Towards Code-Aware Robotic Simulation

Vision Paper

John-Paul Ore, Carrick Detweiler, Sebastian Elbaum
Computer Science and Computer Engineering
University of Nebraska-Lincoln
Lincoln, NE, USA
jore,carrick,elbaum@cse.unl.edu

ABSTRACT

This vision paper explores the potential to dramatically enrich robotic simulations with insights gleaned from program analysis, and promises to be a key tool for future robot system developers to reduce effort and find tricky corner cases. Robotic simulations are a critical, cost-effective tool for developing, testing, and validating robotic software. However, most robotics simulations are intentionally unaware of how the code works. Our approach leverages two recent developments: 1) automatic program analysis that can semantically ground program variables and predicates in physical quantities like distance, velocity, or force; and 2) standardized simulation specifications that identify both what elements are simulated and also how they are simulated. Code-aware robotic simulation could enable robot system developers who increasingly rely on simulation to lower the cost and risk of system development by having access to richer simulation scenarios. We describe the approach using a detailed, step-by-step illustration for C++ using the Robot Operating System (ROS) and the Simulation Description Format (SDFormat), and identify key challenges to realizing this vision.

CCS CONCEPTS

• **Computer systems organization → Robotics**; • **Software and its engineering → Virtual worlds training simulations**; **Automated static analysis**;

ACM Reference Format:
John-Paul Ore, Carrick Detweiler, Sebastian Elbaum. 2018. Towards Code-Aware Robotic Simulation: Vision Paper. In *RoSE'18: RoSE'18:IEEE/ACM 1st International Workshop on Robotics Software Engineering, May 28-June 28 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3196558.3196566

1 INTRODUCTION

High-fidelity robotic simulators can reduce the cost and risk of developing systems that interact with the world. These simulators are governed by modeling specifications that can express a rich set of simulated environments and systems including obstacles, collisions, temperature, lighting, fog, and the granularity of space and time. Simulated scenarios are then composed of such models [18]. Deciding what to include and exclude in a simulation model and its parameters is paramount to its cost-effectiveness to explore scenarios of interest. While making such decisions, simulation users (often not the code developers) are usually aware of higher-level simulation goals [1, 7] but unaware of the physical elements referenced in code, neglecting the value of code-aware robotic simulation.

As shown in Figure 1, code-aware robotic simulation (CARS) is an automatic analysis that starts with two inputs: 1) physical attributes referenced in code and automatically detected by program analysis [11], such as variables that mean distances, velocities, and forces measured in units like `meters` $\mathtt{m}$, $\mathtt{m{\cdot}s^{-1}}$, or $\mathtt{kg{\cdot}m{\cdot}s^{-2}}$; and, 2) simulation specifications that encode both objects to be simulated as well as how they are simulated. The 'diff' of these two inputs is a new simulation specification that is enriched with code-aware concerns. This new simulation specification is then used as input to a regular simulator, but with richer specifications. CARS is enabled by two recent developments: 1) automatic program analysis able to identify physical attributes in code; and, 2) recent standardizations in simulation specifications like SDFormat [5] that standardize parameters and objects of robotic simulations, and externalize these parameters and objects from any particular simulation software platform, making the proposed approach more general.

Consider the simulated quadrotor in the bottom of Figure 1. The simulation executes the quadrotor control code as a 'black-box' while the physics simulator provides sensor readings that change as the quadrotor acts on the simulated world. The simulator has been intentionally designed to be unaware of how the quadrotor's control code works. However, program analysis can detect thresholds in the quadrotor control code triggering new behavior based on torque.

Figure 2 shows a snippet of actual quadrotor torque controller code. Lines 262-263 are part of a torque controller that bounds the commanded torque to within `±limits_.torque.x`. Although this quadrotor has several controllers, only this controller has explicit torque limits. Automatic program analysis can determine[1] that the variable `limits_.torque.x` is a real-world value with units $\mathtt{kg{\cdot}m^2{\cdot}s^{-2}}$. The code reveals not only that torque plays a role in the system behavior, but also that some ranges of torque values may be worth considering (predicate in line 261). These limits can matter when the quadrotor makes sharp turns or has a heavy payload. The simulation specification might indicate that this simulation does not contain sharp turns. Since sharp turns impact the system's

[1]For detail on how this is done for C++ code written for the Robot Operating System (ROS), refer to [11].

