Automatic Parameter Tuning of Robot Simulators with Deep Learning

Robert Kwiatkowski[1], Pol Bernat i Belenguer[2], Abhijeet Bajaj[1], and Hod Lipson[2]

***Abstract*— Properly tuning simulators to achieve good levels of performance when transferring policies from simulation to reality is critical for real world robotics. This task, however, is time consuming and difficult to perform well since it is typically done by hand, relying on heuristics, domain knowledge, and trial-and-error. In this work we present an approach for automatically tuning robot simulators using Deep Learning. Our approach efficiently generates tens of thousands of randomized simulators and identifies which of these parameters is best suited to represent the real world. Once identified, these neighboring simulators can be used to quickly and reliably generate an effective walking policy that works in the real world.**

I. Introduction

The development of novel robot platforms inevitably requires a large amount of work with simulators. These simulators serve a critical role by allowing researchers to develop and test algorithms without needing to risk damaging the often expensive and relatively fragile robot. Thus, the development of, and work to ensure the accuracy of these simulators is a crucial task.

Reflecting the critical nature of building and refining simulators, there have been numerous works that sought to improve the state of the art for robot simulators [1][2][3][4][5][6]. These works tend to focus on improving the precision of the simulator overall, including such complexities as contact dynamics [7]. These studies however always leave the arduous work of tuning each individual simulator to work with the specific robot desired. Leaving the last step to the user is done with good reason. It is very difficult to create a simulator that can take in any model of a robot and then automatically tune this model to perfectly reflect the dynamics of the real robot it seeks to model a priori. Thus, in order to properly execute this tuning some prior information about the dynamics of the real robot is required.

Traditionally, in order to acquire this prior information a roboticist needs to carefully measure and note the specifications of each of its motors, the torques, friction values, speeds, delays, etc. as well as correctly tune the 3D model that accompanies all of this information to have the correct weights, moments of inertia, etc. [1][8]. This process is not trivial and often can be very time consuming and prone to errors, requiring trial-and-error to iteratively refine estimates until tasks performed in simulation translate well enough to the real world. Simply put, the traditional process requires a lot of man-hours to engineer and tune the simulation as well as a lot of machine-hours as the robot has to continually try and fail to execute tasks while iterating on the parameters [9][10].

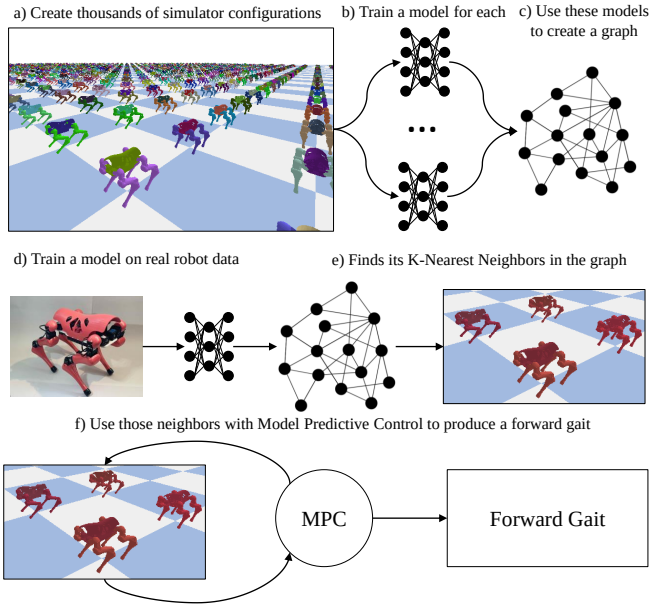


Fig. 1. Overview of our approach. **(a)** We start by creating a large amount (10,000) of simulator configurations. **(b)** Once these configurations are created we produce a small dataset and train a dynamics model on each of them. **(c)** These dynamics models are used to produce the distance values necessary to create a graph. **(d)** The real robot is then used to produce a dataset and a corresponding dynamics model. **(e)** This information is used in conjunction with the existing graph to identify the nearest simulation configurations to the real robot. **(f)** These proximal simulators are used to train a policy which produces a forward gait.

