Tackling simulation inconsistencies in the robot design process by selective empirical evaluation

Anwesha Chattoraj*
anwchatto@g.ucla.edu
UCLA
Los Angeles, California, USA

Eric Vin*
UCSC
Santa Cruz, USA
evin@ucsc.edu

Yusuke Tanaka
UCLA
Los Angeles, California, USA
yusuketanaka@g.ucla.edu

Jill Naldrien Pantig
UCLA
Los Angeles, California, USA
jilliannmpantig@gmail.com

Daniel J. Fremont
UCSC
Santa Cruz, USA
dfremont@ucsc.edu

Ankur Mehta
UCLA
Los Angeles, California, USA
mehtank@g.ucla.edu

ABSTRACT

We present a computational design pipeline that allows evaluation of the robot and environment parameters in a robust manner, giving insight into interactions that can lead to mismatch between simulated behaviour and reality. Our pipeline evaluates robot designs across different design parameters in a large variety of stochastically-defined environments to robustly infer the qualitative effect of robot parameters on its performance. We then quantitatively ground this insight by selecting and building a small number of physical robots to help establish bounds on the trend in parameters observed in simulation. This combination of simulation and empirical evaluation helps narrow the sim-to-real gap without excessive expensive physical testing to augment the intuition of the human designer.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing design and evaluation methods**.

KEYWORDS

robotics, computational design, empirical evaluation, sim2real, scenario description language

ACM Reference Format:
Anwesha Chattoraj, Eric Vin, Yusuke Tanaka, Jill Naldrien Pantig, Daniel J. Fremont, and Ankur Mehta. 2023. Tackling simulation inconsistencies in the robot design process by selective empirical evaluation . In *Cyber-Physical Systems and Internet of Things Week 2023 (CPS-IoT Week Workshops '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3576914.3587527

1 INTRODUCTION

Simulation can be a powerful tool for robot design, allowing inexpensive exploration of many candidate designs in a variety of environments. However, simulated robot behaviors do not always match reality — the so-called "sim-to-real" gap — causing the results of simulation to be approximate or even misleading. Figuring out the interaction of environment parameters and robot parameters that cause the mismatch is a large part of the job of a robot designer. Hence, there is the need for a system that is able to evaluate a variety of robot designs in simulated environments, and then a selected few in a real environment, in order to better interpret the simulation results. In this paper we present such a system, providing a first step toward bridging the gap between reality and simulated behaviours, that helps augment the designer's intuition in dealing with this gap.

In order to evaluate robot and environment parameters together, our system must satisfy two requirements. First, it must provide a design framework to quickly design a wide variety of complex robot morphologies. To assess these designs in real-life as well as simulation, the framework must allow designs to be both exported to a simulator and also be rapidly fabricated.

Second, the system must provide a framework that allows the designer to quickly design parameterized environments, and in fact a probability distribution over such environments. Defining the environments in a probabilistic manner allows a greater variety of scenarios to be explored, while also making the evaluations of the designs robust to noise and uncertainty. This is critical to be able to rely on the results that simulation would provide, as we can verify that our designs are robust against minor perturbations that will almost surely be present in a real-world environment.

An important consideration for such computational design pipelines is closing the "sim-to-real" gap, i.e, establishing a correspondence between the simulated results and real-life behaviour of the robot. [Alattas et al. 2019] defined this "reality gap" as the high cost of performing evolution in physical hardware, which limits the design space that can be realized in real life. [Samuelsen and Glette 2015] explored the reality gap in transferring the simulated behaviour of 3D-printed robots to real life, sub-sampled from a large pool of simulated designs, many of which did not quite match the simulated performance. This evaluation could form the basis of a "transferability model" [Samuelsen and Glette 2015] , based on a small number of real-world samples, to bridge the sim-to-real gap. The robots thus designed should also be easy and cheap to fabricate, so as to be able to quickly test out viable designs.

We take this idea forward into a cohesive framework to assess robot designs, first in simulation and then in reality, bringing an additional level of insight into the design process by increasing the size of the design space that the designer can traverse, giving a more granular look at how the robot and environment parameters interact with one another. The main contributions of the paper can be summarized as follows:

- Provide a oracle to quickly evaluate parameterized robots in both probablistically-defined simulated environments and real-life physical experiments.
- In doing so, provide a way to augment the intuition of the human designer in robustly understanding how the design parameters affect robot performance across a given class of environments.

2 RELATED WORK

The effect of the environment on the performance of different robot morphologies has been a long-standing research topic that has been addressed by several computational design pipelines. Papers such as [Zhao et al. 2020] and [Miras and Eiben 2019] discuss methods of optimizing and generating robot designs over a few specified terrains in simulations, without addressing the question of sim-to-real agreement. Pipelines that provide methods to fabricate feasible designs evaluated in simulated environments include [Auerbach et al. 2014], [Megaro et al. 2015] and [Desai et al. 2017]: these do not incorporate any feedback from real-life evaluations of their robot designs in their selection criteria. The paper [Whitman et al. 2020]

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CPS-IoT Week Workshops '23, May 9–12, 2023, San Antonio, TX, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0049-1/23/05.
https://doi.org/10.1145/3576914.3587527

looked at a deep reinforcement learning-based approach for selecting modular robot designs that optimize performance while minimizing cost and design complexity. We, in contrast provide a method of using empirical real-life tests along with a grid search, to constrain our feasible design space. The work in [Geilinger et al. 2018] does in fact evaluate 3 real-life fabricated models chosen from their optimization routine; the fabrication method however is time-consuming and only really serves to verify the results of the simulations, not augment the intuition of the designer while performing an iterative search over feasible design parameters. This work also does not take into account the effect of varying environmental parameters.