RoSE'18, May 28-June 28 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5760-9/18/05…$15.00
https://doi.org/10.1145/3196558.3196566

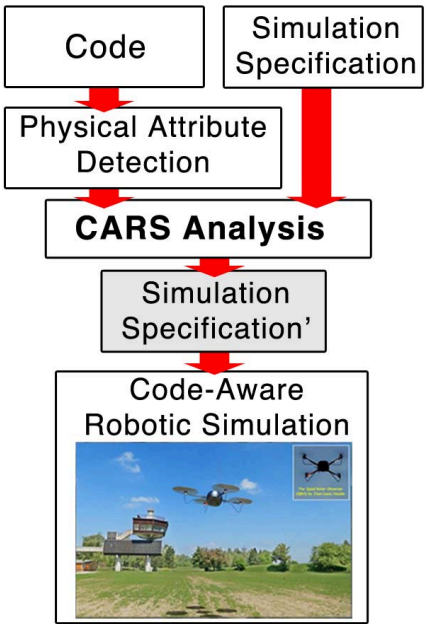

Figure 1: Overview of code-aware robotic simulation.

behavior, to create a rich set of simulations it should automatically be augmented with sharp turns to exercise the torque limits.

The envisioned approach consists of three key steps:

- Use program analysis to automatically detect physical code concerns relevant to simulation.
- Capitalize on recent standardizations of simulation description specifications to connect simulation concerns to code concerns.
- Use the difference between code concerns and simulation specification concerns to generate a new, enhanced simulation specification.

In the rest of this paper we discuss how robotic simulations are used and identify some of the key weaknesses that this work proposes to strengthen. We then give detailed examples of how program analysis can be used together with analysis of simulation specifications encoded in the simulation description format (SDFormat). Lastly, we identify the key challenges moving forward.

2 SIMULATION AND ITS LIMITATIONS

This section provides a brief discussion of how robotic simulations are used, and then describes some limitations in robotic simulation and how our proposed approach aims to address these limitations.

Robotic Simulations. The considerable expense and hazard of testing prototype hardware has spurred development of high-fidelity robotic simulators. Fundamentally, these simulators model physical processes and help 'close-the-loop' between sensing and acting, generating new perceptions available as a result of changes in the simulated world. In the case of robotic systems, these simulation tools seek to approximate a system's behavior in the world by decomposing a system and its environment into manageable pieces. These pieces interact according to detailed rules defined for the simulation, including concerns such as: are the materials deformable, do interactions behave differently because of temperature or pressure, are links rigid or elastic? These simulation rules or specifications have a significant impact on the performance, fidelity, and cost-effectiveness of the simulation, and are often tailored to the goals of the simulation. These simulators are often used to support goal-oriented, task-based simulations such as those used to test competitors in the DARPA Virtual Robotics Challenge [1, 7] and for providing a 'ghost' version of a real robot for teleoperation [10]. Human operators then specify a 'goal configuration' to a planner that then generates sub-goals and control inputs to impel the system toward the goal [16]. The value of these simulations is in revealing real-world failures.

Simulation Limitations. Ideally and with sufficient resolution, every failure in simulation corresponds to a real-world failure, and every real-world failure can be anticipated through simulation [2, 9]. In practice, the game is one of cost-effectiveness. Independent of the choice for simulation tool or model, a key common challenge is deciding what elements to include and at what resolution to include them. Clearly, more elements and resolution can improve simulation fidelity but not necessarily performance, and it can definitely increase the cost of developing and executing the simulation.

Although simulations exercise code, they are intentionally separated from it; missing in the simulation and the scenarios is the connection to code. As Rodney Brooks observed, "Simulation is doomed to succeed" [3], usually because the simulation fails to capture relevant real-world concerns [2]. To overcome these limitations, **we propose that robotic simulations need an awareness of code concerns.**

3 APPROACH

The goal of the approach is to automatically determine if physical concerns present in code are addressed in simulation specifications, and use this analysis to generate a richer simulation specification.

3.1 Approach Enablers

Simulation Description Standardization. To increase the generalization of the techniques we develop, we suggest leveraging standardized simulation specification languages, such as the recent SDFormat [5], V-Rep [14], and MuJoCo [17]. These languages describe not only the scenario but also the parameters governing how the simulator should treat the resolution or granularity of space and time, which is crucial when the system under simulation has high control rates or low error tolerances.