In this work, we present an algorithm to automatically find these simulation parameters as can be seen in Figure 1. We collect a dataset of actions performed by the real robot and a superset of these datasets from thousands of different simulations each with different parameters. Our algorithm uses deep learning to quickly identify which of these simulated datasets most closely matches the dynamics of our real robot. We then use these close simulated results to train our robot to perform a task without any manual tuning of the simulator parameters. We demonstrate that our approach can quickly and efficiently identify parameters that work very closely to those of a manually tuned simulator, and work well enough to allow our robot to learn a forward gait in simulation that transfers well to the real world.

[1]Columbia University, Department of Computer Science
[2]Columbia University, Department of Mechanical Engineering

II. Related Works

A. Automatic Tuning of Simulators

Current solutions to the problem of accurately tuning simulators for transferring policies from simulation to the real world are often based on domain randomization [11]. This is the practice of training a model with a wide range of simulated environments by randomizing the simulator. Prior works have randomized the dynamics of the system [1][12][13][14], the physical environment [15], and the rendering of the system [11]. Tan et al. [1] demonstrated this by learning successful gaits using the Minitaur quadruped from Ghost Robotics [16] by randomizing a set of parameters affecting the dynamics of the system and by heuristically defining the ranges in which that randomization would take place.

Typically, when using domain randomization, one must use heuristics, prior domain knowledge, and trial-and-error to hand-engineer the range of simulation parameters that one is training on. As one increases the range within which randomization is performed it has the trade-off of better performance for computational expense. Thus, domain randomization is reliable but, since the control process has to be tested across all of the randomized simulated environments, it demands a large computational load.

Imitating reference motions has also been proved as a robust general approach for transferring dynamics from the real world to simulation [17][18]. Peng et al. [14] demonstrated an approach for imitating reference motions in animals to allow for sim-to-real transfer of quadruped locomotion on Unitree's Laikago robot [19]. However, the setup is costly and the domain adaptation step still requires heuristics and prior domain knowledge to hand-engineer the range of simulation parameters that one is training on.

Allevato et al. propose TuneNet [20], a one-shot residual tuning for system identification and sim-to-real robot task transfer. It takes as input observations from two different models and, by estimating the parameter gradient landscape, it iteratively updates parameters to converge on simulation parameters.

Recently, Du et al. proposed a method for automatically tuning simulator system parameters to match the real world using real world data from RGB images [21]. Here, auto-tuning of simulation parameters is done by iteratively shifting them to approach that of real world RGB images by predicting whether a given parameter is higher, lower, or close to the real world values. Domain randomization is later applied by restricting randomization to a distribution around the "close enough" parameters that have been determined.

Similarly, the recent SimOpt work [13] also leverages real world data to iteratively shift the distribution of simulation parameters to perform domain randomization that is more reliable. However, they rely on continuous real world data collection prior to every new domain randomization iteration, which hinders the desirable quality of limiting the amount of machine-hours to protect the often fragile robot. Moreover, while this method is effective for robotic arm tasks such as Cabinet Slides, it is challenging to apply these methods to locomotive tasks due to the difficulty of continuously collecting data and resetting the experiment.

Our approach allows us to begin with a very large variety of environments, thus, a large range of simulation parameters, as we do not intend to use all of them. We train a model on a small dataset of actions performed on the real robot, which allows us to quickly discern which simulated datasets are most similar to our real world model and, thus, we only generate a small amount of simulated data from each simulator. This can be done very quickly and is conspicuously parallelizable, eliminating the trade-off for training on large ranges of simulation parameters for great performance. While most of the previous work in this field has been in trying to properly define a realistic domain on which to perform randomization, our system avoids that problem altogether, allowing sim to real transfer on large scale domain randomization without being computationally expensive.

Fig. 2. Fully assembled robot

III. Robot Design

The robot we designed is based on a mammal-type quadruped leg configuration [22], similar to that of the MIT Mini Cheetah [23] or Boston Dynamics Spot [24] robot designs. Figure 2 shows a picture of the assembled physical robot. It weighs roughly 3.6 kg, with 0.15 m and 0.16 m link-lengths for the upper and lower links respectively, and 0.23 m between the front and rear legs. All components, excluding an acrylic sheet for the main body link and off-the-shelf parts, are 3D printed. It was designed to be a robust & low-cost robotic platform which is easy to assemble and service.

A. Mechanical Design

The key functional requirements in the design of our robot were simplicity of assembly and servicing, low-mass, and low-cost.

The main body link of the robot consists of a 5mm thick laser-cut acrylic sheet with an FDM 3D printed cover that houses the hardware and electronics as well as the 'armpit' joint motors for each leg, shown in Figure 3(a). The battery is located in the underside of the acrylic sheet so as to keep the center of mass of our robot low - it is fastened to the body with industrial strength Velcro for ease of replacement.

