Applying Evolution and Novelty Search to Enhance the Resilience of Autonomous Systems

Michael Austin Langford, Glen A. Simon, Philip K. McKinley, Betty H. C. Cheng
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan, USA
{langfo37, simongle, mckinley, chengb}@cse.msu.edu

***Abstract*—We investigate the integration of evolutionary algorithms and novelty search in order to improve the performance and resilience of autonomous systems. We have developed two tools for this purpose: Evo-ROS and Enki. Evo-ROS combines evolutionary search with physics-based simulations of autonomous systems whose software infrastructure is based on the Robot Operating System (ROS). Enki uses novelty search to discover operational scenarios that lead to the most diverse behavior in the target system. Combining these tools yields an automated approach to explore the operational landscape of the target system, identify regions of poor performance, and evolve system parameters that better respond to adverse situations. In this paper, we present results of a case study of the throttle controller on AutoRally, a 1:5-scale autonomous vehicle designed by researchers at Georgia Tech for the study of aggressive autonomous driving. Preliminary experiments demonstrate the ability of the proposed methods to identify and characterize input speed signals that cause the existing controller to perform poorly. The ability to identify these troublesome signals enables development of a control system capable of handling a wider range of conditions by autonomously switching among controller modes that are optimized for different conditions.**

***Index Terms*—autonomous systems, evolutionary computation, novelty search, search-based techniques, Robot Operating System, uncertainty**

I. Introduction

Autonomous cyber-physical systems are required for tasks where the burden of having a human operator is too high. With the absence of human supervision, effort must be made to mitigate sources of failure before the system is deployed or to include a capability for the system to adapt to unexpected conditions. This problem is particularly difficult due to the many sources of uncertainty in natural environments. Here, we describe autonomous systems as being *resilient* when they are capable of mitigating a wide range of adverse conditions, and this paper proposes an automated approach to assist software developers with making design decisions that result in more resilient autonomous systems.

Developing autonomous systems to meet software requirements at run time is challenging, because subsystems must interact with uncertain conditions in both internal mechanisms and the external environment. Techniques are needed to identify key scenarios that will have the most significant impact on system performance at design time to help developers form strategies that can mitigate potential sources of failure. Existing techniques either optimize the system to perform on a manual selection of scenarios, randomly generate scenarios, or use some heuristic-driven approach to create scenarios. Manual selection often requires expert knowledge for the problem domain and can be subject to confirmation bias [1]. Random generation may not be useful for discovering "corner-case" scenarios that cover small regions of the operational landscape [2][3]. Finally, iterative heuristic-driven approaches rely on objectives that may be difficult to define and can often lead to sub-optimal solutions when the operational landscape is not amenable to hill-climbing or gradient search [4].

This paper presents a two-phase evolution-based approach to improve the resiliency of an autonomous system. The first phase optimizes system settings for a given set of scenarios. The second phase identifies sets of scenarios that produce both extreme and *diverse* (i.e., mutually unique) system behavior. Using these two phases in tandem, the proposed approach can systematically improve autonomous systems with less reliance on *a priori* expert knowledge of the problem domain and less subject to bias that could cause harmful corner cases to remain unexposed.

We have created two techniques to accomplish this task, Evo-ROS [5] and Enki. Evo-ROS evolves a system's configuration based on its performance under simulation. Enki performs a novelty search [6][7][4] to discover simulation conditions that lead to unique types of system behavior. Both techniques are black box (i.e., agnostic to system details) and observe the autonomous system in a given simulation environment. The proposed framework is generalizable, but in this paper, we focus on a case study involving an AutoRally [8] autonomous vehicle (displayed in Fig. 1) and on the problem of tuning its throttle controller to match target speeds. Through experimenting with the throttle controller, we aim to answer the following research questions:

RQ1.) Can Evo-ROS evolve resilient systems?
RQ2.) Can Enki discover more challenging test scenarios to identify weaknesses in a system, when compared to random techniques?
RQ3.) Can Evo-ROS use diverse scenarios from Enki to evolve more resilient systems?

Preliminary results from our experiments demonstrate that the proposed approach can successfully evolve a more resilient autonomous system, evidenced by the fact that we have been

(a) Michigan State's physical AutoRally platform (under construction).

(b) Gazebo-simulated AutoRally platform, viewing from the front.

(c) Gazebo simulated AutoRally platform, visualizing Lidar field of sensing.

Fig. 1: Physical and simulated versions of the AutoRally platform.

able to produce settings for the throttle controller that have, on average, less error when comparing actual speed to desired speed. The remainder of the paper is organized as follows. Section II elaborates on the background and related work to this paper. Section III summarizes the proposed framework. Section IV reviews results from the case study. Finally, Section V provides a concluding discussion with ideas for future work.