Semantic Grounding of Program Variables and Predicates. Our recent work and tool *Phriky Units* [11, 12] (*Phriky*) demonstrate a technique to semantically ground program variables and predicates to physical quantities like durations, distances, angular velocities, torques, or forces. *Phriky* works on C++ programs built with the Robot Operating System (ROS). ROS is a message passing middleware [13], and standard message structures for sensor values and motor commands are defined in shared libraries and commonly re-used to promote code portability [6]. For example, the shared library `sensor_msgs` defines a message `BatteryState` with an attribute `voltage`. *Phriky* includes a mapping between attributes of shared libraries and physical units for ROS messages. The mapping

```
261 if (limits_.torque.x > 0.0) {
262     if (wrench_.wrench.torque.x >  limits_.torque.x) wrench_.wrench.torque.x =  limits_.torque.x;
263     if (wrench_.wrench.torque.x < -limits_.torque.x) wrench_.wrench.torque.x = -limits_.torque.x;
264 }
```

Figure 2: Quadrotor controller code that considers torque. *source*: https://git.io/vXKpl

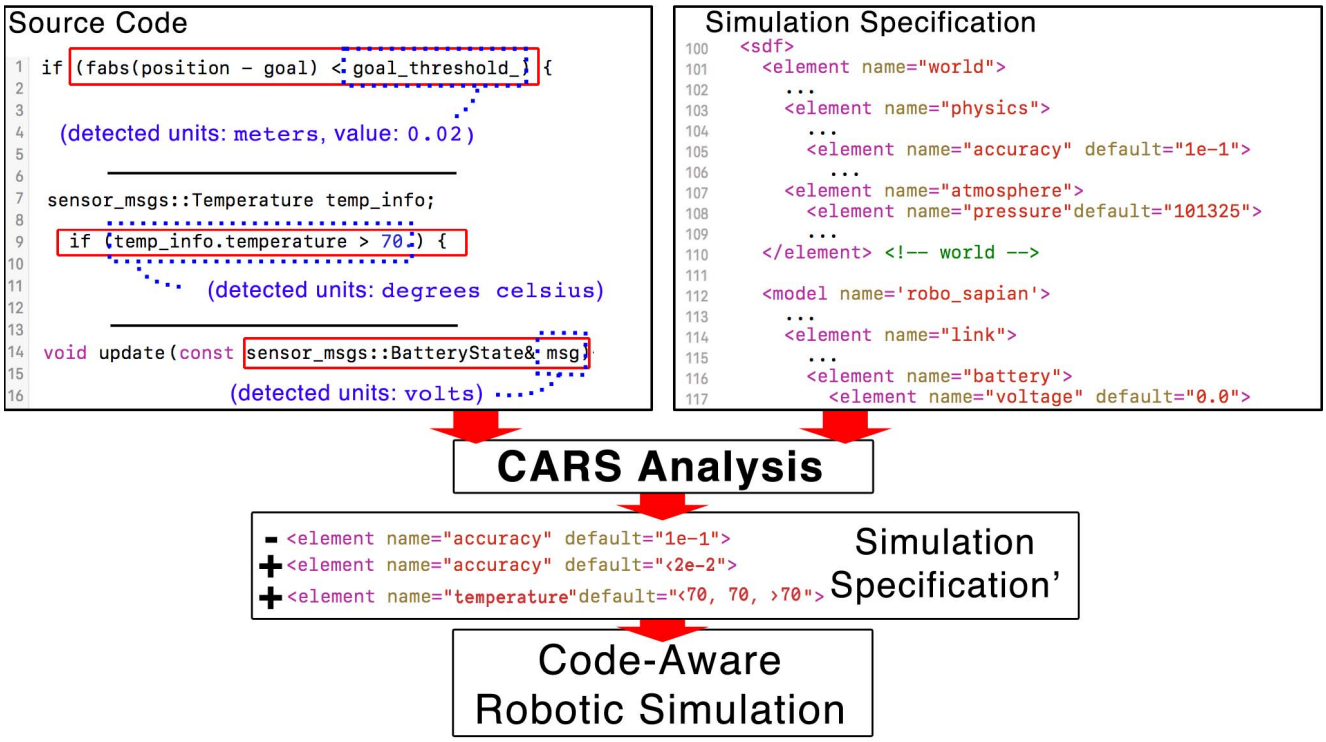

Figure 3: Simulation concerns informed by code.

encodes that the attribute `voltage` in `BatteryState` has the physical units $kg{\cdot}m^2{\cdot}s^{-3}{\cdot}A^{-1}$. *Phriky* can then propagate these physical units through the code using static analysis, to semantically ground predicates and variables in physical units.

3.2 Step-by-Step Illustration

Figure 3 shows a detailed example of the CARS approach. The left-hand-side of the figure shows code examples with concerns that are relevant to the simulation, and the right-hand-side shows XML elements from a simulation specification in SDFormat. Our first task consists of identifying physically relevant elements for a simulation, and encoding it in a target simulation specification language. Our static analysis technique [11] of the code can determine, for example, that the code on Line 14 of the left side of Figure 3 is concerned with a `BatteryState` type that maps to the physical unit of `volts`. An analysis of the simulation specification (a walkthrough of a specification in XML with standardized fields) can verify that the simulation addresses this kind of concern in Lines 116-117 of the right-hand-side of Figure 3.