All leg links were 3D printed using desktop FDM 3D printers. To provide the legs with increased mechanical properties, we post-processed the links by applying a coating of XTC-3D^{TM} epoxy resin. Links were also added M2 and M3 threaded thermal inserts to provide secure fastening to the motor brackets.

Each of the four legs has 3-DOF, one controlling roll ('armpit') and two controlling pitch ('shoulder' & 'elbow') - see Figure 3(a) - with respect to the axis describing the forward-direction of the robot. The joints have a range of motion of ∼90º, 175º, and 225º, respectively.

B. Hardware & Electronics

The main body link houses all the hardware and electronic components excepting the servo motors in the legs, as seen in Figure 3(b). The robot is powered by a 12 V, 72 W·h Li-Ion battery. All motors and control unit are powered by the same battery, with a DC-to-DC converter to step down the voltage and provide electrical isolation between the power electronics and the computer.

All joints are driven by LewanSoul LX-16A Serial Bus Servo motors, which provide 17 kg·cm of torque when powered with 7.4 V and can be controlled with 0.24º accuracy.

Control of the robot is performed using a Raspberry Pi 4 Model B. The computer communicates with a Bus Servo Controller breakout board which directly connects and controls the 12 Servos on the robot. Accelerometer and gyroscope data is collected with the Raspberry Pi Sense HAT attachment board. We control the robot by remotely accessing the on-board Raspberry Pi via WIFI from a laptop computer.

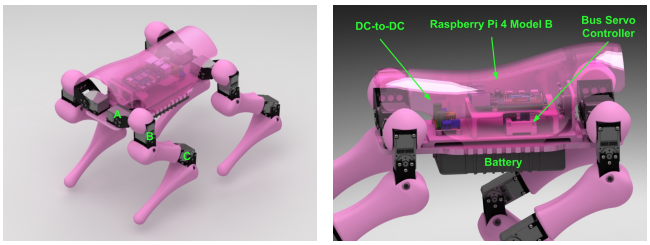
C. Robot Simulator

We built a physics simulation of our robot - as seen in Figure 3(c) - using PyBullet [25], a Python module that extends the Bullet Physics Engine with machine learning and robotics capabilities. Bullet solves the equations of motion for articulated rigid bodies and allows one to adjust parameters to set physical constraints such as contact, friction coefficients and joint torques & velocities.

To simulate the robot, we created a Unified Robot Description Format (URDF) [26] file for our design using data from our Computer Aided Design (CAD) model given specifications for off-the-shelf parts and assuming uniform density for 3D printed parts.

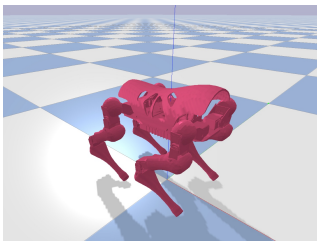
We set up an environment in which we tuned the following parameters for a baseline simulation environment that approached the real robot:

1) Number of simulated *actions per second*
2) *Lateral friction* between the robot links and the floor



(a) Labeled joint names - A: armpit; B: shoulder; C: elbow

(b) Labeled electronic components



(c) URDF of our robot in PyBullet environment

Fig. 3. Renderings with labeled joints & components and URDF or robot

3) *Maximum joint torques*
4) *Maximum joint velocities*
5) *Joint limit positions*
6) *Maximum joint delta positions* allowed
7) *Motor Noise*
8) *Link lengths, masses & moments-of-inertia*
9) *Gravity*

These were the same parameters that we sought to automatically tune with our approach.

IV. Automatic Tuning of a Simulator

A. Individual Model Design

In order to automatically tune the simulator we first collect data on both real and a variety of simulated robots. This data is used to train Deep Learning neural networks to model the dynamics of each of these robots and evaluate which simulator most closely approximates the real robot. In order to do this we need to establish a neural network that will do the modeling.

We use a neural network with 3 layers each having a hidden size of 128. This network is trained with the Adam [27] optimizer and uses ReLU [28] activations between the layers. The model is given an input data corresponding to the current state and action s_t, a_t and is expected to predict the next state s_{t+1}. The state in this case captures a large amount of information about the robot such as its motor positions, roll, pitch, yaw, and velocity of the body. The action is a vector that is passed to the robot's motors to command them to move to different positions. All this information is sufficient to predict the next state and the network that models this can be considered a dynamics model in that it is modeling the dynamics of this system. These models are all trained using Nvidia 1080 TI and Nvidia 2080 TI GPUs.

