Proceedings of the ASME 2020
International Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
IDETC/CIE 2020
August 16-19, 2020, St. Louis, MO, USA

18260

MULTIPARAMETER REAL-WORLD SYSTEM IDENTIFICATION USING ITERATIVE RESIDUAL TUNING

Adam Allevato*
The University of Texas at Austin
Walker Dept. of Mech. Engr.
Austin, TX 78712
allevato@utexas.edu

Mitch Pryor
The University of Texas at Austin
Walker Dept. of Mech. Engr.
Austin, TX 78712

Andrea L. Thomaz
The University of Texas at Austin
Dept. of Elec. Engr.
Austin, TX 78712

ABSTRACT

In this work we consider the problem of nonlinear system identification, using data to learn multiple and often coupled parameters that allow a simulator to more accurately model a physical system and close the so-called reality gap for more accurate robot control. Our approach uses iterative residual tuning (IRT), a recently-developed derivative-free system identification technique that uses neural networks and visual observation to estimate parameter differences between a proposed model and a target model. We develop several modifications to the basic IRT approach and apply it to the system identification of a 5-parameter model of a marble rolling in a robot-controlled labyrinth game mechanism. We validate our technique both in simulation— where we outperform two baselines—and on a real system, where we achieve marble tracking error of 4.02% after just 5 iterations of optimization.

1 INTRODUCTION

A robot operating a complex system or under unknown dynamics can greatly benefit from an accurate model or simulation of its environment. Many modern robot and system control techniques such as model-predictive control [1] require a model. Even for model-free techniques, a simulated model can be useful for safely learning new robot policies using techniques evolutionary strategies [2, 3] or reinforcement learning [4]. Accurate simulations can also be useful as an aid for teleoperation or as predictive tool, as in the case of *digital twins* [5], which are simulated surrogates of complex systems. In all these instances, simulation accuracy is important, since inaccurate models can harm task performance and even small errors can often compound over time. The difference between the simulated and real response to a given input is known as the *reality gap*. While several works have studied the reality gap [6–10], narrowing or closing it remains a significant challenge. System identification, which learns a model that matches measurements of the system state, can help close the reality gap significantly by simply choosing appropriate parameter values for a simulated system and robotics researchers have recently taken more interest in the topic [11, 12]. A new technique called *iterative residual tuning*, or IRT, uses a neural network and simulated pretraining to perform efficient and accurate parameter estimation from minimal real-world observation (measurement). In prior work, IRT was shown to work on both simulated and real-world system identification problems, but only considered the problem of tuning a model with a single parameter, whereas most models have multiple (and likely coupled) unknown parameters that must be tuned.

In this work, we use IRT to perform system identification and generate a simulated version of a physical mechanism manipulated by a robot: a marble rolling freely in a wooden labyrinth game. We tune five different physical parameters in the simulator, many of which would be difficult to measure directly on the physical system. We perform one experiment en-

*Address all correspondence to this author.

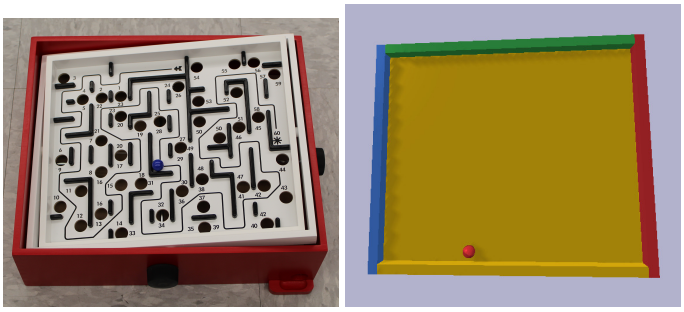

FIGURE 1: Left: the physical labyrinth used in this study. Right: our simulated surrogate labyrinth used to identify multiple system parameters.

tirely in simulation to provide a detailed look at parameter tuning performance, and another to tune the simulator to match the physical system. We also make several adjustments to the base IRT method that allow it to perform well in the more difficult multiparameter case, including additional randomized data for pretraining, parameter scaling to normalize network inputs, and decaying-rate gradient descent to mitigate noisy gradient estimation.

We demonstrate that IRT can successfully tune multiple simulation parameters using real-world data and outperforms three competing approaches, including an evolutionary strategy approach used in prior work [10] to characterize the same mechanism. Our method outperforms competing approaches, achieving a lower mean absolute state tracking error when modeling the real system as well as the lowest observed parameter error in the simulation experiment.

2 RELATED WORK

Our work builds on developments in the field of system identification and dynamics model learning.

