training data, as opposed to other neural network approaches that often require a transformation or fine-tuning for a new output range.

Residual tuning can be seen as gradient descent in parameter space, using a neural network to approximate the gradient. While gradient-based optimization techniques such as momentum could potentially be combined with this approach, in this paper we show that TuneNet performs well using standard gradient descent.

4 Experiments

We performed three experiments to test that TuneNet can efficiently tune an off-the-shelf simulator to match a target environment, including ones it has not encountered before (Exp. 1 and 2), that it can generate tuned simulations that can be used for object motion prediction (Exp. 2), that it can be used to estimate real-world parameters, even when trained purely in simulation (Exp. 3), and that we can use a simulator tuned with TuneNet as a platform for learning tasks different than the one used

Table 1: Percent and absolute error values on the end effector mass system identification task.

Method	Val Set (Absolute)	Test Set (Absolute)	Test Set (Mean %)
Max of training range (1kg)	–	0.504	28.0%
Mean of test range (1.504kg)	0.25	0.255	17.1%
TuneNet	**0.011**	**0.198**	13.2%

for data collection (Exp. 3). We implement all networks using PyTorch [25]. Network architecture details, layer sizes, and training hyperparmaeters are provided in the appendix.

4.1 Experiment 1: System Identification in Simulation

We first performed a simulated experiment to validate TuneNet's ability to tune one model to match another. The tuned parameter was the mass of an object held in a robot's gripper—a classic system identification objective. To generate training data for the experiment, we use pairs of PyBullet[1] simulations. Each simulation consists of a Kinova Jaco 7-DOF robot arm with a Robotiq 85 2-finger gripper at the end effector, moving in a circle as it holds an object (see appendix for task details). During this motion, we record the 7 raw joint torques over time, and provide this as the input observation signal. To make this task more difficult, in the test set, the *target* simulator has its object mass increased by an additional $1\,\mathrm{kg}$, putting the all correct mass values completely outside the range of values seen during training. This experiment therefore evaluates TuneNet's ability to complete more general tuning tasks, and ones that may take multiple iterations to complete.

After training, we applied TuneNet with $K = 9$ iterations over the validation and test sets to measure the estimation accuracy (K was chosen empirically based on observed tuning behavior). We report the mean absolute error (MAE) of the estimated object mass over both the validation and test sets. We compare TuneNet's performance to several baselines: a prediction of $1\,\mathrm{kg}$ for every data point (which would be the best our network could do if it was unable to tune values outside those seen in the training set) and a constant prediction of the mean parameter value in the *test* set.

On this task, TuneNet achieved an MAE of $\pm 0.011\,\mathrm{kg}$ over the validation set, where both target and proposed masses were were in the training range (see Table 1). When evaluating on the test set, where the target masses were all outside the range used for training, TuneNet still achieved an MAE of $\pm 0.198\,\mathrm{kg}$—higher than on the validation set, but still far better than either baseline of always predicting $1\,\mathrm{kg}$ (MAE=$0.504\,\mathrm{kg}$) or predicting the average test-set value (MAE=$0.255\,\mathrm{kg}$).

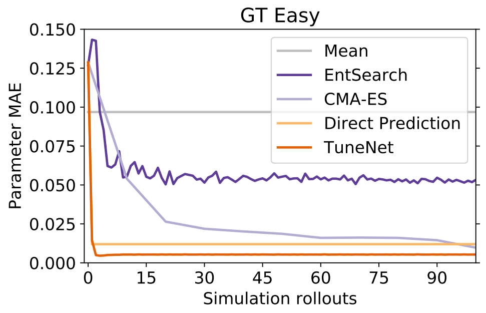
4.2 Experiment 2: Bouncing Ball in Simulation

To test TuneNet's ability to predict rigid body motion, we used it to tune the coefficient of restitution (COR) of a simulated bouncing ball. Our proposed and target dynamics models were again PyBullet simulations. For training, we recreated the simulated dataset used in Ajay et al. [19]: in each episode, a 0.5m-radius sphere is dropped onto a flat plane from a starting height in the range 4-5m, having COR $\sim$ U$(0.3, 0.7)$. At each timestep, we recorded the xyz position of the ball as the world state. We used the same sampling and dataset parameters as [19]—800 simulations for training and 100 for testing, with a physics update rate of 60Hz and episode length of 400 frames. Unlike in [19], each example in our dataset consists of a pair of simulations. In this experiment, the observation functions z_P and z_T were identity, i.e., we had direct access to the world state ground truth. We refer to this case as *GT* (ground truth) in the rest of the paper.