B. Detecting Similarity

In traditional domain randomization tasks which seek to accomplish a similar goal, a gait would be produced through testing on a wide variety of different sets of environments [11]. This approach is robust but requires a significant computational load as the control process has to be repeated across each of the randomized environments.

In our approach we similarly begin with a large variety of environments but as we do not intend to use all of them we can generate a very large set of environments relative to the computation power available. We assume that by creating a sufficiently broad set of parameters, some of them are likely to be close to the real world, an assumption common to domain randomization as a whole. As our approach solely relies on generating a small amount of simulated data from each simulator it can be done very quickly and is eminently parallelizable.

Once these datasets generated from the simulators are fully collected we then train a deep neural network to model the dynamics of each of these datasets. The details of these models can be seen in Section IV-A. Once all of these models are trained we can use them to evaluate the distance between two different datasets. This distance can be expressed as:

$$d(a,b) = \frac{L_a(b) + L_b(a)}{L_a(a) + L_b(b)} \tag{1}$$

C. Identifying the Nearest Simulators

This distance function outlines the relative symmetric distance between each of the two robots. By evaluating the loss of model a on dataset b and vis versa we describe the cumulative difficulty these models have describing their counterpart. This difficulty is inversely correlated with the similarity between these two models. As a result by dividing the performance on their counterpart's dataset by the difficulty on their own we describe the relative performance of these two models normalized by how difficult their own task is.

This distance value can be used as to produce an undirected graph. In order to produce this graph we construct it in a manner very similar to the Hierarchical Navigable Small Worlds Graphs as described in [29]. In our approach we first seek to find the K-Nearest-Nodes to our new node n, as outlined in Algorithm 1, we begin by selecting a random node a from our graph G. Once chosen we then go through all of the edges E_a of node a and compute the distance $d(n,a)$. We then find the neighbor of node a which we call node b that has the lowest distance to node n. Once found we repeat this process with node b. This continues until the chosen node being evaluated exists at a local minimum.

This process is an approximate k-nearest neighbors (KNN) method and as it is approximate it can result in suboptimal outcomes if the graph is not convex with respect to the distance problem. In order to mitigate this problem in practice, we repeat the algorithm 4 times and take the best k choices. For all experiments $k = 5$. This significantly reduces the probability of a local minimum appearing and results in a more robust selection.

Algorithm 1 K-Nearest-Nodes

Require: a graph G, a node n to test against
Ensure: $|G| > 0$
  $N = \{\}$ We initialize a set of the top k nearest neighbors
  while $|N| < k$ **do**
    $a =$ a random node from G
    $S = \{\}$ The set of visited nodes
    while $a \notin S$ **do**
      $V = \{k \in E_a$ with the lowest $d(n,i) \forall i \in E_a\}$
      $N =$ the k elements with the lowest d in $N \cup V$
      $b =$ the node with the $\min d(n,j)\ \forall j \in V$
      if $d(n,b) < d(n,a)$ **then**
        $a \leftarrow b$
      end if
    end while
  end while