II. Background and Related Work

In this section, we briefly discuss background and related work in search-based methods with evolutionary computation, evolutionary robotics, novelty search, and adaptive controllers.

A. Search-based Methods with Evolutionary Computation

With Search-based Software Engineering (SBSE), various search techniques have been proposed to solve typical software engineering problems, such as finding an optimal system configuration or automatically generating test suites [9][10][11]. Many search-based tools have been based on Evolutionary Algorithms (EAs) [12]. These tools require software engineers to specify candidate solutions according to a *phenotype*, where each instance of a solution is described as a *phenome*. EAs then encode these phenome solutions into corresponding *genomes*, following a specified *genotype*. Through an iterative process, a population of one or more genomes is refined such that when decoded into their corresponding phenomes, better quality solutions are found. Genomes are manipulated through evolutionary operations such as *crossover* and *mutation*, and selected for preservation between generations by a *fitness* heuristic. For SBSE, fitness is typically a metric based on observable properties of the software system, such as code coverage or execution time. Example techniques include EvoSuite [13] and Sapienz [14].

B. Evolutionary Robotics

The field of evolutionary robotics (ER) [15] harnesses the open-ended search capabilities of EAs by using genomes to encode specifications of the robot's control system or even aspects of its morphology (external structure), allowing for evolution to be leveraged in the process of designing more resilient robots. An artificial genome specifies the robot's control system and possibly aspects of its morphology. Individuals in a population are evaluated with respect to one or more tasks, with the best performing individuals selected to pass their genes to the next generation. Simulation is typically used to evaluate individuals, greatly reducing the time to evolve solutions while avoiding possible damage to physical robots. From an engineering perspective, a major advantage of evolutionary search is the possible discovery of solutions (as well as potential problems) that the engineer might not otherwise have considered.

While it is common for EAs to be used at design time, they have also been used at run time to help the system adapt to unforeseen changes to either the robot's environment or even the robot itself. For example, Bongard et al. [16][17][18] developed the Estimation-Exploration Algorithm (EEA) to address the problem of recovering a robot's functionality in reaction to unanticipated damage or environmental changes. The scope and aim of our research differs from EEA. EEA aims to evolve a simulator to close the so-called "reality gap" [19][20], whereas the proposed framework in this paper aims to use novelty search to automatically discover limitations for the evolved system configuration. Furthermore, an objective of EEA is to adapt a robot's controller to the current operational conditions at run time, whereas the goal of this paper's framework is to produce a system that is more resilient to a broader range of operational conditions at run time rather than reactive to specific run time conditions.

The work described in this paper extends the research conducted by Clark et al. [21] on discovering execution mode boundaries for adaptive controllers. Clark et al. describe a method for automatically enhancing and discovering the boundaries of an adaptive controller for robotic fish. In their work, they describe a mode discovery algorithm that evolves a controller to a fixed set of scenarios and then generates new scenarios to add to the set for another round of evolution. Two different scenario selection methods were considered, but both approaches can be considered adaptive random techniques, where scenarios are randomly generated from a base scenario and selected based on different criteria. A major difference between the methods we describe in this paper and their work is that we use *novelty search* (discussed below) for generating new scenarios without any need for a pre-defined "base" scenario.

C. Novelty Search

In the context of EAs, Lehman and Stanley [6][7] have championed the idea of abandoning fitness objectives in favor of maximizing population diversity with novelty search. They argue that in many cases, a strictly objective-driven search can lead to sub-optimal solutions, where the greatest discoveries are often not the result of simply optimizing for some preconceived objective. Due to the highly non-convex nature of any interesting problem space, a greedy objective-driven search can often be drawn to local optima, and in cases where the global optimum is surrounded by drastically inferior solutions, greedy algorithms have no means for finding the best solution. As an alternative, novelty search aims to optimize *diversity*, determined by comparing individuals with their nearest neighbors, with respect to their phenotype. In conjunction with an EA, an archive can be maintained to record the most unique individuals from each generation to produce a collection with widely differing phenomes.

Novelty search has previously been explored for system analysis and testing. Ramirez et al. [22][23] developed a tool called Loki to discover undesirable behaviors in Self-Adaptive Systems (SAS). Loki monitors the state of a SAS over time and defines the behavior of the SAS as a vector of utility values, where utility is defined as a function of the SAS's state given a set of initial input conditions. Novelty search is used to discover input conditions that lead to the most diverse set of SAS behavior, allowing for the discovery of potentially undesirable behaviors. Enki is directly inspired by Loki's approach, where diversity is defined in terms of observed system behavior rather than system input and access to the target system's source code is not required. However, Enki extends the capabilities of Loki to allow application to a wider range of target systems, while providing the user more flexibility in terms of how the novelty search is conducted.