Several additional works attempt to bridge the sim-to-real gap in robot design without exploring the effect of environment parameters, e.g. [Moreno et al. 2017] on quick-assembling robot modules, [Schaff et al. 2022] on soft robots, and [Kriegman et al. 2020] on pneumatically-actuated silicone robots on a flat plain. In these works the environment plays no significant role in determining the design parameters.

[Park and Lee 2021] specifically seek to generate novel designs of wheeled mobile robots from combinations of modular components, evaluated in 2 static environments. The robots can be realized in real life as well; however the authors have not demonstrated a use of the real-life fabrication and validation in influencing the choice of a feasible set of design parameters, instead relying solely on simulation results.

[Zonghao et al. 2022] evaluates parameterized legged robots in three fixed environments. This work seeks to define the impact of human intuition on the evolutionary process. Our work in contrast seeks to study the combined effect of design and environmental parameters, by including the impact of real-life tests in determining the feasibility of a given design.

The technique of domain randomization has previously been used to avoid overfitting when training models or designing policies, as in [Mehta et al. 2019; Ramos et al. 2019]. Our work can be viewed as applying a form of domain randomization, where the randomization is used to provide more robust evaluations of different designs' performance in real-world scenarios.

Finally, in the domain of autonomous vehicles, [Fremont et al. 2020] bridges the sim-to-real gap by using tests in simulation to guide selection of tests to perform in reality. As in our framework, tests are generated from an environment model written in the Scenic probabilistic programming language [Fremont et al. 2019]; however, their framework assumes a fixed vehicle design, while our paper enables joint exploration of designs and environments.

3 SYSTEM OVERVIEW

Our goal is to empower designers to make better choices while designing robots; hence this simulation-aided design tool that allows one to close the sim-to-real gap that exists between robot designs and real-life behaviour. What sets our paper apart is that our design pipeline allows us to test designs in environments with varying parameters and perturbations that make the results more robust. These then let us hone down on a set of real-life exploration points, which can be set up and run rapidly to help identify trends in design parameters more accurately. In particular, we address the problem of uncertainty and noise in simulations by anchoring to real-life experiments as opposed to limiting to "sim2null" or simulation-only research as discussed in [Höfer et al. 2020].

Our paper seeks to go beyond the approaches in the literature on two fronts: 1) Have a rapid, low overhead way of prototyping selected robot designs, and 2) Be able to define parameterized and potentially-stochastic environments that can lead to more robust assessments of candidate designs. These two features would allow the user to extract meaningful trends in design parameters by anchoring them with the real-life data gathered from the smaller set of experiments.

Doing so requires two frameworks: A robot design module and an environment design module. This is achieved in this paper by building on the work of two pre-existing tools. First, the Robot Compiler (RoCo) [Mehta et al. 2015, 2014; Mehta and Rus 2014] creates parameterized, modular robot designs that can be evaluated in a simulator. RoCo also allows the user to rapidly and inexpensively fabricate these designs in real life by cutting and folding 2D patterns and adding simple electronic parts. This lets the user iterate through a large number of designs in simulation and (fewer) in reality with reduced time and overhead.

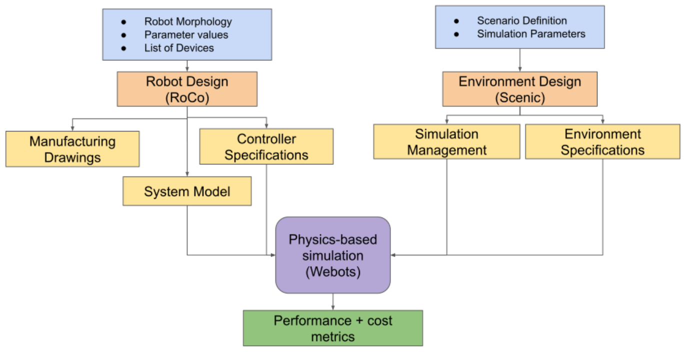


Figure 1: System Overview.

Second, Scenic [Fremont et al. 2019, 2022] is a probabilistic programming language for modeling the environments of cyber-physical systems. When paired with a compatible simulator, Scenic allows the the user to describe, generate, and simulate scenarios of interest while enforcing requirements and recording data.

We bring the outputs generated by these two components into the robot simulator Webots [Michel 2004], which allows for a seamless evaluation of selected designs in well-defined, controllable, probabilistically-defined environments. Webots provides a plethora of sensors and custom physics options, while being the least CPU-intensive among similar simulators [Collins et al. 2021].

The architecture of our system is depicted in Fig. 1. In the remainder of this section, we describe the robot and environment design components of the pipeline in detail.

3.1 Defining and Modifying Robot Designs

3.1.1 Parameterized Components. A key requirement of our system is the ability to easily generate a large number of robot designs. The Robot Compiler is a tool that lets the user design robots by using parameterized, reusable components. This framework allows for creation of modular, reusable components that can be superimposed via complex shape grammars, while still allowing for manufacturable designs.