We test over two datasets, GT Easy and GT Hard. GT Easy consists of simulations with proposed and target COR values in the range $[0.3, 0.7]$, the same as in the training set (this is for a more fair comparison with [19]). In GT Hard, the target COR values are extended to the range $[0, 1.0]$.

After training, we evaluated TuneNet by applying $K = 5$ tuning iterations to tune a unique model for each run in the test set. For each run, we report the mean absolute error (MAE) between the final proposed model's COR and the target model's COR. We compare against four baselines: predicting the mean of the *test* set target parameters (*Mean*), our implementation of greedy entropy search [14] (*EntSearch*), the *CMA-ES* optimization method [26], and a neural network that learns to directly predict parameters in one step (*direct prediction*). Details on baseline implementations are in the

[1]https://pybullet.org



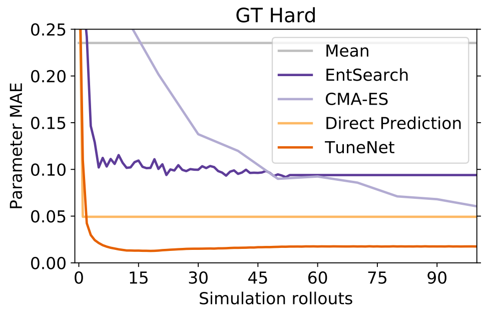


Figure 2: Mean parameter error after N iterations over the simulated ball-bouncing test set for various estimation techniques. TuneNet achieves the lowest error using just a few simulated iterations. Left: GT Easy. Right: GT Hard.

Table 2: Mean absolute error in COR prediction for the simulated ball-bouncing test set at various numbers of simulation rollouts. Best results in bold.

	GT Easy				GT Hard			
Method	$K = 1$	5	10	100	$K = 1$	5	10	100
Mean	0.0968	0.0968	0.0968	0.0968	0.2353	0.2353	0.2353	0.2353
EntSearch	0.1433	0.0622	0.0553	0.0533	0.2876	0.1021	0.1153	0.0939
CMA-ES	0.0540	0.0540	0.0540	0.0097	0.2766	0.2766	0.2766	0.0605
Direct Prediction	**0.0120**	0.0120	0.0120	0.0120	**0.0493**	0.0493	0.0493	0.0493
TuneNet	0.0145	**0.0050**	**0.0053**	**0.0053**	0.1103	**0.0212**	**0.0145**	**0.0175**

appendix. To test our tuned model's overall accuracy in object motion prediction, we record the ball's position from a rollout of the final proposed model, compare it to the target model, and report the MAE and percent error (trans err, cm, and trans err %) over all runs and timesteps.

Fig. 2 compares the average tuning performance (measured by the MAE between the true and estimated parameters) as more and more simulation rollouts are used for estimation. *Direct Prediction* and *Mean* require 0 rollouts, since they are computed directly from the target observation. *CMA-ES* uses 10 rollouts per iteration, and the other algorithms use 1 rollout per iteration. Table 2 quantifies the values in the table for various numbers of simulation rollouts, averaged across the entire training set. TuneNet's one-step prediction accuracy is slightly lower than direct prediction, but it achieves the best accuracy of all methods in less than 5 iterations. In contrast, *EntSearch* has difficulty converging to a good value, and *CMA-ES* requires far more samples to develop a good estimate.

Table 3 shows the object motion prediction error for the bouncing ball dataset. We compare to Ajay et al. [19]'s VRNN hybrid network, although we note that those results are from one-step prediction models, whereas our model uses the entire state history to tune and is reported averaged over the entire dataset. TuneNet GT outperforms all other approaches for position prediction, with an MAE of $7\,\mathrm{mm}$, or 1.08%.