2.1 System Identification

System identification [13, 14] is the process of selecting a model to fit observed data from a dynamical system. It is one way of narrowing the reality gap, and has usually been performed via maximum likelihood techniques, such as least-squares fitting [13, 15, 16]. This approach becomes less useful as the modeled system becomes more complex and especially when the input-output relationship is nonlinear. These statistical techniques can also require large data sets to ensure an accurate estimate of the system parameters. Newer system identification techniques have considered data-driven optimization techniques that do not represent the relationship between inputs and outputs in a closed form. They have shown that in this problem formulation, system parameters can be directly estimated by neural networks [12] or by using global optimization techniques such as entropy-based search [6, 11]. Another line of research has developed a pseudo-maximum likelihood estimator for nonlinear system identification based on Monte Carlo simulations [17], although they do not evaluate their approach on a physical system. Finally, IRT [18], described in more detail below, is a recently-developed gradient-free system identification technique which we use and extend in this work.

2.2 Learning Dynamics Models

Our work is also similar to related research in learning models of dynamic and articulated objects using machine learning. Several researchers have predicted the motion and physical parameters of objects using observed interaction data and techniques such as neural networks [19–22], Gaussian processes [23], and linear regression [24]. These approaches are generally not data-efficient, as they encode no notion of object dynamics and must learn the laws of physics from scratch. Other approaches, including IRT, combine neural networks with simulation [25–27] to take advantage of both a flexible learned representation and explicit physics prediction.

Researchers have studied the control and modeling of the same labyrinth game considered in this work [28], where one particular work specifically considered the problem of closing the reality gap for this system [10]. However, it does not consider the modeling of the system itself but rather optimizes the PID parameters of servo motors which turn the labyrinth's control knobs; the other parameters of the simulation were hand-tuned. This work also used custom instrumentation to allow absolute measurement of the control knobs' position in relation to the commanded position, simplifying the system identification problem, and tuned only one motor at a time, whereas we use a technique which allows us to learn parameter values for the entire system directly from the resulting motion of the marble in the labyrinth.

3 PROBLEM STATEMENT

The goal of this effort is to optimize the parameters of a simulated version of a wooden labyrinth game operated by a robot so that the simulated behavior better matches that observed in the real world. We do not instrument the board itself in any way, and tune using only visual observations of the system during interaction. Mathematically, we seek a vector of parameters ζ which minimizes the mean state error (averaged over all state variables) between the real system $f_{\zeta_{\text{true}}}(\mathbf{x}, u)$ and the simulated system $g_{\zeta}(\mathbf{x}, u)$ as measured over τ timesteps:

$$\arg\min_{\zeta \in Z} \frac{1}{\tau} \sum_{t=1}^{\tau} \left| f_{\zeta_{\text{true}}}(\mathbf{x_{t+1}}, u_{t-1}) - g_{\zeta}(\mathbf{x_{t+1}}, u_{t+1}) \right| \qquad (1)$$

In short, we wish to close the reality gap as much as possible. Furthermore, we wish to perform this minimization with as little data from the real system as possible.

3.1 The Labyrinth Game

The labyrinth (Fig. 1) is a dexterity-based game where the player must guide a marble through a series of maze-like walls without it falling into one of the marble-sized holes in the game's surface. The player guides the marble by turning two knobs, which tilt the surface along the X and Y axes via an internal pulley mechanism. In this study, we consider a labyrinth model sold by the Brio company. This particular model is functionally identical to the one used in prior studies [10].

In our case, the "player" is two Kinova Jaco2 7 degree-of-freedom robot arms with Robotiq 85 2-finger grippers, as shown in Fig. 2. Two important skills for completing the labyrinth are the ability to understand how the marble moves in response to the surface's tilt, and the dexterity to turn the knobs correctly to obtain a desired motion. In software, we provide motion prediction by using an off-the-shelf physics engine, and we learn dexterity by using system identification to match the physics engine to the real system. In this study, we are interested only in observing marble motion and using it to model the overall system parameters. Therefore, we replace the game surface with a smooth flat plane, allowing the marble to roll anywhere in the labyrinth freely. This allows us to focus on the system identification problem, rather than the control and planning problem of avoiding holes and walls in the labyrinth, while also keeping in mind the future development of control strategies and game-playing policies.

In order to build an accurate simulation of the labyrinth, we must estimate several different parameters. Some parameters, such as the size of the labyrinth board and the mass of the marble, are easy to measure directly. However, other important parameters are more difficult to measure:

1. The internal pulley mechanism, which is imprecise and contains a spring (to ensure tension) results in an **effective gear reduction** between the input knobs and the board angle, which is difficult to measure precisely without instrumentation such as rotary encoders. The gear reduction also differs for each knob.
2. The **coefficient of rolling friction** of the marble is extremely important, since it determines the angle that cause marble motion. It is also difficult to measure without a special experimental apparatus. Even if the correct value could be measured, because of how contact dynamics are implemented in physics engines [29], using the true friction value may not actually provide realistic behavior.
3. The software system used to drive the robots which operate the labyrinth causes a significant control lag, where each input command takes hundreds of milliseconds to be applied to the actual system. This **control lag** should also be represented to achieve simulator accuracy and represents another tunable parameter.
4. Because we tune using visual observations rather than directly measured states, we must manage sensing errors. In particular, the exact position of the marble is difficult to measure in 3D without knowing the angle of the labyrinth surface (which is in turn dependent on the effective gear reduction another unknown parameter). Measuring the 2D position of the marble (in the image plane) will present issues due to parallax—as the table tilts, the marble's 2D position will change, and this change also depends on the effective gear reduction. As described below, this discrepancy can be managed by adding another learnable parameter to the model: **observation scaling**.