An example of the composition of complex geometries using simple modules can be seen in Fig. 13 in the Appendix. Simple components are progressively combined into more complex ones, constraining their parameters at each stage, until the robot design is complete. (A similar example is shown in [Yan et al. 2022].) Since the sub-components inherit the properties of their parent modules, the user need only modify parameters at the root level to get a family of designs. As an example, a wheeled mobile robot with two hinges can be constructed by combining 3 wheel modules, 2 hinge modules and one brain module, resulting in a snake-like pivoting car.

We extended the RoCo platform to generate Webots-compatible robot definitions, preserving functional relationships between parts, allowing the user to define physics variables, and providing all the sensing and actuation capabilities defined for a given robot. This ensures that any general robot designed within RoCo can be imported into Webots and simulated.

3.1.2 Robot Morphology and Controller. Another key requirement is the ease of manufacturability of the robot designs. Many computational design pipelines addressing the sim-to-real gap make use of readily-available components to create new designs [Park and Lee 2021], which severely limits the capability of the system to explore the effect of varying the design parameters of the individual parts. Other pipelines make use of

3D-printed custom robot modules [Auerbach et al. 2014], which can take hours to fabricate. Alternatively, the mechanical structures needed to build a robot can be realized from the 2D shell of the desired geometry, obtained by cutting and folding any sort of sheet. RoCo generates such fold patterns for each design; these can be assembled in minutes, greatly accelerating the prototyping phase.

In this paper, we explored 4-wheeled car morphologies with varying parameters that include the length and width of the main body and the radius of the wheels. We limited our experiments to this space of designs in order to obtain a more interpretable relationship between the behaviours observed in simulation and reality. For the same reason, we chose a simple differential drive controller so as to highlight the particular interplay between the robot design and the environmental parameters, which could get obfuscated with a more complex controller. Since we ran multiple experiments over many manufactured robot designs, these considerations were taken into account to showcase the capability of this pipeline in a preliminary manner.

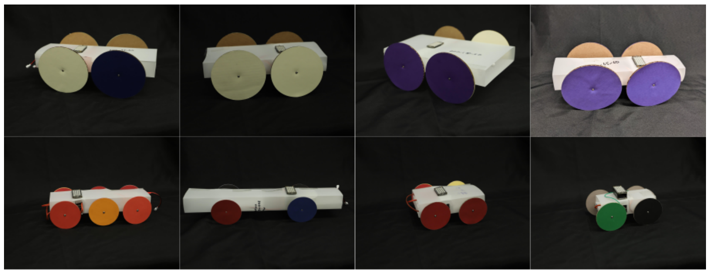


Figure 2: The different fabricated 4-Wheeled and 6-Wheeled cars. Top Row: Designs evaluated in step pyramid environment. These had varying widths, with length, number of wheels and radius fixed; Bottom Row: Other Fabricated Designs, these had an assortment of different parameters.

3.1.3 Fabrication. The fabricated robots need to be easily testable in real-world scenarios to enable rapid iteration in the design process. The robot designs thus realized were fabricated via cutting and folding the 2D patterns generated by RoCo. The sheet material used was PET (Polyethylene terephthalate) which is lightweight and waterproof. The assembly utilized off-the-shelf electronic components including the Adafruit HUZZAH32 – ESP32 Feather Board as the microcontroller, a Servo Featherwing [Earl 2012] and FS90R continuous-rotation servos to drive the wheels.

3.2 Defining Environments

As we have argued above, modeling a space of environments, perhaps with stochastic elements, can be substantially more useful than assuming a fixed environment. For this reason, our framework allows the user to define the environment using the Scenic probabilistic programming language [Fremont et al. 2019, 2022]. Scenic provides flexible, readable syntax for laying out the geometry of an environment which is useful even when the environment is fixed; in addition, Scenic allows placing distributions and constraints on any environment parameters (see Appendix B for detailed example programs based on our experiments). By using Scenic, we are able to define easily define complex and layered distributions over environments, from which Scenic can then sample to generate concrete test cases.

We use Scenic's probabilistic features in two ways: first, to address imperfect fidelity of the environment model and the simulator itself. If a robot performs well on an environment in simulation, assuming the simulator has *perfect* fidelity, we can be confident that the robot will perform well in real life on the *same* environment. These two restrictions are not insignificant: in most applications the simulator will not be perfect and the environment we simulate will not be exactly the environment we intend to deploy our robot to. We can guard against both these cases by considering not only the specific environment, but also *nearby* environments which are almost the same as our original environment. If our design performs well on many environments in such a neighborhood, we can gain some confidence that the design will work well on the real-world instance of that environment, which will probably be a nearby environment. We can also deduce that the results of our simulator are at least consistent over a neighborhood of environments, even if we cannot be sure they will have perfect fidelity. Scenic makes it easy to add random perturbations to an existing environment model with its `mutate` statement.

Second, we can use probabilistic modeling in cases where the exact environment the robot will operate in is unknown, but can be roughly bounded and parameterized. For example, consider a robot that will be deployed somewhere within a region with rough, hilly terrain. The deployment location is not known in advance, and so the exact environment cannot be determined, but by looking at the whole region one can infer a distribution over the size, spacing, etc. of the hills in any section of the overall environment. In this case, even if there are no robot designs that work well for all possible deployment locations, we may be interested in finding a robot that has a high probability of working well. We can then build a Scenic model containing hills with randomized dimensions and positions, and evaluate our designs on this model. The results of test cases sampled from this model can then provide insight into how different aspects of the unknown environment may affect the performance of the robot.

