2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

SIERRA: A Modular Framework for Accelerating Research and Improving Reproducibility

John Harwell
Department of Computer Science & Engineering
University of Minnesota
Minneapolis, MN 55455
Email: harwe006@umn.edu

Maria Gini
Department of Computer Science & Engineering
University of Minnesota
Minneapolis, MN 55455
Email: gini@umn.edu

2023 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-2365-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICRA48891.2023.10161279

***Abstract*— We present SIERRA, a novel framework for accelerating development and improving reproducibility of results in robotics research. SIERRA accelerates research by automating the process of generating experiments from queries over independent variables, executing experiments, and processing the results to generate deliverables such as graphs and videos. It shifts the paradigm for testing hypotheses from procedural ("Do these steps to answer the query") to declarative ("Here is the query to test—GO!"), reducing the burden on researchers. It employs a modular architecture enabling easy customization and extension for the needs of individual researchers, thereby eliminating manual configuration and processing via throw-away scripts. SIERRA improves reproducibility of research by providing automation independent of the execution environment (HPC hardware, real robots, etc.) and targeted platform (simulator, real robots, etc.). This enables exact experiment replication, up to the limit of the execution environment and platform, as well as making it easy for researchers to test hypotheses in different computational environments. Though SIERRA is targeted at robotics research, its design makes it extendable to other fields.**

I. Introduction

Robotics researchers typically spend time on two types of tasks: science and engineering. Science tasks consist of developing new mathematical models, tools, or algorithms, while engineering tasks consist of configuring and running experiments for testing the new "thing", and some aspects of processing results. Frequently, it is only after science tasks have been nearly completed for a project that researchers consider the crucial issue of reproducibility. Challenges in this domain include repeatability (same team, same experimental setup), replicability (different team, same setup), and reproducibility (different team, different setup) [1]. Other factors include imprecise or missing documentation, and a high barrier to integration with existing solutions [2]. These issues are non-trivial; recent studies found that less than half of academic code from papers at recent AI conferences were runnable—not that they reproduced results, but that they ran at all—even with the help of the authors [3], [4].

The difficulties of reproducibility are further compounded by the nature of the tools used to meet the engineering needs of a project: ad-hoc toolchains and scripts that are thrown together on a per-project basis, and reused, modified, or duplicated on the fly. Usually, these toolchains and scripts are for dealing with "accidental complexities" [5]; that is, with engineering difficulties unrelated to the challenges of the science itself. Examples include: handling different configurations for specific platforms, such as ROS [6], or execution environments, such as SLURM [7] or TORQUE [8] clusters, or for processing and visualizing experimental results; e.g., statistically summarizing data and generating graphs. Clearly, this approach is prone to errors and to reinventions of the wheel between research groups and individual researchers.

We present SIERRA, an open source framework for automating engineering tasks to (a) accelerate the later stages R&D cycles, and (b) improve reproducibility. SIERRA automates the process of hypothesis testing and results processing, and handles details for platforms, execution environments, data processing, and results visualization to reduce the burden on researchers allowing them to focus on the "science" aspects of research: creative exploration of data, hypothesis testing, and experimental design.

Once researchers are confident in their code's correctness from initial small-scale testing, SIERRA accelerates research progress through automated hypothesis testing in an end-to-end fashion during the latter parts of the R&D cycle. Researchers using SIERRA need only inspect the final deliverables resulting from testing a hypothesis, and no longer have to manually "shepherd" a hypothesis of interest through the process. A comprehensive demonstration of SIERRA's capabilities can be found here[1], including how it supports exploration of independent variables and experimental data.

II. Motivation

Our motivation in presenting SIERRA is drawn from our experiences during a recent PhD thesis in our lab, and summarized in the following case study from the first author of this paper. John has developed a new distributed task allocation algorithm α for use in a foraging task where robots must coordinate to find objects of interest in an unknown environment and bring them to a central location. John wants to implement his algorithm so he can investigate:

- How well α scales with the number of robots, specifically if it remains efficient with up to 100 robots in several different scenarios.

[1] https://www-users.cse.umn.edu/~harwe006/showcase/aamas-2022-demo

TABLE I

HETEROGENEITY MATRICES. TOP: ALGORITHMIC. BOTTOM: COMPUTATIONAL AND ROBOTICS PLATFORM.