However, the automated analysis of the code will also reveal that this system's behavior can change based on a temperature threshold on Line 9 of Figure 3 (recognized as a variable of a physical unit type `degrees Celsius`) so this should be a candidate for inclusion in the simulation specification. A slightly deeper data-flow analysis can take this further by including not just the candidate type but also the ranges of values to consider as per the predicate evaluation. As seen in the code on Line 9 of Figure 3, a predicate over the variable `temp_info.temperature` branches when the temperature exceeds 70 `Celsius`. Such predicate would help us instantiate simulation values or ranges of values (e.g., less than 70, 70, more than 70) as shown in the middle of Figure 3. This new specification element helps engineers enrich the simulation scenarios.

A more sophisticated analysis of the code is necessary to determine the space and time resolutions of the simulation, which are usually set through global standard parameters. For example, a robotic surgeon might require smaller resolution than a mining electric rope shovel [4], yet guidance for selecting that resolution is often disconnected from the code. To detect such resolution values we will start by searching for constants either in configuration, header, or launch files, and analyzing whether those constants are compared (directly or through propagation) with physical units of *distance* or *duration* base types. As an example, the code on Line

1 of Figure 3 shows a variable `goal_threshold_`, with physical units in `meters`, appearing within a predicate that determines if the system `position` is sufficiently close to `goal`. The magnitude of `goal_threshold_` determines a required lower bound for the spatial resolution or granularity of the simulation, one that serves a potential value of resolution to consider for this simulation. The code analysis determines that `goal_threshold_` is 0.02 *m*, and that the simulation specification should have at least this accuracy in spatial resolution, as shown in the enriched Simulation Specification' in the middle of Figure 3.

This example shows how semantic program analysis can reveal concerns that can be automatically compared with simulation specifications to identify ways to improve robotic simulations.

4 MOVING FORWARD

Our preliminary examination shows that system code often contains clues about elements of the environment that may impact the system behavior and hence are likely to be relevant to include in simulation environments. However, moving forward there are several challenges.

One challenge of CARS is correctly matching code concerns with corresponding elements in the simulation specification. Although an analysis of the code might correctly identify that battery voltage is a concern, there might be multiple matching elements in the simulation specification. It should be possible to start with an over-approximation of potential concerns and to flag ambiguities for further review.

Extending CARS beyond C++ or ROS is a challenge because it requires semantically grounding variables and predicates, either with annotations or language support. Other languages like Java have been grounded to physical units with extensive programmer annotations [19]. Beyond ROS, Simulink blocks can be linked to physical units with type systems like SimCheck [15], but this still incurs an annotation burden. Variables can also be grounded with specialized languages like F# that has unit support as envisioned by Kennedy [8]. Regardless of the language or robot architecture, CARS analysis proceeds the same after the variables and predicates are grounded.

Managing the complexity of the combinatorial space of code concerns might also be a challenge. In a large codebase, there might be many combinations of simulation parameters inferred by the code, requiring a separate simulation for each combination. Efficiently managing this complexity might require new heuristics for ordering simulations by a prioritization scheme.

Another challenge is determining how and when to deploy CARS in a systems' life-cycle. We believe CARS will not displace current techniques of robotic simulation that are *not* code-aware, but rather that CARS will augment and enrich simulation techniques, and that both will be a critical part of creating and testing robotic systems.

In summary, this vision paper explores how physical attributes in code can enrich simulation scenarios and parameters, so that system designers can better investigate the interplay of robotic systems and potential environments. If successful, this approach could dramatically improve the way robotic systems are explored through simulation.

ACKNOWLEDGMENTS

This work is supported by NSF CCF-1718040.

REFERENCES

[1] C. E. Agüero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. L. Rivero, J. Manzo, E. Krotkov, and G. Pratt. 2015. Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response. *IEEE Transactions on Automation Science and Engineering* 12, 2 (April 2015), 494–506. https://doi.org/10.1109/TASE.2014.2368997

[2] S. Balakirsky, S. Carpin, G. Dimitoglou, and B. Balaguer. 2009. *From Simulation to Real Robots with Predictable Results: Methods and Examples.* Springer US, Boston, MA, 113–137. https://doi.org/10.1007/978-1-4419-0492-8_6

[3] Rodney A. Brooks and Maja J. Mataric. 1993. *Real Robots, Real Learning Problems.* Springer US, Boston, MA, 193–213. https://doi.org/10.1007/978-1-4615-3184-5_8

[4] Matthew Dunbabin and Peter Corke. 2006. Autonomous excavation using a rope shovel. *Journal of Field Robotics* 23, 6-7 (2006), 379–394. https://doi.org/10.1002/rob.20132

[5] Open Source Robotics Foundation. 2016. SDFormat, a description language for Scientific Robotic Simulation. (2016). http://sdformat.org/spec http://sdformat.org/spec.