To support our system, we extended Scenic in two ways. First, in order to support application-specific evaluation metrics for robot designs, we added a `record` statement which saves the value of an arbitrary Scenic expression at the start or end of a simulation, or as a time series over the entire simulation. Second, in order to test different robot designs against the same environment, we added an API to *condition* the distributions of specified objects and parameters in a Scenic program to their values in an already-sampled scenario. This allows our pipeline to sample a random environment from the Scenic model, then fix the environment parameters while varying robot parameters to test different robot designs.

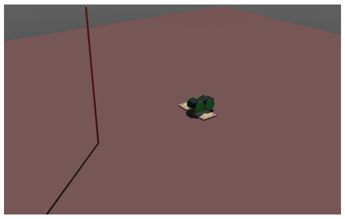
4 EXPERIMENTS AND RESULTS

A key contribution of this paper is the ability of the pipeline to define complex parameterized environments and robots, so the choice of both and the parameters we choose to change to see different behaviours becomes very important.

We performed 2 main experiments to evaluate our pipeline, performed in simulation (in Webots) and in real-life environments to capture the sim-to-real gap. These experiments aim not to identify a single optimal design, but to outline how the complex interplay in the design and environment parameters may influence the space of feasible designs.

Our first experiment seeks to illustrate the correspondence between reality and simulation for a very simple static environment: two speed bumps, parameterized by the height of bumps, the gap between them and their location on on flat plane. The robot must cross this simple obstacle to reach the target. We evaluated a 4-wheeled car on a pair of such speed bumps with steadily increasing heights, until it was no longer able to cross them. The height ranged from 6mm to 30mm high, in 5 steps. We observed that in real life the robot was able to cross the bumps up to a height of 30mm, where it simply stalled as shown in the last image in Fig. 7. In simulation, however, for the same parameter values, the front wheels of the car climbed on top of the bumps and then got stuck, as shown in Fig. 3.

This result demonstrates the sim-to-real gap in the behaviour of the robot. This knowledge is useful to a designer, so as to be able to interpret the results and understand what parameter values might cause these discrepancies. However, it is important to note that this is a static environment, with only one robot being evaluated. This evaluation would be more robust and reliable if the results had been averaged across multiple slightly perturbed environments: this would account for the noise inherent in real-world experiments, smoothing out the effect of discrepancies.


(a) 6mm: climbs over.

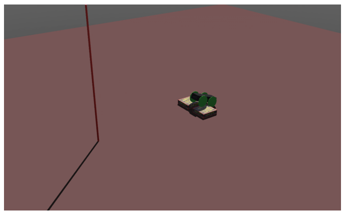

(b) 30mm: gets stuck on top.

Figure 3: Effect of speed bump height on robot behavior.

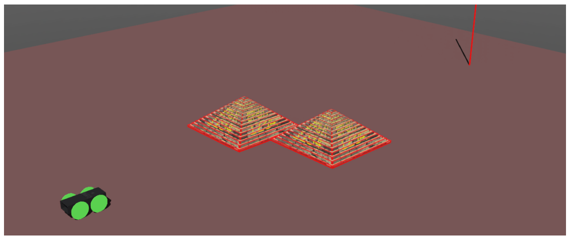


Figure 4: Step Pyramid Experiment in Simulation

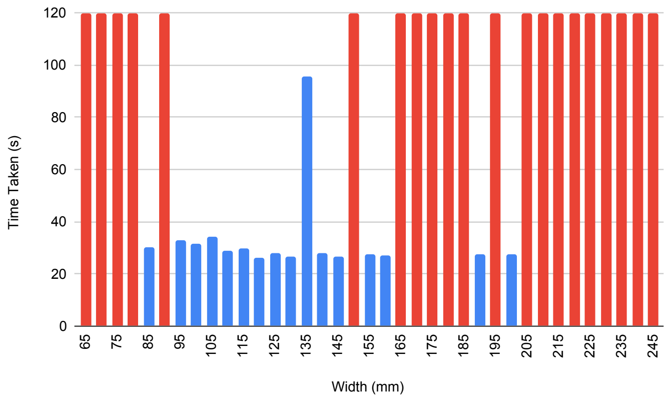


Figure 5: Performance of different 4-wheeled car designs in a single simulated environment. Trials in red failed to complete the pyramid-crossing task within 120s.

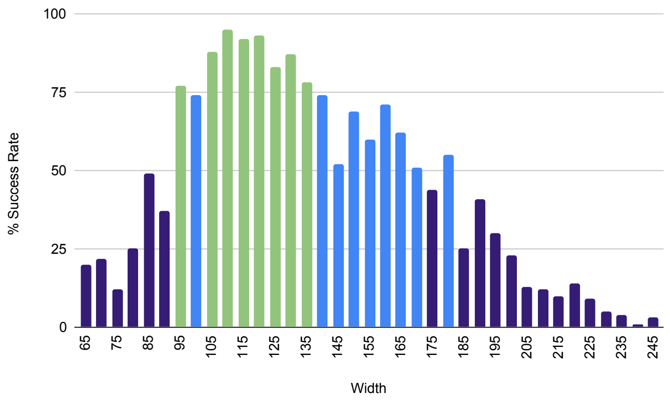


Figure 6: Results averaged over 100 mutated environments. Green indicates $> 75\%$ success rate, blue $> 50\%$ but $< 75\%$ success rate, and navy $< 50\%$ success rate.