Algorithm	Has randomness?	Outputs data in?
α	Yes	`CSV`, `rosbag`
β	Yes	`rosbag`
γ	No	`CSV`, `rosbag`

Research Phase	Computational Platforms	Robotics Platforms
Initial development	Laptop	ARGoS
Scalability experiments	SLURM, TORQUE clusters	ARGoS
Transitioning to ROS	Laptop	ROS, Gazebo
Real robot experiments	Laptop	ROS, TurtleBots

- How robust it is with respect to sensor noise.
- How it compares to other similar state of the art algorithms on a foraging task: β, γ.

John's research investigations can be broken into the following 5 stage pipeline: (1) Form hypotheses and generate experiments, (2) execute experiments, (3) process experimental data, (4) generate visualizations of processed results, and (5) perform comparative analysis on generated visualizations (i.e., "Does this graph show α is better than β, or not?"). John is faced with the algorithm heterogeneity matrix shown in Table I (top). John will perform initial algorithmic development targeting ARGoS [9], then transition to ROS+Gazebo [10] to simulate the TurtleBot [11] robot, of which he has several on hand. Finally, he will do experiments with a set of real TurtleBots to verify his simulation results. John has access to either a large TORQUE- or SLURM-managed cluster for simulations. In addition to algorithm heterogeneity, he is also faced with the computational and robotics platform heterogeneity matrix shown in Table I (bottom).

Because a unified tooling for the different computation platforms (local laptop, SLURM) and robotics platforms (ARGoS, ROS, TurtleBots), is not available, John will have to manually write scripts to reconfigure his code throughout his research investigations. Moreover, he will have to write additional scripts to handle different input formats (e.g., XML vs plaintext) or output formats (e.g., `rosbag`, `CSV`). Finally, as he executes experiments, he will need to create various visualizations (graphs, videos, charts, etc.) from the processed data, all of which will require additional scripting, which might not be reusable in the future.

Overall, John's experiences are emblematic of robotics research, and provide the main motivation for SIERRA: the need for better automation. From Table II, we note the following important insight: most stages contain substantial engineering tasks that are performed manually by researchers; these "accidental complexities" are frequently non-trivial, and slow down the actual research even with many toolkits available to help, such as `pandas` and `matplotlib`.

Existing automation for simulators only targets parts of the common research pipeline in Table II [12]; similarly for real robots [2], [13]. To the best of our knowledge no automation exists for ARGoS [9], Gazebo [10], and ROS1 [6] for hypothesis testing and results processing. The partial automation of Webots [14] done in [12] is a subset of SIERRA's capabilities.

SIERRA accelerates research cycles by allowing researchers to focus on the "science" aspects: developing algorithms and designing experiments to test them. SIERRA changes the paradigm of the engineering tasks researchers must perform from manual and procedural to declarative and automated. That is, from "Do these steps to run the experiment, process the data and generate graphs" to "Here is the environment and platform, the deliverables I want to generate and the data I want to appear on them for my research query—GO!".

Essentially, SIERRA handles the "backend" parts of research, such as: random seeds, algorithm stochasticity, con-

TABLE II

FIVE STAGE RESEARCH PIPELINE IN ROBOTICS RESEARCH FROM AN ENGINEERING PERSPECTIVE

Stage	Description	Current Practice
1. Experiment generation	A researcher designs an experiment to test something they have developed.	Researchers utilize custom and/or throw-away scripts or otherwise manually set parameters defining the experiments [5].
2. Experiment execution	The researcher runs batch experiments, collecting data about different aspects of the application.	Researchers use custom scripts to configure their execution environment and run their experiments on it. There is often tight coupling between the execution environment and the targeted platform in the scripts, which makes reuse difficult.
3. Experiment results processing	The researcher processes the collected data to generate statistical insights about the performance of their software, its limitations, and its strengths.	Researchers use libraries for processing experiment data, such as `pandas`. Scripts for analysis are frequently written for a specific research investigation and are not reusable.
4. Deliverable generation	The researcher generates visualizations from processed data, such as graphs and videos; deliverables are manually polished to make them camera-ready.	Researchers utilize custom and/or throwaway scripts to generate graphs from processed data using `matplotlib` or other toolkits.
5. Deliverable comparison	The researcher generates comparative visualizations of their work for inclusion in publications and reports.	Researchers utilize custom scripts to generate comparative visualizations.