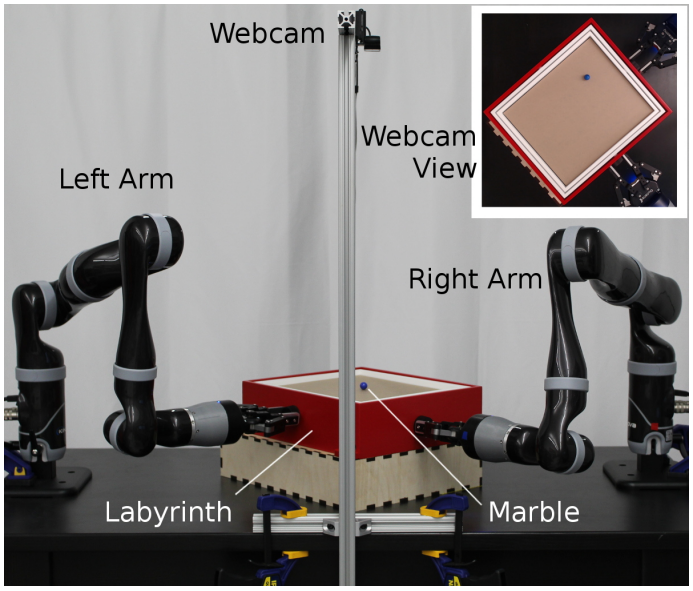

FIGURE 2: Our experimental setup.

4 THEORY: ITERATIVE RESIDUAL TUNING

To estimate the tunable simulator parameters listed above, we use the IRT system identification technique [18]. The process begins with choosing a set of proposed parameters ζ_P and using them to construct a *proposed model* (i.e., an untuned simulator). IRT generates a time series of observations o_P from the proposed model and compares them with observations o_T generated from the *target model*, which could be another simulator (for evaluation purposes) or the physical system we are seeking to identify. The observations are assumed to be a function of the underlying system state and the model parameters, and are used to estimate the difference in parameters between the two models. The parameter difference $\zeta_T - \zeta_P$ is estimated by a neural

network $h_\theta(o_{P_n}, o_{T_n}) = \tilde{\Delta}\zeta_k \approx \zeta_T - \zeta_P$, which is pre-trained on a large dataset of pairs of simulation for which the difference in parameters is known. The estimated difference is added to the current proposed parameters so that they better match the target parameters, and the new proposed model can be used to generate a new set of observations for additional tuning in another iteration. This procedure can be repeated until the proposed parameters ζ_P converge (ideally to a value very close to the target parameters ζ_T.

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

In prior work, the authors introduce a neural network called TuneNet [18] to perform this estimation, and train it using stochastic gradient descent. The λ term in Eq. (2) is a regularization constant used to prevent the network from overfitting.

In IRT, the parameter gradient, or the difference in parameters with respect to the observations, is estimated entirely from data rather than calculated in a closed-form fashion. This makes IRT a **derivative-free** optimization method that is both data-efficient and does not require a closed-form expression for the system being optimized.

In previous work, IRT was shown to outperform two other derivative-free optimization methods, CMA-ES and Entropy Search, in terms of accuracy and data efficiency. However, these results only considered the problem of tuning a model with a single parameter. Adding more dimensions to the parameter estimation problem increases the size of the search space exponentially.

5 APPROACH

To use IRT to estimate labyrinth parameters, we produced an (untuned) simulation of the labyrinth to use as the proposed model and made modifications to the IRT framework.

5.1 Simulation

The simulator we used for both experiments was a custom-designed virtual representation of the labyrinth written using the PyBullet simulation library (see Fig. 1. As described above, the size of the labyrinth board, the marble radius, and the marble mass were determined via direct measurement and held fixed, and the other parameters were configurable for each simulated run. We did not model the exterior or the interior tilt mechanisms of the labyrinth but instead rotate the simulated board surface directly in accordance with the input and gear reductions. For visual tracking, we use a slightly larger colored marble (diameter 13.7mm) instead of the one supplied with the labyrinth.

5.2 Inputs, Observations, and Parameters

We learn to estimate parameter differences using a neural network similar to TuneNet. We pretrained the network using 10000 pairs of simulated samples, each of which have a randomized set of parameters.