Thus, a better method of evaluating trends in parameters is illustrated in our next experiment.

In our second experiment, we evaluate a set of 4-wheeled cars on the task of navigating through two offset step pyramids within 120s, shown in Fig. 4. We constructed a corresponding real-life environment, shown in Fig. 8.

For the simulations, to provide robustness against slight differences in the simulated environment and the real environment, we evaluated all robots on 100 instances of the step pyramid environment with slight random perturbations to the center positions of the pyramid and the start position of the robot. We call these instances of the environment *mutated* environments. For the robots, the length, radius, and number of wheels were kept constant, and the width parameter was varied from 65mm to 245mm.

This class of experiments were selected as this is a constrained environment, and they challenge the idea of intuitively thinking that a bigger robot would perform better across the board: the simulation and experimental results will indicate otherwise. Having multiple mutated environments helps us indicate the direction of these trends in a more robust manner.

We first look at Fig. 5, which shows the variation in performance of various 4-wheeled car designs in a single environment (without mutations). A time taken of 120 seconds indicates a failure case, with shorter times considered successes. The results are clearly not smooth, making the choice the minimal set of designs to validate in real life tricky, since there are many viable candidates. Setting the failure threshold to 40s yields $\sim$ 15 viable designs that seem to perform at the same level, but does not address the outlier design with width 135mm that succeeded but only with a longer timeout.

This problem is addressed by running the same set of robots in multiple environments with slightly mutated parameters as described above. Fig. 6 shows the mean success rate of each of the designs across 100 such mutated environments. The colors for each of these averaged times give an idea of the failure rate of the robots: green indicates >75% success rate, blue indicates >50% but <75% success rate, and navy indicates <50% success rate. Thus the designer can modify the threshold for success according to their requirements, and choose only the designs that meet the criteria.

For our experiment, we chose to target the designs that yield at least a 75% success rate, i.e., the values indicated in green. Thus for our real-life tests we chose to fabricate the designs in the middle of the feasible region (green) and a few on the edges and outside of it. Specifically, we selected robots with widths of 95mm (left edge of the green region), 115mm (middle of the green region), 65mm (the left edge of the navy region) and 180mm (the right edge of the blue region). For each design we conducted several trials; a trial was considered successful if the robot was able to navigate the step pyramids and emerge on the other side upright and kept moving forward. The results from the real-life tests are summarized in Tab. 1.

We can see that the real-life experiments favour the robot with 95mm width, instead of the 115mm width expected from simulations. The designer now knows that they should prefer a width value closer to the edge of the feasible region (closer to 95mm) indicated by real-life experiments. The failure of the robot with 180mm puts a rough upper bound on width values, indicating that the other end of that region (and beyond) would be a waste of time to explore. Similarly, the failure of the 65mm width robot puts a lower bound on the viable region. Thus, the real-life experiments drastically narrowed down the design space the designer needs to explore, or optimize over to settle on a final robot design to be deployed.

This insight would not have been possible to gain without the use of parameterized, rapidly fabricable robots and testing against probablistically-defined environments in our computational design pipeline. The ability to run targeted real-life experiments quickly vastly improves the design decisions the engineer makes with regards to selecting a set of feasible robot parameters.

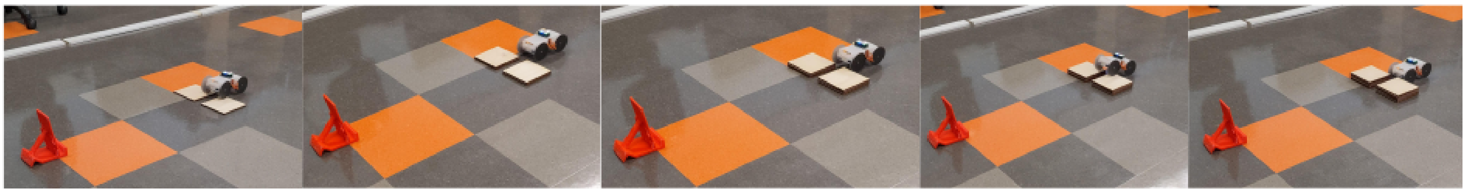


Figure 7: Speed Bump Experiments in Real Life. The height of the bumps ranges from 6mm in the first photo to 30mm in the last, as in Fig. 3. The last image captures the point of departure from simulated results, diverging from Fig. 3 (b).

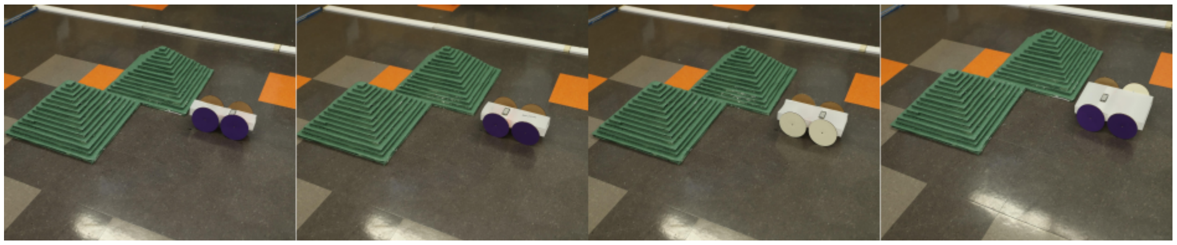


Figure 8: 4-Wheeled Car Real Life Validation Experiments. The 4 robot designs correspond to those in Tab. 5.