[6] Open Source Robotics Foundation. 2018. ROS Common messages. (2018). http://wiki.ros.org/common_msgs http://wiki.ros.org/common_msgs.

[7] John M. Hsu and Steven C. Peters. 2014. Extending Open Dynamics Engine for the DARPA Virtual Robotics Challenge. In *Simulation, Modeling, and Programming for Autonomous Robots*, Davide Brugali, Jan F. Broenink, Torsten Kroeger, and Bruce A. MacDonald (Eds.). Springer International Publishing, Cham, 37–48.

[8] Andrew Kennedy. 2009. Types for Units-of-Measure: Theory and Practice. In *Central European Functional Programming School - Third Summer School, CEFP 2009, Budapest, Hungary, May 21-23, 2009 and Komárno, Slovakia, May 25-30, 2009, Revised Selected Lectures.* 268–305. https://doi.org/10.1007/978-3-642-17685-2_8

[9] T. Kyriacou, U. Nehmzow, R. Iglesias, and S.A. Billings. 2008. Accurate robot simulation through system identification. *Robotics and Autonomous Systems* 56, 12 (2008), 1082 – 1093. https://doi.org/10.1016/j.robot.2008.01.005 Towards Autonomous Robotic Systems 2008: Mobile Robotics in the UK.

[10] Robin R. Murphy. 2015. Meta-analysis of Autonomy at the DARPA Robotics Challenge Trials. *Journal of Field Robotics* 32, 2 (2015), 189–191. https://doi.org/10.1002/rob.21578

[11] John-Paul Ore, Carrick Detweiler, and Sebastian Elbaum. 2017. Lightweight Detection of Physical Unit Inconsistencies Without Program Annotations. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017).* ACM, New York, NY, USA, 341–351. https://doi.org/10.1145/3092703.3092722

[12] John-Paul Ore, Carrick Detweiler, and Sebastian Elbaum. 2017. Phriky-units: A Lightweight, Annotation-free Physical Unit Inconsistency Detection Tool. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017).* ACM, New York, NY, USA, 352–355. https://doi.org/10.1145/3092703.3098219

[13] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3.2. Kobe, Japan, 5.

[14] E. Rohmer, S. P. N. Singh, and M. Freese. 2013. V-REP: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems.* 1321–1326. https://doi.org/10.1109/IROS.2013.6696520

[15] Pritam Roy and Natarajan Shankar. 2010. SimCheck: An Expressive Type System for Simulink. In *Second NASA Formal Methods Symposium - NFM 2010, Washington D.C., USA, April 13-15, 2010. Proceedings (NASA Conference Proceedings)*, César A. Muñoz (Ed.), Vol. NASA/CP-2010-216215. 149–160.

[16] R. Tedrake, M. Fallon, S. Karumanchi, S. Kuindersma, M. Antone, T. Schneider, T. Howard, M. Walter, H. Dai, R. Deits, M. Fleder, D. Fourie, R. Hammoud, S. Hemachandra, P. Ilardi, C. Perez-D'Arpino, S. Pillai, A. Valenzuela, C. Cantu, C. Dolan, I. Evans, S. Jorgensen, J. Kristeller, J. A. Shah, K. Iagnemma, and S. Teller. 2014. A summary of team MIT's approach to the virtual robotics challenge. In *2014 IEEE International Conference on Robotics and Automation (ICRA).* 2087–2087. https://doi.org/10.1109/ICRA.2014.6907140

[17] E. Todorov, T. Erez, and Y. Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

[18] Leon Žlajpah. 2008. Simulation in robotics. *Mathematics and Computers in Simulation* 79, 4 (2008), 879–897. https://doi.org/10.1016/j.matcom.2008.02.017

[19] Jian Xiang, John C. Knight, and Kevin J. Sullivan. 2015. Real-World Types and Their Application. In *Computer Safety, Reliability, and Security - 34th International Conference, SAFECOMP 2015 Delft, Netherlands, September 23-25, 2015. Proceedings (Lecture Notes in Computer Science)*, Floor Koornneef and Coen van Gulijk (Eds.), Vol. 9337. Springer, 471–484. https://doi.org/10.1007/978-3-319-24255-2_34