The five tunable parameters discussed above are defined and given range limits below.

1. Effective Gear Reduction (X-Axis) p_x: Given some control knob angle θ_x (expressed in radians), the board angle was calculated as $\theta_{board_x} = \theta_x * p_x$. Range: [0, 0.1], unitless. This small range was chosen based on empirical observation—the pulleys are designed so that a large change in control knob angle causes on a small change in the overall board angle.
2. Effective Gear Reduction (Y-Axis) p_y: As above, although note that the gear reduction need not be the same for each axis due to the internal geometry of the pulley mechanism. Range: [0, 0.1], unitless, determined as above.
3. Marble Rolling Friction μ_{roll}: the rolling friction coefficient of the marble rigid body. Set directly and handled internally by the physics simulation. Range: [0, 0.001], unitless, selected empirically based on rolling friction values often used in literature and software documentation.
4. Time scale p_t: after each timestep, the simulation is run for round(p_t) physics updates, each representing 1/240s = 4.17ms of simulated time. This parameter is inserted to allow for "time warping" within the simulation, which may be necessary to accurately match how actions are carried out on the physical system. Range: [0, 50] timesteps = [0, 208]ms. Chosen based on assumed 5Hz minimum control rate for the real system.
5. Observation scale p_o: To account for slight discrepancies between the real and simulated labyrinth sizes, and any observation differences in the real world due to parallax, the ground truth observation was scaled by a small amount. Range: [0.8, 1.2], unitless, chosen empirically.

For each simulated sample, the input to the system at each timestep was the control knob angle. For each sample, we conducted a 100-timestep *rollout*. During each rollout, we moved both control knobs throughout a full period of sinusoidal motion motion according to Eq. (3), where t is the current timestep and θ_x and θ_y are the rotation angles in radians about the x and y axes respectively.

$$\theta_x = 0.1\sin(\frac{t}{200\pi}); \theta_y = 0.1\cos(\frac{t}{200\pi}) \qquad (3)$$

At each timestep, the two-dimensional observation from the simulator was the ground truth position of the marble. This position was normalized to the range [-1, 1] by dividing by the edge length of the board, as well as multiplied by the observation scale

factor (see below). This 200-dimensional observation (X and Y marble positions at each of the 100 timesteps) are passed into the tuning network, which generates a 5-dimensional output representing the estimated difference for each of the 5 model parameters.

5.3 Training Modifications

We made a number of key modifications to allow IRT to perform well on this more difficult problem.

In the original IRT work, the procedure was used to tune parameters (coefficient of restitution and the mass of a block) which naturally ranged from 0 to 1. In our problem, the different parameters have significantly different magnitudes, so before passing them to the neural network, we normalize all of them from the ranges listed above to the range [0, 1] to make training easier and to not incentivize the network to tune some parameters more aggressively than others. When the parameters are supplied to the simulator, they are re-transformed back to the ranges listed above.

During tuning, a poor initial proposed parameter could cause the tuning process to diverge. To prevent this, we clipped the possible parameters to the ranges listed above. We also added a step size decay, $\beta = 0.85$, to the parameter update at each step to improve convergence at later tuning iterations: $\zeta_{p_{k+1}} = \zeta_{p_k} + \beta^k \tilde{\Delta}\zeta_k$. We found that this change had no adverse effect for purely simulated experiments, but was very important when tuning to match the real system, where the estimated gradient had a large amount of noise.

Our network architecture was identical to TuneNet, except that it used a larger hidden layer size of 256. Finally, we observed that using an Adam optimizer (learning rate 3e-4) performed better than stochastic gradient descent for this problem, and also allowed us to remove the regularization coefficient λ from the loss function.

6 EXPERIMENTS

We performed two experiments to validate IRT for this system. The first experiment, which was entirely simulated, measured the parameter tuning accuracy directly. The second experiment tuned a simulator to match the real world and measured the accuracy of the predicted marble state.

Both experiments used the same trained neural network and parameter update procedure described above.

6.1 Experiment 1: Simulated Validation

In this experiment, we tested the ability for IRT to tune one simulation to match another. Using simulations for both the proposed and target models allowed us to measure the parameter estimation error, since we have access to ground truth for both models.

We generated a test dataset of 10 randomly sampled parameter values, which were used to produce 10 target models and associated observations using the same sinusoidal inputs described above. For each of the 10 target models, we selected a random proposed model from the same parameter space and attempted to tune this model to match the target model.

We compare our IRT implementation to two other baselines. The first baseline, "direct prediction," is a neural network trained to predict parameters from the target observation alone. We constructed this baseline using a neural network that takes only one set of observations as input but is otherwise identical to the network we used for IRT, and training over the same dataset as the full network.