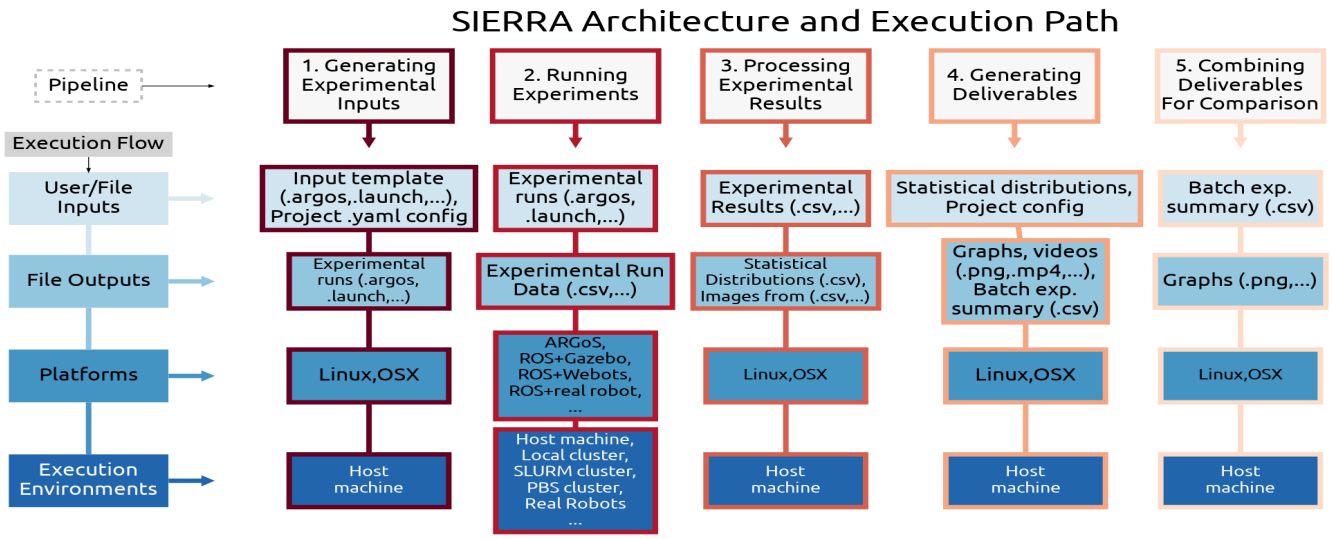

Fig. 1. Architecture of SIERRA, organized by pipeline stage, implementing the research pipeline shown in Table II. Pipeline stages are listed left to right, with an approximate joint architectural/functional stack from top to bottom for each stage. "( )" indicate areas where SIERRA utilizes plugins to perform environment specific actions in an agnostic manner. "..." indicates areas where SIERRA is configurable or extensible with plugins. "Host machine" indicates the machine SIERRA was invoked on.

figuration for a given execution environment or platform, generating statistics from experimental results, and generating visualizations from processed results. By using declarative specifications it eliminates manual re-configuration when changing platforms or execution environments, effectively decoupling platforms from environments; that is, any pair (platform, execution environment) can be selected in a mix-and-match fashion. It also removes the need for throw-away scripts for data processing and deliverable generation by providing rich, extensible faculties for those tasks. An architectural overview of how SIERRA implements the stages listed in Table II is shown in Fig. 1.

SIERRA is designed to have minimal barriers to adoption by researchers across disciplines through *in situ* integration with existing code implementations, while also being extensively customizable for advanced users; i.e., it is a "low threshold, no ceiling" tool [15], and meets the following criteria from [3], [5]: (a) has an easy to understand configuration, (b) has easy to reuse custom configuration and functionality across projects and researchers, (c) has plug-and-play facilities that do not require recompilation or repackaging to incorporate new functionality, and (d) has high-quality documentation and many examples. It is designed to be customizable, in order to support rapid adoption by researchers, and to accommodate potentially unknown future needs.