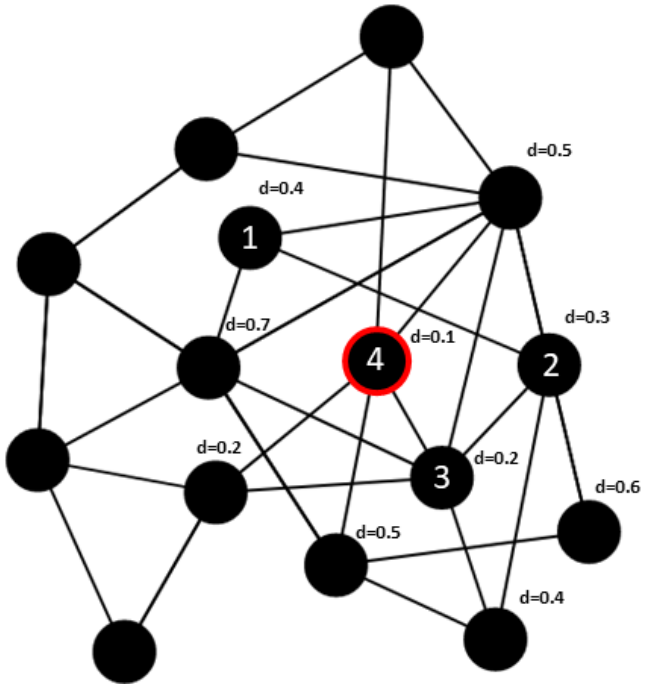
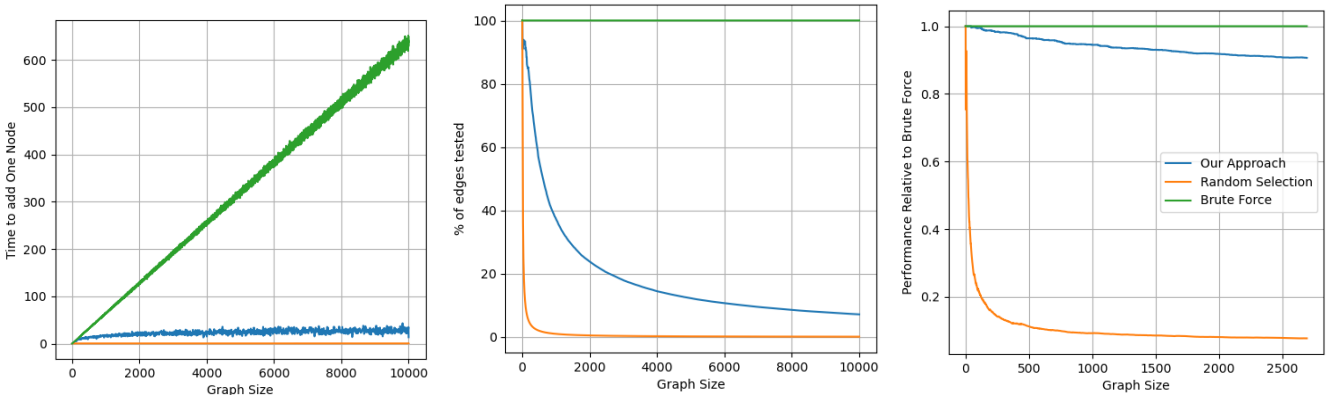


Fig. 4. An example of our KNN algorithm in practice. The first node (1) is selected randomly and then each of its neighbors are tested to reveal their distance. This process is continued until all surrounding nodes have a higher distance score than the chosen node.

Our approach results in an expected $O(log(n))$ which allows for a significant number of robots to be evaluated. Furthermore, the high degree of accuracy present in the approximation means that the chosen nodes are nearly always close to the real dynamics of the system we choose to model. These two properties combined result in a system that can quickly evaluate a number of simulator configurations and detect those that would be closest to a collected dataset from the robot.



(a) Time Taken to Add a New Node (b) The Number of Edges Evaluated (c) The Performance Relative to Optimal

Fig. 5. The three main tests of the efficiency and effectiveness of our approach. These charts show that our approach is significantly faster and more efficient than brute force while approaching the effectiveness of it. Whereas a random choice has virtually no effectiveness demonstrating that this is not a trivial task.

D. Testing Our Approach

In order to evaluate our approach we use the simulators to find a walking policy that works in the real world on the physical robot without any manual tuning of the simulator. In order to produce this walking gait we use a Hill Climber [30] to find the parameters necessary for a sinusoidal gait. These sinusoidal gaits use a few parameters to modulate the motors of the robot in a sinusoidal fashion. These gaits have been frequently used in the past and are robust and very efficient [31][32][33].

In order to use this Hill Climber we test each of the candidate gaits on all of the k-nearest simulators. Any gait that successfully performs a gait on all of the k-nearest simulators is very likely to succeed in the real world. As such, our approach pulls from the traditional approach of domain randomization and is able to achieve good results without the need to test gaits on thousands of simulators.

V. Results

A. Assessing Our Algorithm

We assess our algorithm across three key tests the results of which can be seen in Figure 5. These three tests assess how likely it is that this algorithm will be useful in determining which of the identified nearest nodes in the graph will be a good approximation for our new robot. These metrics can be summarized as those that measure the efficiency of our algorithm, and thus how broad a search space we can cover, and the effectiveness of our algorithm representing how well our choices will be within the large search space. These two metrics are both crucial as without a broad set of parameters to search over it is unlikely that we will find a set that is close to the real world. Furthermore, without an effective search algorithm we will be unlikely to effectively identify the correct set of parameters even if the search space is guaranteed to contain a good set of parameters.