4.3 Experiment 3: Tuning for Sim-to-Real Task Learning

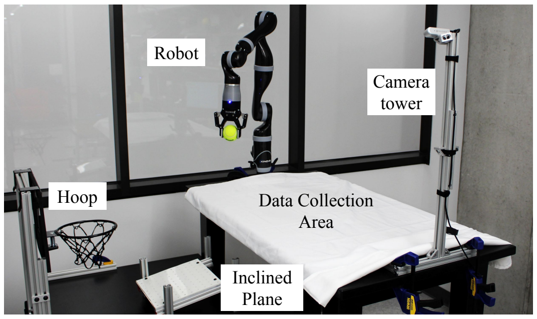
Finally, we tested TuneNet for sim-to-real learning by estimating the COR of a bouncing ball again, but this time in the real world. Our task is to use a tuned simulator to bounce balls off an inclined plane into a hoop (a "bounce shot"), as seen in Fig. 3a. This is a difficult task, as striking the rim of the hoop will often cause the ball to bounce away rather than passing through the hoop.

Our hardware platform for evaluation consists of a Kinova Jaco 7DOF robot arm with a Robotiq 85 2-finger gripper (see Fig. 3a), controlled using ROS[2]. A camera is positioned opposite the robot for perception purposes.

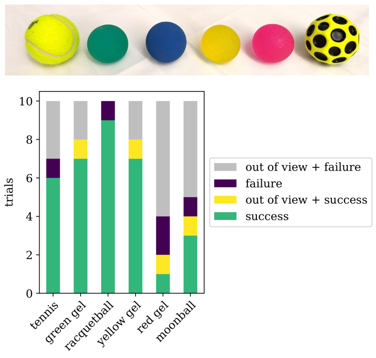
[2]https://ros.org

Table 3: Mean absolute error in object motion prediction for the GT Easy dataset (Exp. 2).

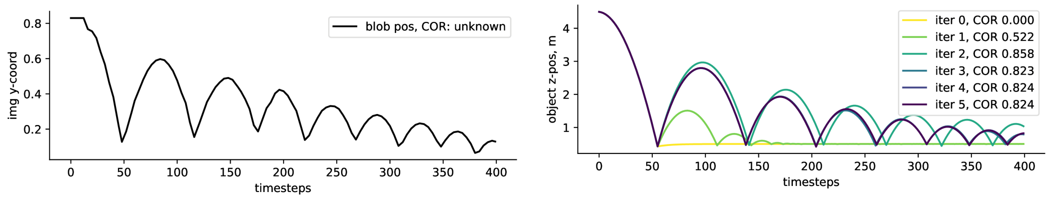
Method	trans err, %	trans err, cm
Zero (inelastic)	100	97.0
Un-Tuned Physics	20.57	13.8
Neural [19]	9.16	5.8
VRNN Hybrid [19]	2.42	1.6
TuneNet (Obs)	6.03	5.7
TuneNet (GT)	**1.08**	**0.7**



(a)



(b)



(c)

Figure 3: Experiment 3 sim-to-real task results. (a): The hardware setup used to collect data and perform tasks. (b): The ball types tested and the success rates for each type. (c) Left: observations from uncalibrated RGB video of robot dropping the *red gel* ball (measured COR: 0.791). Right: bounce trajectories from a simulator using 5 TuneNet iterations to tune the ball's coefficient of restitution (COR), starting from an initial guess of COR=0 (no bouncing).

We trained TuneNet on a visual version of the GT Easy dataset used in Experiment 2. For the *target* simulator in each episode, we rendered a 2D video using PyBullet's renderer, and applied an RGB-only OpenCV object tracker to calculate the position of the ball in each frame. The tuning problem remains challenging after using this object detector, since in addition to the presence of visual noise, 2D pixel coordinates do not directly correspond to z-height in 3 dimensions and our camera images are uncalibrated. The network input for each tuning task is the ground truth 3D position for the proposed model and the 2D (pixel) object position for the target model. This tests TuneNet's ability to learn transformations for different input channels. We refer to this network as TuneNet Obs.

We first tested TuneNet Obs's motion prediction error over the simulated dataset from Experiment 2 (Table 3). It still achieves position accuracy within 6%, and while it performs more poorly than the GT case, this is expected, since the network is operating on noisy, uncalibrated observations and the other approaches in the table have access to the true simulator state.

For the bounce shot task, we performed 10 independent tuning trials for 6 different ball types. In each trial, the robot first dropped the ball on a flat table and recorded a video of the resulting bounces. The video was resampled to 60 Hz, clipped to 400 frames, and processed using the same OpenCV object tracker as on the training set. TuneNet operated on the observation from one real-world drop and the ground truth state from an un-tuned simulator which started with a COR of 0 (no bounce). We used TuneNet ($K = 5$) to tune the simulated COR, clamping to the range $[0, 1]$, as values outside this range are physically meaningless.