The second baseline is covariance matrix adaptation evolution strategy, or CMA-ES [3]. This is a gradient-free optimization technique, and is an updated and improved version of original (5+100) ES optimization used to characterize this labyrinth system in prior work [10]. CMA-ES has the benefit of having few hyperparameters to tune, but requires λ function evaluations on each iteration for some fixed population size λ (set to 10 in this experiment). We use the negative mean absolute state error between the target and proposed models as the fitness function (optimization objective) for CMA-ES, and use the initial proposed model for IRT as the CMA-ES initial guess.

We report two metrics for this experiment. First, we calculate $|\zeta_{P_k} - \zeta_T|$, the absolute error between the proposed and target parameters averaged across all parameters and timesteps $1 \dots \tau$, for each optimization iteration. More efficient methods will more quickly reduce the error rate. We also report the mean absolute observation error between the proposed and target models at each iteration, $\text{MAE}_\text{o} = |o_P - o_T|$.

In general, IRT performed well on this problem. Fig. 3 shows an example of IRT tuning two proposed simulations with different starting parameter values to match a target simulation. The untuned case (iteration 0, first row) shows the initial discrepancy between the observations. The first IRT iteration improves the behavior considerably, although the optimization is still unstable as shown in the next iteration. After 20 iterations, both proposed models approximate the dynamics and maximum magnitudes quite well.

Fig. 4 and Fig. 5 show the parameter and state absolute error, respectively, as a function of the number of proposed simulations evaluated during optimization. The direct prediction method fails to learn an accurate predictor and results in decreased performance compared to the initial proposed guess. This is likely due to the very large search space as well as the lack of data provided by a proposed simulation. CMA-ES is able to make modest progress in reducing both state and parameter error, but doing so requires a large number of simulations. IRT, on the other hand, is much more efficient, optimizing parameters to the best of its ability (around 10% mean error averaged across all timesteps) almost immediately. Note that each iteration of IRT need not be

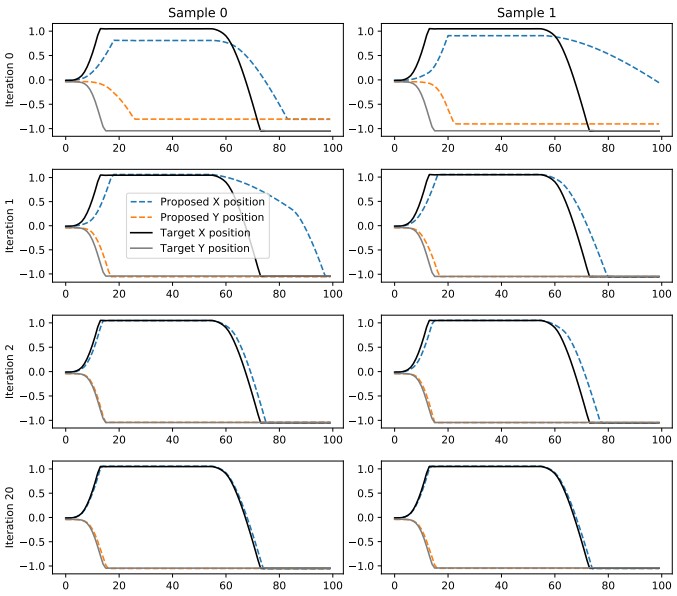

FIGURE 3: Observations for proposed and target models during tuning in Experiment 1 (sim-to-sim). Each column represents a different initial proposed model and each row represents a different tuning iteration. Time is plotted on the X axis and marble position (normalized to the range [-1, 1] as described in the text) on the Y axis.

strictly better than previous iterations, as seen in Fig. 5.

Fig. 6 shows how individual parameters converge to a value over several iterations of IRT. Some parameters overshoot their final value, but eventually, all trials converged to the same optimum.

6.2 Experiment 2: Real-World System Identification

In the next experiment, we used IRT to tune a simulator to match our physical experimental setup (Fig. 2), where two robots interact with the labyrinth.

On this system, due to delay and lag in the controller, we did not choose a control rate but instead executed motions as quickly as our controller would allow (this is why the time scale factor is necessary). We applied the same sinusoidal motions as in the simulated environment and observe the resulting marble behavior. The control knob position commands were sent to the robots' 7th joints (wrists) using the Robot Operating System (ROS) [30] and executed by the robots' internal controllers. Visual observations were collected using Logitech HD webcam mounted above the labyrinth, and we used thresholding operations and contour detection provided by OpenCV [31] to extract the marble position as the observation at each timestep. This observation was then used as o_T. The initial proposed parameters ζ_{p0} were chosen at random from the overall parameter space.