Table 1: Experiment outcomes for the fabricated 4-wheeled cars. The robot length and wheel radius were fixed at 200mm and 60mm respectively. A trial "failed" if the robot did not pass the pyramids within 120s while remaining upright.

Width Parameter value (in mm)	Number of trials passed	Number of trials failed	% of trials passed
65	0	10	0
95	7	3	70
115	3	9	25
180	0	12	0

5 DISCUSSION

Running simulations over multiple, perturbed environments helps bring the simulated results closer to real-world behaviour by accounting for the noise inherent in real-life situations. In this way, we can be more confident that our design is actually functional for a space of environments instead of merely one, and that its success was not due to simulator error. Having vastly reduced the viable candidates to target for real-life experiments, the engineer only needs to build a select few designs to test in real life.

These experiments allow the engineer to be reasonably confident that the design would work in the one known environment in real life, relying on the performance bounds that the simulations provided. In the case of a very narrow feasibility region in simulation, it might not overlap with reality at all, which may lead to not learning much. Nonetheless, that indicates that the simulation model is not sufficiently accurate for the task at hand. Testing edge cases in real-life experiments would indicate similar modelling deficiencies, for behaviors not observed in simulation. These can then be incorporated in future modelling iterations, tweaking or adding model parameters to better capture reality.

Another line of enquiry is exploring a wider variety of more interesting designs with more parameters and consequently more complex controllers. The current work omitted doing that, so as to keep the focus solely on the interplay between the design and environmental parameters that characterize the sim-to-real gap. For larger design spaces, more sophisticated search and optimization techniques than the simple grid search we used will be vital; our pipeline can serve as a black-box performance oracle for such techniques, as we demonstrate in [Yu et al. 2023]. In future work we plan to extend such automated design space exploration algorithms with feedback from real-world tests: for example, running a coarse search to narrow the space of feasible design parameters, testing a few designs rapidly in reality, then using these results to guide further exploration in simulation.

Along with increasing the complexity of the design space, we can also explore more complex spaces of environments. Our system's use of Scenic enables substantial scalability in this direction, as a more complex environment such as the field of randomized Gaussian hills shown in Fig. 9 are not much harder to model in Scenic than the simple environments we used in our experiments. Scenic abstracts away the complexity of managing distributions with complicated dependencies, and uses reasoning techniques to preserve the efficiency of sampling. Both of these are essential to constructing complicated environments that allow the evaluation of robots in scenarios closer to real-life deployment situations.

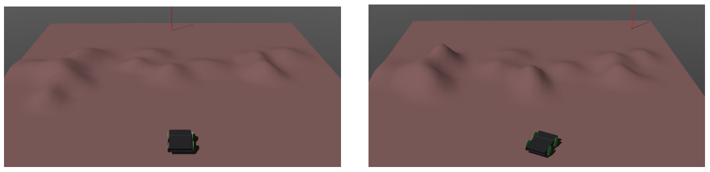


Figure 9: Two Instances of the Hill Field Experiments in Simulation

ACKNOWLEDGMENTS

This work was supported in part by DARPA contract FA8750-20-C-0156 (SDCPS)

REFERENCES

Reem J Alattas, Sarosh Patel, and Tarek M Sobh. 2019. Evolutionary modular robotics: Survey and analysis. *Journal of Intelligent & Robotic Systems* 95, 3 (2019), 815–828.

Joshua Auerbach, Deniz Aydin, Andrea Maesani, Przemyslaw Kornatowski, Titus Cieslewski, Grégoire Heitz, Pradeep Fernando, Ilya Loshchilov, Ludovic Daler, and Dario Floreano. 2014. Robogen: Robot generation through artificial evolution. In *ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems*. MIT Press, 136–137.

Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. 2021. A review of physics simulators for robotic applications. *IEEE Access* 9 (2021), 51416–51431.

Ruta Desai, Ye Yuan, and Stelian Coros. 2017. Computational abstractions for interactive design of robotic devices. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1196–1203.

Bill Earl. 2012. Adafruit PCA9685 16-Channel Servo Driver. *Adafruit Learning System* (2012).

Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. 2019. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 63–78.

Daniel J. Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2022. Scenic: A language for scenario specification and Data Generation. *Machine Learning* (2022). https://doi.org/10.1007/s10994-021-06120-5

Daniel J. Fremont, Edward Kim, Yash Vardhan Pant, Sanjit A. Seshia, Atul Acharya, Xantha Bruso, Paul Wells, Steve Lemke, Qiang Lu, and Shalin Mehta. 2020. Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World. In *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*. IEEE, 1–8. https://doi.org/10.1109/ITSC45102.2020.9294368

Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. 2018. Skaterbots: Optimization-based design and motion synthesis for robotic creatures with legs and wheels. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.

Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Florian Golemo, Melissa Mozifian, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, et al. 2020. Perspectives on Sim2Real transfer for robotics: A summary of the R: SS 2020 workshop. *arXiv preprint arXiv:2012.03806* (2020).

Sam Kriegman, Amir Mohammadi Nasab, Dylan Shah, Hannah Steele, Gabrielle Branin, Michael Levin, Josh Bongard, and Rebecca Kramer-Bottiglio. 2020. Scalable sim-to-real transfer of soft robot designs. In *2020 3rd IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, 359–366.

Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. 2015. Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–9.

Ankur Mehta, Joseph DelPreto, and Daniela Rus. 2015. Integrated codesign of printable robots. *Journal of Mechanisms and Robotics* 7, 2 (2015), 021015.

Ankur M Mehta, Joseph DelPreto, Benjamin Shaya, and Daniela Rus. 2014. Cogeneration of mechanical, electrical, and software designs for printable robots from structural specifications. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2892–2897.

Ankur M Mehta and Daniela Rus. 2014. An end-to-end system for designing mechanical structures for print-and-fold robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1460–1465.

Bhairav Mehta, Manfred Diaz, Florian Golemo, Christopher Pal, and Liam Paull. 2019. Active Domain Randomization.

O. Michel. 2004. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems* 1, 1 (2004), 39–42. http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf

Karine Miras and AE Eiben. 2019. Effects of environmental conditions on evolved robot morphologies and behavior. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 125–132.

Rodrigo Moreno, Ceyue Liu, Andres Faina, Henry Hernandez, and Jonatan Gomez. 2017. The EMeRGE modular robot, an open platform for quick testing of evolved robot morphologies. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 71–72.

Jai Hoon Park and Kang Hoon Lee. 2021. Computational Design of Modular Robots Based on Genetic Algorithm and Reinforcement Learning. *Symmetry* 13, 3 (2021), 471.

Fabio Ramos, Rafael Possas, and Dieter Fox. 2019. BayesSim: Adaptive Domain Randomization Via Probabilistic Inference for Robotics Simulators. In *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson (Eds.). https://doi.org/10.15607/RSS.2019.XV.029

Eivind Samuelsen and Kyrre Glette. 2015. Real-world reproduction of evolved robot morphologies: Automated categorization and evaluation. In *European conference on the applications of evolutionary computation*. Springer, 771–782.

Charles Schaff, Audrey Sedal, and Matthew R Walter. 2022. Soft Robots Learn to Crawl: Jointly Optimizing Design and Control with Sim-to-Real Transfer. *arXiv preprint arXiv:2202.04575* (2022).

Julian Whitman, Matthew Travers, and Howie Choset. 2020. Modular mobile robot design selection with deep reinforcement learning. In *NeurIPS Workshop on ML for engineering modeling, simulation and design*.

Wenzhong Yan, Dawei Zhao, and Ankur Mehta. 2022. Fabrication-aware design for furniture with planar pieces. *Robotica* (2022), 1–26.

Sheng-Jung Yu, Inigo Incer, Valmik Prabhu, Anwesha Chattoraj, Eric Vin, Daniel Fremont, Ankur Mehta, Alberto Sangiovanni-Vincentelli, Shankar Sastry, and Sanjit Seshia. 2023. Symbiotic CPS Design-Space Exploration through Iterated Optimization. In *Proceedings of the 5th Workshop on Design Automation for CPS and IoT (DESTION)*. IEEE/ACM CPS-IoT WEEK, San Antonio, Texas, USA.

Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. 2020. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–16.

Huang Zonghao, Quinn Wu, David Howard, and Cynthia Sung. 2022. EvoRobogami: co-designing with humans in evolutionary robotics experiments. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 168–176.

A JSON LISTING EXAMPLE FOR INPUT TO THE PIPELINE

The parameters of all the robot designs and environments under consideration are defined first, Then a list of all the experiments i.e the pair of robot and environments to be evaluated in the simulator are defined. These can also be supplemented with additional experimental parameters such as timeout and so on. This is encapsulated into a lightweight format that can be modified by a human during development, but also that can be easy to parse by a machine once the file is ready to be automatically generated (as an output of a design factory for example). The seamless nature of the pipeline is seen in the way it allows for simulations to be run in parallel,recovery and resumption of partially completed jobs, and running in containerization via Docker, all via the command line utility provided.

Listing 1: Example of using the JSON interface of the pipeline, defining the robot and environment parameters

```
{
  "experiments": {
      "3": {
            "veh_id": 5,
            "env_id": 4,
            "params": {"timeout": 400}}
        },
  "vehicles": {
        "1":
        {
            "type": "B_H_W",
            "params": [
                {"common" : {
                    "width":80,
                    "height":36,
                    "radius" : 30,
                    }
                },
            {"B": {
                "length":60
            }
            },
            {"H": {
                "length":20
            }},
            {"W": {
                    "tire_thickness":1,
                    "radius" : 30,
                    "length":40
                }},

        ]
}

  },
  "environments": {
        "4": {
            "scenic_filename": "middle_hill.scenic",
            "wbt_filename": "StandardMap.wbt",
            "seed": 42,
            "scenic_params": {"hillX": -0.3, "
    hillWidth": 1.25,"hillLength": 1.25,"hillHeight":
    0.3 } }
          }
}
```

B SCENIC ENVIRONMENT EXAMPLE

In Figure 10, we show a simplified version of one of our experiment environments.