Fig. 3c shows an example of TuneNet using $K = 5$ iterative tuning rounds to match observations from a video of a dropped racquetball, beginning with a COR of 0. The first parameter updates are large, and successive iterations fine-tune the estimate because of the more accurate tuning behavior for small parameter deltas (see Section 3.3).

The tuned COR value was used to construct a new simulator, consisting of a dropped ball, inclined plane, and hoop that match our real-world task setup. The simulator was used to uniformly sample 100 shot heights from a range within the robot's reachable space, and the optimal height was chosen as the shot height that passed the ball as close to the center of the hoop as possible. The real robot then executes the shot from the optimal height. We repeat the task for all 10 independent trials and all 6 ball types (not all of which are round, see Fig. 3) and record the success or failure of the bounce shot.

Fig. 3 shows the results of the bounce shot task and a representative execution. Overall, the robot succeeded at 62% of the bounce shots. The primary reason for failure was that during the pre-tuning data collection step, the ball bounced or rolled off the table or out of view of the camera. This resulted in a meaningless input observation, which caused in a low estimated COR and incorrect drop height. Indeed, when these "off table" trials are removed, the success rate increases to 87%, whereas the "off table" trials were only 18% successful, as shown in Fig. 3b. Eliminating these examples where the ball bounced off the table in the observation drop, two of the six ball types were successful in every trial. Successful shots with all 6 balls can be found in the supplementary video.

5 Discussion and Conclusion

TuneNet is **fast**. Since each tuning iteration consists of a single network forward pass and a single simulation rollout, TuneNet reduces the number of parameter-space simulation samples required compared to existing gradient-free optimization, and it tunes significantly faster than techniques that alternate between domain randomization in simulation and real-world data collection [14, 2]. Our dataset generation and training procedure is also fast due to being trained entirely in simulation, and completes in under 7 minutes using a single NVIDIA GTX 1080Ti.

TuneNet is **effective**. Our results show that even though a simulation cannot perfectly model the real world, much of the reality gap can be closed by choosing good simulation parameters. We note that TuneNet is compatible with recent approaches that learn data-driven transformations or residuals to improve simulations [20, 27, 4, 5], and envision a two-step approach where TuneNet first matches a real-world environment as closely as possible using parameter tuning alone, then additional unmodeled effects are accounted for using learned transformations or domain randomization.

That said, there are some limitations of our approach that suggest avenues for future work. For example, parameters may not be fully identifiable based on observations alone, as explored in several studies [28, 29, 30]. As discussed in [30] and [12], the *identifiable set* of parameters may be coupled, which would result in multiple local minima in parameter space. This may prove to be an issue if the goal is to accurately measure system parameters, but as long as the tuned parameters enable accurate prediction and sim-to-real learning using the proposed model, which local minimum the algorithm converges to is less important. For example, in our study, the *racquetball* had high task success rates despite a technically inaccurate COR estimate (0.789 ± 0.024 vs. regulation values of 0.68–0.72 [31]). [32] is also concurrently studying the problem of modeling belief over coupled parameter values.

In this paper, we introduced TuneNet, a neural network-based technique to perform residual tuning between two models, and validated the approach for use in system identification and sim-to-real robotics tasks. TuneNet represents a new method for closing the reality gap, showing that a simulator can be matched to the real world with just one observation and minimal simulation sampling. We plan to use TuneNet to rapidly generate tuned simulators for new environments, developing strong models for object manipulation without extensive real-world practice.

Acknowledgments

We thank Josiah Hanna and Scott Niekum for their helpful suggestions. This work was supported by the US NSF (IIS-1564080, IIS-1724157) and US ONR (N000141612835, N000141612785).

References

[1] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual Reinforcement Learning for Robot Control. *CoRR*, abs/1812.0, 2018.

[2] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience. *CoRR*, Oct. 2018.

[3] J. P. Hanna and P. Stone. Grounded Action Transformation for Robot Learning in Simulation. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 3834–3840, Feb. 2017.

[4] F. Golemo, A. A. Taïga, P.-Y. Oudeyer, and A. Courville. Sim-to-Real Transfer with Neural-Augmented Robot Simulation. In *2nd Conference on Robot Learning (CoRL18)*, volume 87, pages 817–828, 2018.