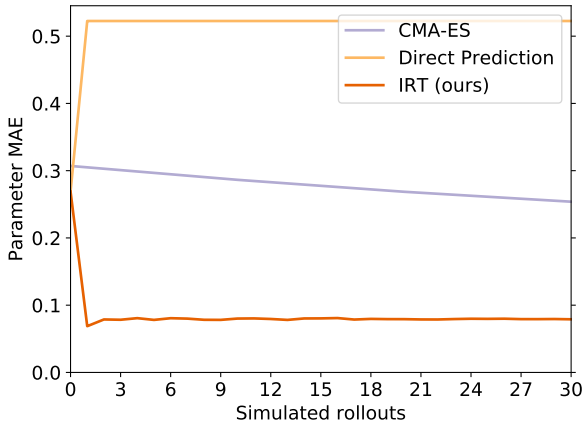

FIGURE 4: Mean absolute parameter error (averaged over 10 trials) for Experiment 1. In this and the other error plots, the Y axis units are normalized marble position units: an error of 1 equals the size of the labyrinth board.

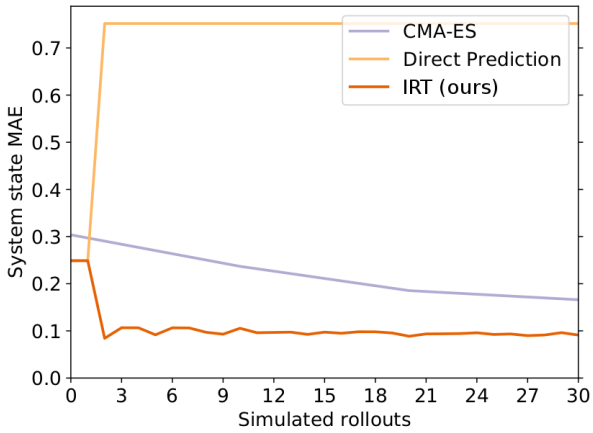

FIGURE 5: Mean absolute state error (averaged over 10 trials) for Experiment 1 expressed in normalized board coordinates (see Fig. 4 for explanation).

For this experiment, we report observation error as in the previous experiment, but cannot report parameter error as we do not know the correct values for the physical system. We compare against the same two baselines.

Overall, this task was significantly harder than the sim-to-sim tuning task. The main issue lies in the fundamental differences in model type. In the case of sim-to-sim, the observations are generated from two models that are identical except for the parameterization, and so it is theoretically possible to achieve zero state estimation error. This cannot be said for the real system, which also includes noise in the sensor observations and dy-

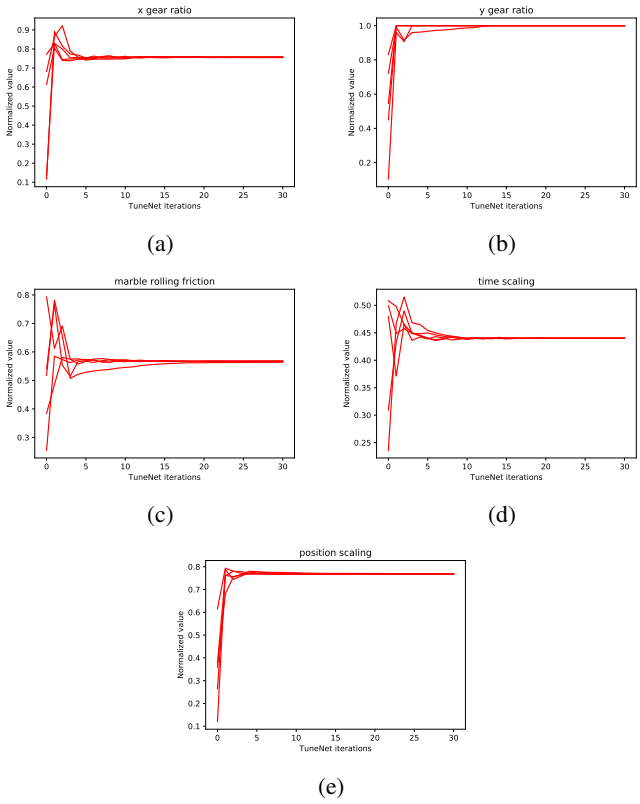

FIGURE 6: Parameter evolution curves for IRT in Experiment 1 (10 traces shown).