This code first places two bumps left and right of a center point. We then create a robot at the start position facing towards the end position, which becomes the ego object. Finally we set up our termination conditions and recording statements. The termination conditions simply terminate the simulation if we get close enough to the target or if we reach a timeout. The record statements record several metrics once at the end of the simulation. Note that Scenic also supports recording values at the start of simulation, or at every timestep.

The code in Figure 11 augments the code in Figure 10 to add slight position mutations to the various locations in the program. Each of the positions has gaussian noise (standard deviation of 0.1) added to the x and y values. We could also easily mutate the dimensions of the speed bumps in a similar fashion.

We show a short example of an almost fully random program in Figure 12. Scenic takes care of the particulars, such as ensuring the bumps do not intersect.

```
# Declare several useful locations
bumpCenter = Point at (0,0)
startPos = Point at (0,1)
endPos = Point at (0,-1)

# Create two bumps left and right of the bump center.
Bump left of bumpCenter by 0.05,
    with width 0.1,
    with length 0.1,
    with height 0.03
Bump right of bumpCenter by 0.05,
    with width 0.1,
    with length 0.1,
    with height 0.03

# Create a robot, representing the ego
ego = Robot at startPos facing endPos

# Termination conditions
terminate when distance from ego to target <= 0.05
terminate after simulationTimeLimit seconds

# Record simulation data
record final (distance to target) as distanceToTarget
record final ego.consumedEnergy as consumedEnergy
record final simulation().currentRealTime as elapsedTime
```

Figure 10: A simplified Scenic program expressing an environment with two speed bumps.

```
# Mutate the points to slightly change their position
mutate bumpCenter, startPos, endPos by 0.01
```

Figure 11: A line modifying the simple Scenic program to have slightly random positions

```
# Declare several useful locations
startPos = Point at (0,1)
endPos = Point at (0,-1)

# Create 10 randomly sized bumps in a central region of the map
bump_region = RectangularRegion((0,0), 0, 2, 0.5)

for _ in range(10):
    Bump in bump_region,
        with width Range(0.05,0.1),
        with length Range(0.05,0.1),
        with height Range(0.01,0.03)

# Create a robot, representing the ego
ego = Robot at startPos facing endPos
```

Figure 12: A simplified Scenic program describing a field of bumps

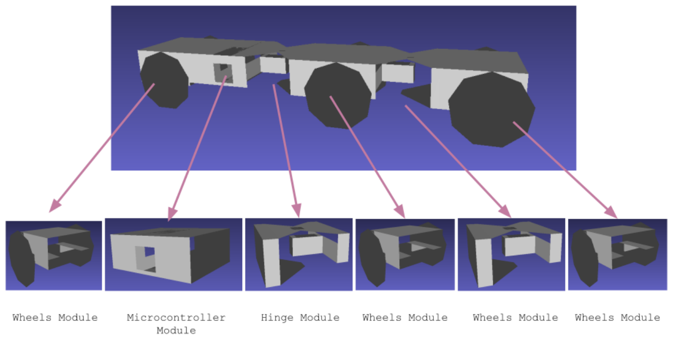


Figure 13: RoCo Example: Building a robot with a variety of modules

C ROCO DESIGN EXAMPLE

This example builds the robot shown in Fig. 13. The first step is to add all the necessary parameters for the robot (the same goes for the components):

```
c = Component("length", 100, paramType="length", minValue=40)
c.addParameter("width", 60, paramType="length", minValue=60)
c.addParameter("height", 36, paramType="length", minValue=36)
c.addParameter("radius", 36, paramType="length", minValue=13)
```

This is then followed by adding the modules. This section adds the Hinge Modules, which inherits certain parameters, from the parent robot, thus constraining it:

```
# Add vertical Hinge Module
c.addSubcomponent("vhinge0","vHingeModule", inherit="length width height ".split(), prefix=None)
c.addSubcomponent("vhinge1","vHingeModule", inherit="width height driveservo".split(), prefix=None)
```

Next the microcontroller module is added, also setting certain parameters such that it can be connected properly to the other modules:

```
c.addSubcomponent("brain0","BrainModule", inherit="width height ".split(), prefix=None,root=True)
```

Then the Wheel modules are added:

```
c.addSubcomponent("car0", "WheelModule", inherit="length width height driveservo radius".split(), prefix=None)
c.addSubcomponent("car1", "WheelModule", inherit="length width height  radius".split(), prefix=None)
c.addSubcomponent("car2", "WheelModule", inherit="length width height driveservo".split(), prefix=None)
```

Lastly, the connections are made for the components which can be done in a loop:

```
for i in range(0,4):
        c.addConnection(('car0','joinright%d'%(3-i)),('brain0','joinleft%d'%(i)))
        c.addConnection(('brain0','joinright%d'%(3-i)),('vhinge0','joinleft%d'%(i)))
        c.addConnection(('vhinge0','joinleft%d'%(3-i)),('car1','joinright%d'%(i)))
        c.addConnection(('car1','joinright%d'%(3-i)),('vhinge1','joinleft%d'%(i)))
        c.addConnection(('vhinge1','joinleft%d'%(3-i)),('car2','joinright%d'%(i)))
```

RoCo takes care of the details of generating these patterns and handling the parameters and interfaces of the components.