[5] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. A. Funkhouser. TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. In *Proceedings of Robotics: Science and Systems*, Freiburg im Breisgau, Germany, June 2019.

[6] T. Silver, K. Allen, J. Tenenbaum, and L. P. Kaelbling. Residual Policy Learning. *CoRR*, abs/1812.0, 2018.

[7] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-Real Robot Learning from Pixels with Progressive Nets. In *Conference on Robot Learning*, pages 262–270, 2017.

[8] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE International Conference on Intelligent Robots and Systems*, pages 23–30. IEEE, Sept. 2017.

[9] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. In *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, June 2018. ISBN 1804.10332v2.

[10] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. EPOpt: Learning Robust Neural Network Policies Using Model Ensembles. *CoRR*, 2016. ISSN 0027-8424.

[11] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric Actor Critic for Image-Based Robot Learning. In *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, June 2017.

[12] P. Khosla and T. Kanade. Parameter identification of robot dynamics. In *1985 24th IEEE Conference on Decision and Control*, pages 1754–1760. IEEE, Dec. 1985. ISBN 9684641354.

[13] M. Gautier and W. Khalil. On the identification of the inertial parameters of robots. In *Proceedings of the 27th IEEE Conference on Decision and Control*, pages 2264–2269. IEEE, 1988. ISBN 0-8186-0852-8.

[14] S. Zhu, A. Kimmel, K. E. Bekris, and A. Boularias. Fast Model Identification via Physics Engines for Data-efficient Policy Search. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3249–3256, 2018.

[15] J. Tan, Z. Xie, B. Boots, and C. K. Liu. Simulation-based design of dynamic controllers for humanoid balancing. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2729–2736. IEEE, Oct. 2016.

[16] S. Zhu, D. Surovik, K. E. Bekris, and A. Boularias. Closing the Reality Gap of Robotic Simulators through Task-oriented Bayesian Optimization. *Journal of Machine Learning Research*, 2019.

[17] S. Kolev and E. Todorov. Physically consistent state estimation and system identification for contacts. In *IEEE-RAS International Conference on Humanoid Robots*, volume 2015-Decem, pages 1036–1043. IEEE, Nov. 2015. ISBN 9781479968855.

[18] W. Yu, J. Tan, C. K. Liu, and G. Turk. Preparing for the Unknown: Learning a Universal Policy with Online System Identification. In *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[19] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez. Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing. In *International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[20] A. Kloss, S. Schaal, and J. Bohg. Combining learned and analytical models for predicting action effects. *CoRR*, 2017.

[21] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[22] J. Zhang, L. Tai, P. Yun, Y. Xiong, M. Liu, J. Boedecker, and W. Burgard. Vr-goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, 4(2):1148–1155, 2019.

[23] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell. CyCADA: Cycle-Consistent Adversarial Domain Adaptation. *CoRR*, abs/1711.0, 2017.

[24] Z. Xu, J. Wu, A. Zeng, J. Tenenbaum, and S. Song. DensePhysNet: Learning Dense Physical Object Representations Via Multi-Step Dynamic Interactions. In *Proceedings of Robotics: Science and Systems*, Freiburg im Breisgau, Germany, June 2019.

[25] A. Paszke, G. Chanan, Z. Lin, S. Gross, E. Yang, L. Antiga, and Z. Devito. Automatic differentiation in PyTorch. In *31st Conference on Neural Information Processing Systems (NIPS)*, pages 1–4, Long Beach, CA, 2017. ISBN 9788578110796.

[26] N. Hansen. The CMA Evolution Strategy: A Tutorial. *CoRR*, abs/1604.0, 2016.

[27] J. P. Hanna, P. S. Thomas, P. Stone, and S. Niekum. Data-efficient policy evaluation through behavior policy search. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1394–1403. JMLR.org, 2017.

[28] K. Ayusawa, G. Venture, and Y. Nakamura. Identifiability and identification of inertial parameters using the underactuated base-link dynamics for legged multibody systems. *The International Journal of Robotics Research*, 33:446–468, 2013.

[29] D. Ma and A. Rodriguez. Friction Variability in Auto-collected Dataset of Planar Pushing Experiments and Anisotropic Friction. *IEEE Robotics and Automation Letters*, 3(4):3232–3239, 2018. ISSN 2377-3766.