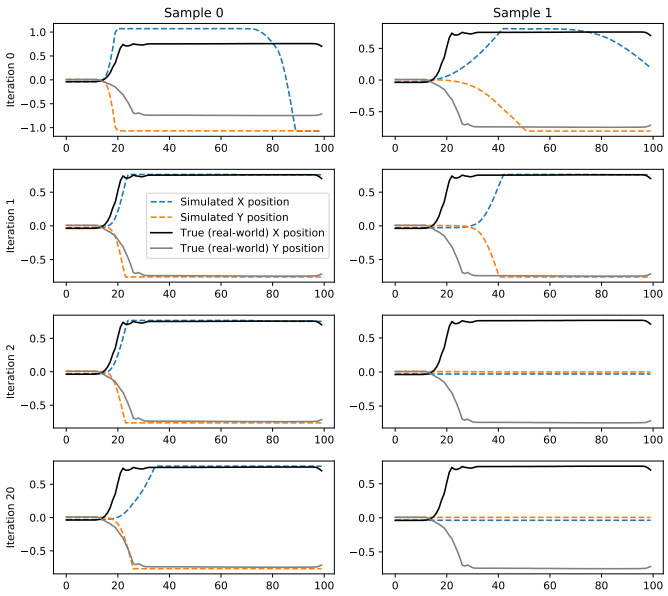

FIGURE 7: Observations for proposed and target models during tuning in Experiment 2 (sim-to-real) expressed in normalized board coordinates (see Fig. 4 for explanation). Each column represents a different initial proposed model and each row represents a different tuning iteration. Time is plotted on the X axis and marble position (normalized to the range [-1, 1] as described in the text) on the Y axis.

namic interactions (as well as non-approximated contact dynamics). These deviations cause significant noise in the estimated gradient. This is apparent in Fig. 8, which shows tuning performance starting in the middle of the parameter space ([0.5, 0.5, 0.5, 0.5, 0.5]). The state error achieved by IRT is low, but unstable.

Fig. 7 figure shows one representative success and one failure case from real-world tuning. In the failure case, the estimated parameters diverged towards 0, predicting no motion of the labyrinth table or marble.

However, despite the noise issues, IRT was still able to tune simulator parameters more quickly than the baselines as summarized in Table 1, achieving a 4.02% error rate after evaluating only 5 proposed simulations.

7 DISCUSSION

This work shows IRT's applicability not just for tuning multiple physical parameters for a given system, but also the parameters necessary to understand how the simulator itself works such as controlling how observations are generated and how quickly the simulation runs. Simulators often are highly sensitive to internal variables, such as timesteps, solver parameters, and the choice of contact models or constraint solvers. The ability to adjust these parameters automatically can relieve some of the burden of the system designer to produce a correctly-tuned simulation even before considering a particular system.

The physical labyrinth exhibits several unmodeled effects, including anisotropic friction, random deviations in the motor delay (due to software) and in the marblei's surface. To model these, we could either characterize the noise and produce a probabilistic estimate, or could add state adjustments to those created by the simulator using a neural network learned function, as has been done in similar works [7, 8, 25].

8 CONCLUSIONS

In this work, we have shown that IRT can perform multiparameter system identification on real-world systems using only visual observations from that system.

The ability to identify multiple and coupled system parameters generates several possible areas for future research. The method enables the improved implementations of model-predictive control (MPC), reinforcement learning, or other model-based control methodologies by closing the reality gap.

TABLE 1: Predicted marble state error statistics for the different optimization methods tested in Experiment 2 (sim-to-real).

	CMA-ES	**Direct Prediction**	**IRT (ours)**
Minimum state MAE	6.68%	34.6%	**4.02%**
Number of simulated rollouts to reach minimum	30	0	5

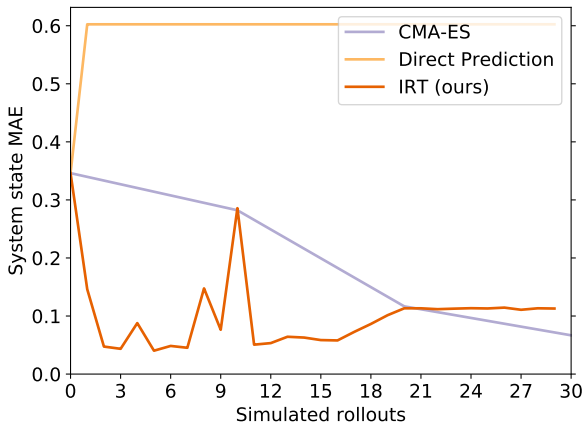

FIGURE 8: Mean absolute state error for Experiment 2 (sim-to-real.

Such parameters could be models as probabilistic functions to account for variance due to system imperfections such as surface irregularities, variations in restitution coefficients depending on the contact point, etc. Using IRT, such models can be made to include more difficult-to-calculate system parameters unrelated to its physical characteristics including communication delays and processing time. These methods could be explored specifically for the completing the labyrinth faster, more safely, or more efficiently. Future work could also look to improve the algorithm by using cameras with higher frame rates and a faster control rate.

We look forward to automatic system identification and parameter tuning from minimal amounts of real-world data as a way for robots to accurately and quickly build new simulators representing their environment.

ACKNOWLEDGMENT

This work was supported by the US NSF (IIS-1564080, IIS-1724157) and US ONR (N000141612835, N000141612785).

References

[1] Camacho, E. F., and Alba, C. B., 2013. *Model predictive control*. Springer Science \& Business Media.