It accomplishes this in two ways. First, it is written in the python programming language, which is "write once, run anywhere" and has a human readable syntax. Second, it is organized into a reusable core and a plugin manager, which supports any number of plugins of any type that can be used to customize nearly every aspect of its implementation of the research pipeline shown in Table II. Thus, adding support for a new platform or execution environment is as simple as implementing a python interface, and placing the resulting file(s) on SIERRA's plugin path. Plugins can be written in any language; only the bindings must be written in python. Finally, SIERRA is open source, allowing researchers to modify it according to their needs.

III. SIERRA PIPELINE AUTOMATION

SIERRA is designed to automate research queries expressed in a researcher-defined command line syntax. Examples of research queries include: "How will this algorithm perform in this scenario with this range of inputs?", "What are the practical limits of this algorithm?", and "How does this algorithm compare to other similar algorithms?" Research queries are different than scientific hypotheses, which are possible explanations for an observed phenomenon or answers to a posed research query.

Each "value" of the independent variable in this range forms the basis for an *experiment*. In SIERRA terminology, this is the univariate *batch criterion* used to define a batch experiment. SIERRA also supports *bivariate* batch criterion, in which researchers are interested in how system behavior changes in response to the values of two independent variables jointly varying; in such cases, the state space for the batch experiment is a 2D grid instead of a one-dimensional line. SIERRA handles both types of batch criteria transparently.

The syntax for expressing research queries is entirely arbitrary, and can be set according to each researcher's needs; researchers also define parsers for their syntax. In our case study, John defined many different variables to pass as `--batch-criteria`, some of which are:

- `system_size.Log128`, representing univariate experiments with $\{1, 2, 4, 8, \ldots, 128\}$ robots.
- `task_alloc.Z100`, representing univariate experiments with one of a hard-coded set of task allocation

```
sierra-cli \
--platform=platform.argos \
--batch-criteria system_size.Log128 \
--exp-setup=exp_setup.T1000.K100 \
--n-runs=4
```

Fig. 2. A partial SIERRA command for a batch experiment containing 7 sets of experiments, one for each system size $\in \{1, 2, 4, 8, 16, 32, 64, 128\}$. The total # of ARGoS simulations is then 7 ($\log_2 128 = 7$) times –n-runs, for a total of 28 ARGoS simulations. Experiments will be 1,000 seconds long, with robot controllers running at 100 Hz. Many details such as which controller to run, the input file to use, etc., are omitted for clarity—for complete, functional example commands, see the online demo referenced earlier.

policies of interest $\{\alpha, \beta, \gamma\}$, with the number of robots fixed to 100 for all runs.

- `system100 saa_noise.C10`, representing bivariate experiments with $\{1, 2, 3, \ldots, 100\}$ robots and 10 different levels of Sensor and Actuator (SAA) noise applied to both robot sensors and actuators.

We provide details of the provided automation for each pipeline stage in the rest of this section, using our motivating case study from Section II as context. To more concretely demonstrate SIERRA's capabilities we will reference the partial SIERRA command in Fig. 2 throughout. We note that running stages $\{1, 2, 3, 4, 5\}$ from Fig. 1 in sequence is not required; any topologically ordered subset can be executed. For example, suppose John has just changed the configuration for what deliverables to generate. He could then instruct SIERRA to run stages $\{3, 4\}$ only by adding `--pipeline 3 4` to Fig. 2.

A. Stage 1: Experimental Input Generation

To generate the batch experiment, researchers provide a template XML file containing all configuration necessary to answer the research query. Any XML element SIERRA is not directed to change through a batch criterion or other plugin will remain unchanged, allowing researchers to set common configuration options that should remain the same for all experiments and all experimental runs. SIERRA currently requires that the template input file be XML, which was chosen because many major robotics platforms already support it, such as ARGoS, ROS, and WeBots. If a researcher wants to target a platform that does not support XML, SIERRA's modular architecture makes it easy to do so.