D. Controllers

Many components of cyber-physical systems are governed by *controllers* [24]. Controllers are responsible for monitoring and adjusting a system's state to ensure the system behaves correctly. More specifically, controllers monitor *process variables* and take corrective actions on *control variables* to ensure the system's output remains within expected limits. This paper includes a case study involving a Proportional-Integral-Derivative (PID) throttle controller. PID controllers are given a *reference signal* as input as the target value for a process variable, and an *error signal* is computed as the difference between the target value and the actual value from the process variable. In the case of a PID throttle controller, the process variable is the actual speed of the vehicle, the reference signal is the target speed, and the error is the difference between the target speed and actual speed. PID controllers attempt to minimize error by adjusting the control variable via three control terms (P, I, and D). In order for the controller to respond correctly, certain *tuning constants* associated with each of these terms (K_P, K_I, and K_D, respectively) must be tuned for the application. Improper values for these constants can lead to problems such as oscillation and overshoot.

III. Framework

The proposed framework integrates two major components to support the evolution of new system configurations and to discover limitations in current system configurations. Fig. 2 illustrates the dataflow between these components, and each component is described in this section.

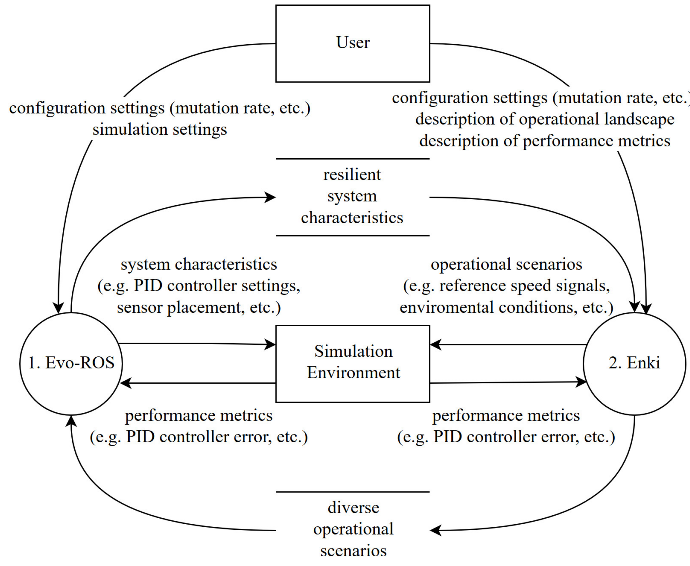

Fig. 2: High-level dataflow diagram of Evo-ROS and Enki.

A. Evo-ROS

Evo-ROS provides evolutionary search to ROS-based platforms [5]. ROS is widely used by the general robotics community, and Evo-ROS has been developed to bridge the gap with the narrower community of evolutionary robotics. Specifically, Evo-ROS defines the interface between an external evolutionary algorithm and the simulation environment used by ROS.

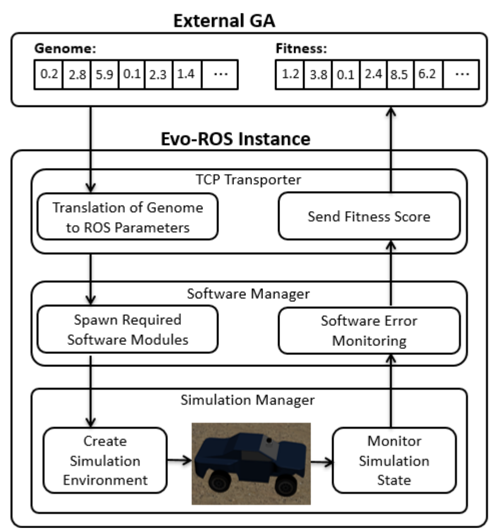

Fig. 3: High-level workflow diagram of Evo-ROS.

As shown in Fig. 3, Evo-ROS comprises three main modules: the transporter, the software manager, and the simulation manager. The transporter is responsible for the interchanging of data between the external EA and the Evo-ROS instance. The software manager is responsible for spawning all required packages and modules for the evaluation job. Lastly, the simulation manager is responsible for running and monitoring the platforms required tasks, during which it frequently logs the characteristics of the simulation.

An important feature of ROS is that code from a simulated system can be directly inserted into a physical robot. Therefore, compared to many evolutionary robotics platforms, Evo-ROS enables evolutionary search to be applied to state-of-the-art robots and full-sized autonomous vehicles. To address the execution time needed for multiple high-fidelity simulations, Evo-ROS provides an interface to parallelize jobs across multiple (physical or virtual) machines.