[2] Beyer, H.-G., and Schwefel, H.-P., 2002. "Evolution strategies–A comprehensive introduction". *Natural computing*, ***1***(1), pp. 3–52.

[3] Hansen, N., 2016. "The CMA Evolution Strategy: A Tutorial". *CoRR*.

[4] Sutton, R. S., and Barto, A. G., 2018. *Reinforcement Learning: An Introduction*, second ed. The MIT Press.

[5] Boschert, S., and Rosen, R., 2016. "Digital twin—the simulation aspect". In *Mechatronic futures*. Springer, pp. 59–74.

[6] Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., and Fox, D., 2018. "Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience". *CoRR*, 10.

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

[8] Golemo, F., Taiga, A. A., Courville, A., and Oudeyer, P.-Y., 2018. "Sim-to-Real Transfer with Neural-Augmented Robot Simulation". In Conference on Robot Learning (CoRL), pp. 817–828.

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

[10] Bergatt, C., Metzen, J. H., Kirchner, E. A., and Kirchner, F., 2009. "Quantification and Minimization of the Simulation-Reality-Gap on a BRIO (R) Labyrinth Game". In Proceedings of the First International Workshop on Learning and Data Mining for Robotics, Slovenia, pp. 26–38.

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

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

[13] Åström, K.-J., and Torsten, B., 1965. "Numerical identification of linear dynamic systems from normal operating

records". *IFAC Proceedings Volumes,* ***2***(2), pp. 96–111.

[14] Ho, B. L., and Kálmán, R. E., 1966. "Effective construction of linear state-variable models from input/output functions". *Automatisierungstechnik,* ***14***(1-12), pp. 545–548.

[15] Khosla, P., and Kanade, T., 1985. "Parameter identification of robot dynamics". In 1985 24th IEEE Conference on Decision and Control, IEEE, pp. 1754–1760.

[16] Gautier, M., and Khalil, W., 1988. "On the identification of the inertial parameters of robots". In Proceedings of the 27th IEEE Conference on Decision and Control, IEEE, pp. 2264–2269.

[17] Abdalmoaty, M. R., and Hjalmarsson, H., 2017. "Simulated pseudo maximum likelihood identification of nonlinear models". *IFAC-PapersOnLine,* ***50***(1), pp. 14058–14063.

[18] Allevato, A., Schaertl Short, E., Pryor, M., and Thomaz, A. L., 2019. "TuneNet: One-Shot Residual Tuning for System Identification and Sim-to-Real Robot Task Planning". In CoRR, Vol. abs/1907.1.

[19] Agrawal, P., Nair, A. V., Abbeel, P., Malik, J., and Levine, S., 2016. "Learning to Poke by Poking: Experiential Learning of Intuitive Physics". *Advances in neural information processing systems*, pp. 5074–5082.

[20] Pinto, L., and Gupta, A., 2017. "Learning to push by grasping: Using multiple tasks for effective learning". In International Conference on Robotics and Automation (ICRA), IEEE, pp. 2161–2168.

[21] Herman, M., Gindele, T., Wagner, J., Schmitt, F., and Burgard, W., 2016. "Inverse reinforcement learning with simultaneous estimation of rewards and dynamics". In Artificial Intelligence and Statistics, pp. 102–110.

[22] Xu, Z., Wu, J., Zeng, A., Tenenbaum, J., and Song, S., 2019. "DensePhysNet: Learning Dense Physical Object Representations Via Multi-Step Dynamic Interactions". In Proceedings of Robotics: Science and Systems.

[23] Bauza, M., Hogan, F. R., and Rodriguez, A., 2018. "A Data-Efficient Approach to Precise and Controlled Pushing". *CoRR*.

[24] Hermans, T., Fuxin Li, Rehg, J. M., and Bobick, A. F., 2013. "Learning contact locations for pushing and orienting unknown objects". In 2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids), IEEE, pp. 435–442.

[25] Zeng, A., Song, S., Lee, J., Rodriguez, A., and Funkhouser, T. A., 2019. "TossingBot: Learning to Throw Arbitrary Objects with Residual Physics". In Proceedings of Robotics: Science and Systems.

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

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

[28] Kirchner, E. A., Woehrle, H., Bergatt, C., Kim, S. K., Metzen, J. H., Feess, D., and Kirchner, F., 2010. "Towards operator monitoring via brain reading–an EEG-based approach for space applications". In Proceedings of the 10th international symposium on artificial intelligence, robotics and automation in space, pp. 448–455.

[29] Erez, T., Tassa, Y., and Todorov, E., 2015. "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx". In 2015 IEEE international conference on robotics and automation (ICRA), IEEE, pp. 4397–4404.

[30] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y., 2009. "ROS: an open-source Robot Operating System". In International Conference on Robotics and Automation (ICRA).

[31] Bradski, G., 2000. "The OpenCV Library". *Dr. Dobb's Journal of Software Tools*.