The XML template input file is modified according to the research query, with one experiment generated for each "value" of the independent variable(s). Each "value" may correspond to a single change to the template, such as `system_size.Log32` for changing the number of robots, or it can correspond to multiple changes, such as `saa_noise.all.C10` for changing the level of noise applied to multiple sensors and actuators in each experiment. SIERRA also supports changing additional parts of the template input file uniquely for each experiment, or uniformly for all experiments, providing unparalleled expressiveness to support research automation through experiment generation.

SIERRA provides support for research that requires multiple experimental runs in each experiment, which may be required due to randomness in the robots, e.g., imperfect sensors/actuators on real robots, or algorithm stochasticity. SIERRA manages this complexity transparently for researchers. For example, it can save generated random seeds, ensuring that if all platform and researcher code respects the random seed, then stage 2 of the pipeline is idempotent; that is, the same SIERRA invocation for stage 2 always produces the same outputs in each experiment.

In our case study, John can put common parameter options in the XML template file in a `<common>` section and then unique subsections for each algorithm: `<alpha>`,`<beta>`, etc. He could also give each algorithm its own XML file and duplicate the `common` section for each, according to his preference. To handle the stochasticity of α, β, he sets `--n-runs=4` as shown in Fig. 2. John can then generate experimental inputs for each platform from the *same* research query as easily as changing `--platform.{argos,ros1gazebo}`.

B. Stage 2: Running Experiments

After a batch experiment has been operationalized and written to the filesystem, SIERRA can execute all experiments in the batch, or some arbitrary subset. For example, if experiments #10–12 keep crashing, John can enable more debugging in his code and then re-run the problematic experiments by adding `--exp-range=10:12` to Fig. 2. We note that SIERRA's automation for this pipeline stage enables it to provide concurrent execution of experimental runs for platforms that do not support it natively, such as Gazebo [10], and to utilize intrinsic parallelism for platforms that do support it, such as NetLogo [15].

Experimental run inputs are executed using GNU parallel [16] on a selected execution environment and targeted platform (see online docs for SIERRA's current support matrix). SIERRA handles the necessary configuration for all supported platforms and execution environments, allowing researchers to transparently switch between them with minimal code changes; cross-compiling or re-architecting may be necessary depending on the nature of researcher code, and the selected platform. This effectively makes the question of "Where can I run my experiment?" logistical and declarative, rather than technical and procedural.

In our case study, after John is sure his code is correct, he can run medium-scale experiments on his laptop via `--exec-env=hpc.local`. Once satisfied with the results, he only has to pass `--exec-env={hpc.slurm,hpc.pbs}` to tell SIERRA to run his code on a SLURM cluster, or TORQUE cluster, as appropriate. He will only have to submit a job containing his SIERRA invocation with a given set of resources and SIERRA will figure out everything else. Similarly, for his TurtleBot experiments, he will only need to pass `--exec-env=robots.turtlebot3`

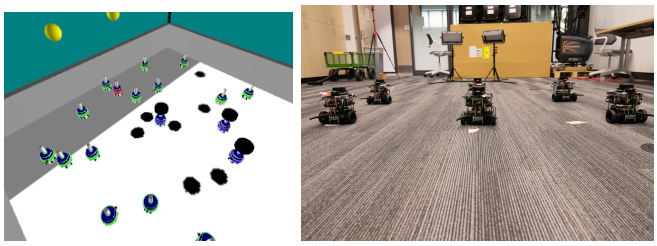

Fig. 3. *Left:* Screenshot from running an ARGoS experiment generated with SIERRA. *Right:* The six robot swarm of real TurtleBots used in foraging experiments, driven by SIERRA and using the same research query.

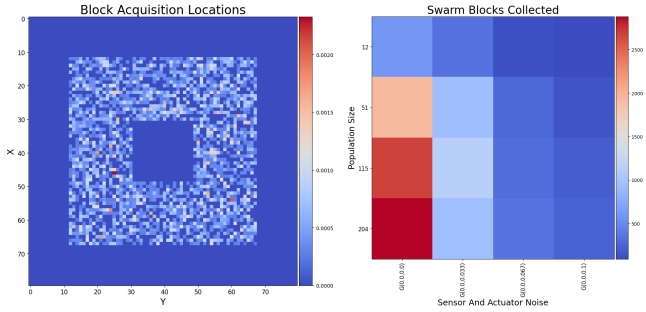

Fig. 4. Example of graphical deliverable generation in stage 4. *Left:* Intra-experiment heatmap showing the average locations where objects are in the environment (nest in the center). *Right:* Summary performance heatmap showing how the foraging behavior varies under a bivariate batch criterion with changing system size and different levels of sensor and actuator noise.

and `--platform=platform.rosrobot` to SIERRA. Examples of executing experiments generated and run using the *same* research query are shown in Fig. 3—SIERRA handles all of the messy details of platform switching.