*1) **Efficiency**:* The first core property of our approach is its efficiency. For the purposes of our test, we define efficiency by the number of nodes that can be processed in a given time. Based on this metric we measure the speed of adding a new node as the set of nodes increases linearly. As can be seen in Figure 5(a) our approach scales logarithmically with the size of the graph. This means that we can cover an exponentially larger search space than other linear domain randomization tools making the probability that we encounter a good set of parameters exponentially greater as well.

This efficiency is furthered by looking at the number of edges that are actually evaluated. The reason for this efficiency is because there is an inverse relationship with the size of the graph and the number of edges that need to be explored in order to produce a good approximation of the nearest neighbors (and thus the closest set of parameters).

*2) **Effectiveness**:* The second core property of our approach is its effectiveness. This effectiveness can be defined as how well our algorithm finds simulators that closely approximate the best possible simulators available in the graph. This metric is important to understand as this effectiveness underpins the entire ability of our approach to identify a good set of parameters when confronted with real data. In order to evaluate this effectiveness we search the entire set of simulators observed before in a brute force fashion, testing every possible pair of simulators and returning those that perform best thus eventually finding the optimal set of simulators.

When compared with this brute force method our approach, while it does become worse as the size of the graph increases, achieves very good results and those that are significantly better than a random selection suggesting that we have statistical significant results. These results overall paint a picture of a very high success rate of 90% as good as the theoretically optimal brute force.

The significance of these results is further emphasized by the sheer improvement over a random search. Random search is, for all intents and purposes, the standard domain randomization algorithm. Typically in domain randomization a curated or random selection of environmental parameters

is chosen to help the policy become closer to the real world. Our results demonstrate that these random selections become exponentially less useful than our approach as the size of the graph increases. This is a very significant finding as it shows the power of increasing the search space when it is able to be searched effectively.

Furthermore, this test against random selection serves to identify the difficulty of the problem being solved. If a random selection produces good results then there is no need for a complex algorithm as all produced sets of simulation parameters will perform good enough. However, this is not the case. Due to the great diversity in the simulations tested we are able to ensure that at least some of those parameters will be close to the real world. While this comes at the expense of being a more difficult problem to solve, our approach is able to mitigate that issue and ensure a high level of performance. This performance suggests that our approach, for all intents and purposes will be good enough to identify very high quality simulators if given a representative set of data.

Fig. 6. Transfer of our forward gait policy with a gait trained on on an automatically tuned simulator. The bottom figure pictures the gait on our baseline hand-tuned simulator and the top figure shows the real robot successfully moving forward using a policy trained using our approach.

B. Results in the Real World

Our results in the evaluation of the algorithm, however, are not as important as how well our approach translates to real robots. As such, we tested the robot defined in Section III in the manner described in Section IV-D. We collected 4 distinct datasets in order to provide a broad coverage for the robot and the algorithm.

When tested the robot was able to use the automatically tuned simulators to find a gait 100% of the time, transferring to the real world successfully as shown on Figure 6. This result alone strongly indicates that this task we were attempting to solve was not trivial and the appropriate tuning of the simulator was critical to its success.

Furthermore, when gaits were tuned on simulators with parameters automatically detected by our algorithm their gaits achieve scores similar to that of gaits generated in an identical fashion as described in Section IV-D but by using a hand tuned simulator. The main difference here is in the time needed to collect the 1000 datapoints necessary to automatically tune the simulator was around ten minutes versus the countless hours needed to manually tune the simulator to closely approximate the complex dynamics of a real robot.

VI. Future Works

While this work represents a significant step in the search for efficient simulation tuning for real robots there is still much work to be done. We limited our approach to randomizing elements about the robot however there is also significant work that can be done applying the same technology to the world itself. Many robots struggle to transfer to complex and difficult environments and using a similar technology to adaptively train the robot to better deal with new terrains based on situations in which it fails could be a powerful application of this work.

Furthermore, this technology could also be applied to the rapid prototyping of robots. If instead of using a variety of simulators to tune the robot one instead used a variety of robot morphologies it would be possible to efficiently identify which robot morphologies are similar to each other. Through this identification one could leverage the datasets or the dynamics models of those identified morphologically similar robots to speed up the process by which the dynamics model on the new robot is learned. Such a work would need to identify how to effectively transfer knowledge from one morphology to another but could, for all intents and purposes, use our framework for identifying similar body configurations to do much of the work.