[30] N. Fazeli, R. Tedrake, and A. Rodriguez. Identifiability Analysis of Planar Rigid-Body Frictional Contact. In *International Symposium on Robotics Research (ISRR)*, page To appear. Springer, 2015. ISBN 9781424479740.

[31] O. E. Dietrich. *USA Racquetball Official Rules of Racquetball*. USA Racquetball, 2015.

[32] F. Ramos, R. C. Possas, and D. Fox. BayesSim: Adaptive Domain Randomization Via Probabilistic Inference for Robotics Simulators. In *Proceedings of Robotics: Science and Systems*, Freiburg im Breisgau, Germany, June 2019.

Appendix

TuneNet Architecture Implementation Details

We implement all networks using PyTorch [25], and train them using stochastic gradient descent. Our TuneNet architecture for all experiments consists of fully-connected (FC) layers with ReLU activation functions for each of the feature extractors, with each feature extractor output size set to 32. The estimator therefore has an input size of 64. We model the estimator with two fully connected layers with a hidden size of 32, ReLU activation for the first layer, and a *tanh* activation for the final output.

Details for Experiment 1: End-effector Mass Identification

The object's mass is randomly chosen, $m \sim U(0\,\mathrm{kg}, 1\,\mathrm{kg})$, for each training point. In each simulated episode, which runs for 5 seconds, the robot moves its end effector in a circle defined by the coordinates $[-0.4, 0.2\cos(\tau), 0.2\sin(\tau)], \tau = 1.2t$. We collect 1000, 300, and 300 episodes for the training, validation, and test sets, respectively. We trained the model for 1000 epochs using a batch size of 50, a learning rate of 0.01, L2 regularization $\lambda = 1\mathrm{e}{-3}$, and 1% learning rate decay every 50 epochs. During training, the input torques were normalized to the range $[0, 1]$ but the outputs were not transformed.

Details for Experiment 2: Bouncing Ball COR Tuning

For both datasets, we trained TuneNet for 200 epochs, using a batch size of 50 and a learning rate of $1\mathrm{e}{-2}$, with learning rate decay of 1% per 5 epochs. The L2 normalization constant λ is set to 0.01. During training, the input and output channels were normalized to the range $[0, 1]$.

For the Obs case, the visual OpenCV tracker uses hue, saturation, and value windows to threshold a camera image and isolate an object. It then calculates the y pixel position of the ball in each frame and normalizes the image pixel positions to the range $[0, 1]$. For each randomized simulation render, the PyBullet camera was placed at random polar coordinates $r \sim \mathrm{U}(5\,\mathrm{m}, 10\,\mathrm{m})$, $\theta \sim \mathrm{U}(0, 2\pi)$, $z \sim \mathrm{U}(5\,\mathrm{m}, 8\,\mathrm{m})$, and aimed at the point $(0, 0, 2\,\mathrm{m})$.

Baseline Implementation Details

For CMA-ES, we use the open-source package `pycma`[3]. The inputs to CMA-ES are identical to those provided to TuneNet, with the initial proposed physics parameters being used as the initial guess for the CMA algorithm. The CMA tolerance was set to 0.1.

For Greedy Entropy Search (*EntSearch* in this paper), we re-implemented the algorithm provided in [14]. The underlying Gaussian Process (GP) sampling was implemented using the open-source Python package `GPy`[4], and we used an RBF GP kernel. Again, the initial proposed physics parameters were used as the initial guess for the algorithm. We discretize the parameter space into 50 bins, and use a gaussian process sampling population size of $n = 100$. Because we do not calculate the value function for each policy, we cannot use the value as the stopping criterion as in Zhu et al. [14]. Instead, we stop iterating when the predicted parameter value does not change for two consecutive timesteps. This stopping parameter was determined empirically for our experiment after being shown to outperform two other stopping criteria: no stopping (which performed poorly because of noise in the GP sampling), and stopping after the parameter estimate changed by less than a threshold $\epsilon > 0$.

The initial guesses and simulated outcomes supplied to both CMA-ES and Greedy Entropy Search were identical to the inputs supplied to TuneNet, including all normalization transformations applied to the data.

[3] `https://github.com/CMA-ES/pycma`
[4] `http://github.com/SheffieldML/GPy`