C. Stage 3: Processing Experimental Results

After a batch experiment has been completed (or even part of it has), SIERRA can process outputs from arbitrary subsets of experiments in the batch using the `--exp-range`, analogous to stage 2. To process results, researchers specify which experimental outputs and types of statistics they are interested in, and SIERRA does the rest. SIERRA currently supports several types of results processing; additional types of processing can be added via plugins. First, statistical distribution generation across experimental runs for the selected experiments in the batch (intra-experiment statistics), as well as across experiments in a batch (inter-experiment statistics). This includes generating statistics suitable for plotting mean, 95% confidence interval, and box and whisker plots during deliverable generation during stage 4, specified via `--dist-stats={mean,conf95,bw}`. Second, converting output `CSV` files into heatmap images (see Fig. 4) that can be stitched together into videos during stage 4.

In our case study, John could generate statistics for all simulations by telling SIERRA that his data are stored in `CSV` format by adding `--storage-medium=storage.csv` to Fig. 2. If he wants standard deviation information, he could further add `--dist-stats=conf95`. For his TurtleBot experiments, he can write a plugin converting `rosbag` files into `pandas` dataframes, and tell SIERRA to use it.

D. Stage 4: Generating Deliverables

SIERRA has a rich model plugin framework. It allows researchers to generate data from first principles or from experimental results (or both), and plot the generated data alongside empirical results; this is commonly used for plotting model predictions. Models can be written in any language; only the bindings must be written in python. An example of this capability is shown in Fig. 5.

Visualizations of processed experimental results (deliverables) are included in published research, and will often need to be processed/tweaked multiple times to be camera-ready. In SIERRA, deliverables can include graphs or videos showing different aspects of the system's response to the research query; which graphs or videos are generated is controlled by YAML configuration, allowing researchers to easily disable deliverables not of interest. SIERRA's automation in this stage makes it easy to modify a specific graph or video. For some example graphs generated by SIERRA, see Fig. 4.

In our case study, suppose that John did not like the initial axes labels on the heatmap generated in Fig. 4. He could change these labels in his YAML configuration, and then re-run SIERRA to regenerate the graph.

E. Stage 5: Deliverable Comparison

After deliverable generation, multiple deliverables can be combined to provide side-by-side graphical comparisons; that is, SIERRA can take any data from two graphs of any type from any two batch experiments and replot them on a single figure. This comparison can take two forms. First, *intra-scenario* comparison, in which graphs from experiments evaluating different algorithms in the same context (scenario) are combined, as shown in Fig. 5(a). Second, *inter-scenario* comparison, in which graphs from batch experiments evaluating the same algorithm in different contexts (scenarios) are combined, as shown in Fig. 5(b). Such high-level comparisons are useful for demonstrating where/how a given algorithm is better or different than other methods. In our case study, suppose that John did not like the side-by-side heatmaps showing differences in algorithm performance, because they did not show which differences were statistically significant. He could ask SIERRA to generate a set of linegraphs instead, showing summary statistics such as confidence intervals or box and whisker plots graphically.