VII. Conclusion

In this work we present a fully developed and deployment ready algorithm for automatically tuning a simulator in a data driven fashion making use of the power of Deep Learning to model the dynamics of a robotic system. Our approach is both efficient and effective allowing us to consider parameters from tens of thousands of robots even with relatively low computation power.

Through this approach we seek to give researchers the power to leverage techniques previously available to large labs and institutions with access to boundless compute resources. By presenting an incredibly scalable algorithm that allows researchers to automatically produce a gait from a robot without any tuning of the simulator behind it we present a way for an individual with a single desktop computer to save many hours that would have otherwise been spent in the arduous task of tuning and tweaking a simulator to properly represent a robot.

Acknowledgments

Supported by DARPA MTO grant HR0011-18-2-0020.

REFERENCES

[1] 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," *arXiv preprint arXiv:1804.10332*, 2018.

[2] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, p. eabc5986, 2020.

[3] K. Choromanski, A. Iscen, V. Sindhwani, J. Tan, and E. Coumans, "Optimizing simulations with noise-tolerant structured exploration," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2970–2977.

[4] A. Boeing and T. Bräunl, "Leveraging multiple simulators for crossing the reality gap," in *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*, 2012, pp. 1113–1119.

[5] S. Koos, J.-B. Mouret, and S. Doncieux, "Crossing the reality gap in evolutionary robotics by promoting transferable controllers," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 119–126. [Online]. Available: https://doi.org/10.1145/1830483.1830505

[6] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1133687

[7] S. Kolev and E. Todorov, "Physically consistent state estimation and system identification for contacts," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 1036–1043.

[8] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2245–2252.

[9] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International journal of robotics research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[10] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 3406–3413.

[11] 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 *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[12] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3803–3810.

[13] 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," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8973–8979.

[14] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.

[15] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[16] G. Kenneally, A. De, and D. E. Koditschek, "Design principles for a family of direct-drive legged robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 900–907, 2016.

[17] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–14, 2018.

[18] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, and S. Levine, "Sfv: Reinforcement learning of physical skills from videos," *ACM Transactions On Graphics (TOG)*, vol. 37, no. 6, pp. 1–14, 2018.

[19] "Laikago: Let's challenge new possibilities," 2018, http://www.unitree.cc/e/action/ShowInfo.php?classid=6id=1.

[20] A. Allevato, E. S. Short, M. Pryor, and A. Thomaz, "Tunenet: One-shot residual tuning for system identification and sim-to-real robot task transfer," in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 445–455. [Online]. Available: https://proceedings.mlr.press/v100/allevato20a.html

[21] Y. Du, O. Watkins, T. Darrell, P. Abbeel, and D. Pathak, "Auto-tuned sim-to-real transfer," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 1290–1296.

[22] S. Tsunoda, H. Nabae, K. Suzumori, and G. Endo, "Power consumption comparison between mammal-type and reptile-type multi-legged robots during static walking," in *2022 IEEE/SICE International Symposium on System Integration (SII)*, 2022, pp. 459–466.

[23] B. Katz, J. D. Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6295–6301.

[24] E. Ackerman, "Spot is boston dynamics nimble new quadruped robot," *IEEE Spectrum*, 2015.

[25] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[26] "URDF - ROS wiki." http://wiki.ros.org/urdf.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[29] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2020.

[30] B. Selman and C. P. Gomes, "Hill-climbing search," *Encyclopedia of cognitive science*, vol. 81, p. 82, 2006.

[31] F. Iida and R. Pfeifer, "Cheap rapid locomotion of a quadruped robot: Self-stabilization of bounding gait," in *Intelligent autonomous systems*, vol. 8. IOS Press Amsterdam, 2004, pp. 642–649.

[32] N. Kau, A. Schultz, N. Ferrante, and P. Slade, "Stanford doggo: An open-source, quasi-direct-drive quadruped," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6309–6315.

[33] G. A. Prasetyo, A. F. I. Suparman, Z. Nasution, E. H. Binugroho, and A. Darmawan, "Development of the gait planning for stability movement on quadruped robot," in *2019 International Electronics Symposium (IES)*, 2019, pp. 376–381.