IV. Discussion

We have given a brief tour of some of SIERRA's features using a motivating case study to show how SIERRA can be used to address two pressing needs in robotics research simultaneously: accelerating R&D cycles through automation of engineering tasks, and increasing reproducibility of results. Specifically, John used SIERRA to generate, run, and process 100s of GB of simulation results autonomously, making repeated R&D cycles and figure generation for [17] seamless; i.e., a "set-it-and-forget-it" mentality for going from raw inputs to camera-ready graphs. The transition

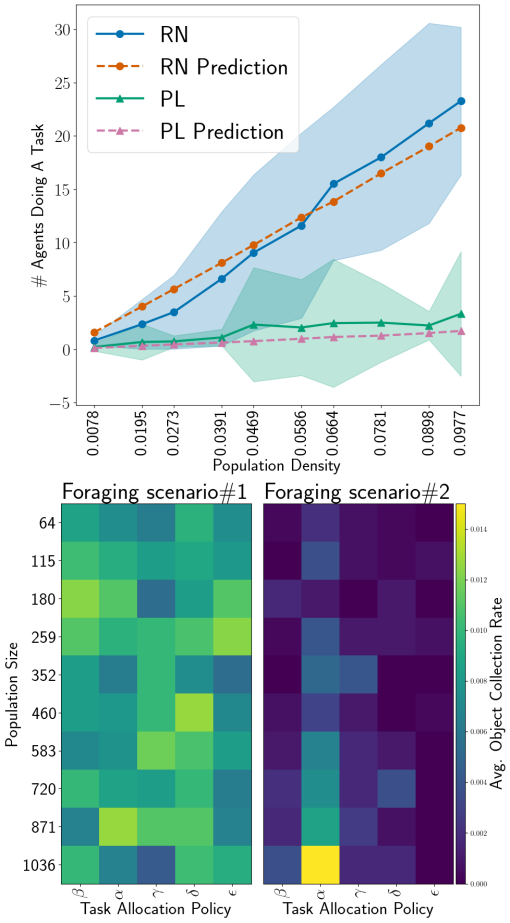

Fig. 5. Example of graphical deliverable comparison using both univariate and bivariate batch criteria. *Top:* Inter-scenario comparison of a single algorithm (α). An analytical model was developed for α, and the resulting predictions plotted alongside actual data. *Bottom:* Intra-scenario comparison of several task allocation algorithms on two different scenarios.

from ROS+Gazebo to ROS+TurtleBots was equally seamless: SIERRA handled all of the messy details of platform switching, enabling John to focus on investigating research questions. We believe that adoption of a tool such as SIERRA to provide a near-universal pipeline for robotics research that supports reproducibility and reusability is paramount to continuing to make meaningful progress as systems and approaches become more complex. Finally, we note that his successful usage of SIERRA was predicated on his code's correctness; SIERRA does not provide verification support.

SIERRA was developed for robotics research with the needs of robotics researchers in mind, so its *direct* applicability to other domains may be limited. It is currently restricted to use cases where experimental inputs are specified in XML; that is, as a single XML file containing all input parameters. This is not a major limitation in robotics, as most of the major platforms support XML inputs. Outside robotics, platforms of interest might not support XML. In the worst case, researchers would have to write XML bindings for their chosen platform to translate the generated XML inputs into a form their platform understands.

In addition, the translation from automating robotics research to broader intelligent systems research involving machine learning, deep learning, or other non-robot approaches may not fit the pipeline in Table II. For example, training runs for neural networks, or qualitative coding of Human-Robot Interaction studies may be difficult to fit into SIERRA's paradigm. In general, if researchers (a) mainly develop algorithms that do not contain elements of randomness, or (b) work in a field with a single platform/simulator that everyone uses, or (c) work on applications requiring only a single execution environment, then SIERRA may not provide benefits. Nevertheless, for fields for which SIERRA may be difficult to use or not provide benefits, we would hope that it would serve as inspiration to build similar tools.

A. History and community acceptance

SIERRA has been under development since 2016. Early versions of SIERRA have been used to automate experiments using ARGoS and ROS1 on TORQUE and SLURM HPC clusters, and with real TurtleBots at the University of Minnesota [17]–[19]. Version `1.3` is available as a stable release that comes with extensive documentation and tutorials. An earlier version was presented as a demonstration [20] at the AAMAS conference. The version presented here contains several new features not present in the earlier paper: extensible model plugin framework, integration with ROS+real robots, support for OSX as a host OS, among others. SIERRA is open source, and is available on PyPI[2]. SIERRA receives about 100 downloads/week, a sign of its relevance as a researcher tool.

V. Conclusions

We have presented SIERRA, a new tool that addresses the need for better automation of engineering tasks that many researchers perform. As a "low threshold, no ceiling" tool, it substantially lowers the barrier to collaboration between researchers across disciplines without compromising customizability for advanced users. Thus, SIERRA is not only relevant for the current needs of robotics researchers, but also for their future needs, and we strongly argue for its inclusion in any researcher's toolbox. Further improvements to SIERRA include: (a) removing the restriction that experimental inputs be specified in XML, (b) expanding the execution environments and platforms it supports natively, (c) improving configurability during deliverable generation to expose more of the underlying `matplotlib`, and (d) adding new visualizations such as parsing experiment logs to generate 2D or 3D "field" graphs of system behavior, messages exchanged, etc., live or post-hoc.

Acknowledgments: Partial support for this work was provided by the MnDRIVE initiative, the Minnesota Robotics Institute, and the University of Minnesota Informatics Institute. The Minnesota Supercomputing Institute provided computing resources.

[2]`https://pypi.org/sierra-research`

REFERENCES

[1] M. Mora-Cantallops, S. Sánchez-Alonso, E. García-Barriocanal, and M.-A. Sicilia, "Traceability for trustworthy AI: A review of models and tools," *Big Data and Cognitive Computing*, vol. 5, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2504-2289/5/2/20

[2] A. Pörtner, M. Hoffmann, S. Zug, and M. Knig, "Swarmrob: a docker-based toolkit for reproducibility and sharing of experimental artifacs in robotics research," in *Proc. IEEE Int'l Conf. on Systems, Man, and Cybernetics (SMC)*, 2018, pp. 325–332.

[3] O. E. Gundersen, S. Shamsaliei, and R. J. Isdahl, "Do machine learning platforms provide out-of-the-box reproducibility?" *Future Generation Computer Systems*, vol. 126, pp. 34–47, 2022.

[4] A. Bellogín and A. Said, "Improving accountability in recommender systems research through reproducibility," *User Modeling and User-Adapted Interaction*, vol. 31, no. 5, pp. 941–977, Nov 2021.

[5] A. Afzal, D. S. Katz, C. Le Goues, and C. S. Timperley, "Simulation for robotics test automation: Developer perspectives," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2021, pp. 263–274.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," ICRA Workshop on Open Source Software, 01 2009.

[7] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Springer, 2003, pp. 44–60.

[8] G. Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 8–es. [Online]. Available: https://doi.org/10.1145/1188455.1188464

[9] C. Pinciroli *et al.*, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, pp. 271–295, 12 2012.

[10] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[11] R. Amsters and P. Slaets, "Turtlebot 3 as a robotics education platform," in *Robotics in Education*, M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh, and D. Obdržálek, Eds. Springer International Publishing, 2020, pp. 170–181.

[12] M. Franchi, "Webots.HPC: A parallel robotics simulation pipeline for autonomous vehicles on high performance computing," arXiv:2108.00485v1 [cs.DC], 2021.

[13] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," arXiv:1609.04730v1 [cs.RO], 2016.

[14] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004.

[15] S. Tisue, "NetLogo: Design and implementation of a multi-agent modeling environment," in *Proc. Agents 2004 Conference*, Oct. 2004.

[16] O. Tange, "GNU parallel - the command-line power tool," *;login: The USENIX Magazine*, vol. 36, no. 1, pp. 42–47, Feb. 2011. [Online]. Available: http://www.gnu.org/s/parallel

[17] J. Harwell, A. Sylvester, and M. Gini, "Characterizing the limits of linear modeling of non-linear swarm behaviors," arXiv:2110.12307v2 [cs.RO], 2022.

[18] J. Harwell, L. Lowmanstone, and M. Gini, "Demystifying emergent intelligence and its effect on performance in large robot swarms," in *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 5 2020, pp. 474–482.

[19] J. Harwell and M. Gini, "Swarm engineering through quantitative measurement of swarm robotic principles in a 10,000 robot swarm," in *Proc. 28th Int'l Joint Conf. on Artificial Intelligence (IJCAI-19)*, 7 2019, pp. 336–342.

[20] J. Harwell, L. Lowmanstone, and M. Gini, "SIERRA: a modular framework for research automation," in *Proc. Int'l Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2022, p. 1905–1907.