B. Enki

Enki is a technique for assessing the operation of a target software system to discover operational conditions that lead to unique, extreme, and possibly unexpected behavior. Enki uses an evolution-based novelty search algorithm to manage populations of individual scenarios. Each scenario is defined by a genome that encodes its operational conditions and is associated with a phenome that encodes the target system's behavior when exposed to the scenario. Enki evolves this population for a number of generations, using standard operations of evolution (i.e., recombination, mutation, and selection). However, unlike traditional evolutionary search methods, Enki does not guide the population toward a single fitness objective. Instead, a novelty archive is maintained across generations that ranks all individual scenarios in the current population with a novelty score and only archives the scenarios that exhibit the most diverse behavior from the target system. The novelty score for an individual scenario is determined by comparing its phenome to its nearest neighbors in the archive and averaging the distance. Thus, through evolution, Enki guides the search for scenarios outward in the operational landscape in terms of the type of behavior they produce from the target system. A benefit of this approach versus traditional evolutionary search is that the output is a set of individual scenarios that produce unique results, which may be adverse or favorable for the target system across a number of metrics, instead of producing only scenarios that maximize a single objective metric.

Fig. 4 illustrates the effect of Enki's novelty search process on the archived collection of scenarios. Each point corresponds to an individual scenario. Blue points correspond to scenarios currently in the archive, and gray points correspond to those that have been evaluated but not archived. All points are projected onto a 2D plane with distances scaled to match the relative distance between the scenarios' phenomes. In early generations, where the scenarios are closer to a random selection, the archived scenarios produce very similar behavior from the target system. As the search progresses, the archive "pushes" outward, demonstrating that each archived scenario is increasingly affecting the target system in different ways (i.e., becoming more diverse).

C. Simulation Environment

AutoRally [8] is an open-source platform designed and developed by researchers at the Georgia Institute of Technology (GeorgiaTech) as a test bed for various autonomous vehicle sensing and control methods. AutoRally is derived from a 1:5 scale RC truck (see Fig. 1) and has a top speed of 27 m/s (60 mph). Software for controlling the system is modular and highly customizable. An advantage for using AutoRally for research purposes is that it is smaller and less expensive than a full-sized autonomous vehicle yet uses much of the same control software and mechanical structures.

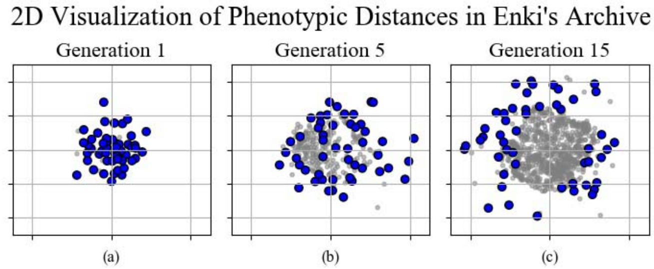

Fig. 4: A visualization of phenotypic distances between scenarios explored by Enki in Section IV-A. Blue points show archived scenarios. Gray points show all other scenarios evaluated. Archived scenarios increase in diversity over generations.

Simulation of AutoRally is managed by the Gazebo simulator, chosen for its support of complex environments and sensors modeled after many commercially available devices [25]. Furthermore, GeorgiaTech provides an accurate simulation model of the physical AutoRally platform within Gazebo (see Figs. 1b and 1c), closely matching all components and physics characteristics. With the capabilities offered by Gazebo and an accurate model of the vehicle, the reality gap between what is observed in simulation and the behavior of a physical system is expected to be minimal.

IV. Experiments and Results

This section describes a set of experiments to answer the research questions posed in Section I. The overall objective of this case study is to improve the AutoRally throttle controller's ability to handle a wider range of reference speed signals. AutoRally uses a PID throttle controller that is calibrated by adjusting four tuning constants, K_P, K_I, K_D and I_{max}. We designate a controller with the default values for these tuning constants as C_0. In our experiments, we evolved two separate sets of values for these tuning constants, which we have labeled C_1 and C_2. The values associated with these settings are listed in Table I.

A. Evolving a New Controller

For this case study, we evolve a controller configuration (C_1) with Evo-ROS that improves upon the default configuration (C_0); improvement is measured by the reduction in controller error when exposing AutoRally to a single reference signal (see Figs. 5a and 5e). Our configuration settings for Evo-ROS are listed in Table II. Upon completion, Evo-ROS was capable of reducing the mean-squared-error (MSE) between the actual speed of the vehicle and reference speed for the controller from 0.528 for C_0 to 0.018 for C_1. However, further assessment is required to show that C_1 also shows

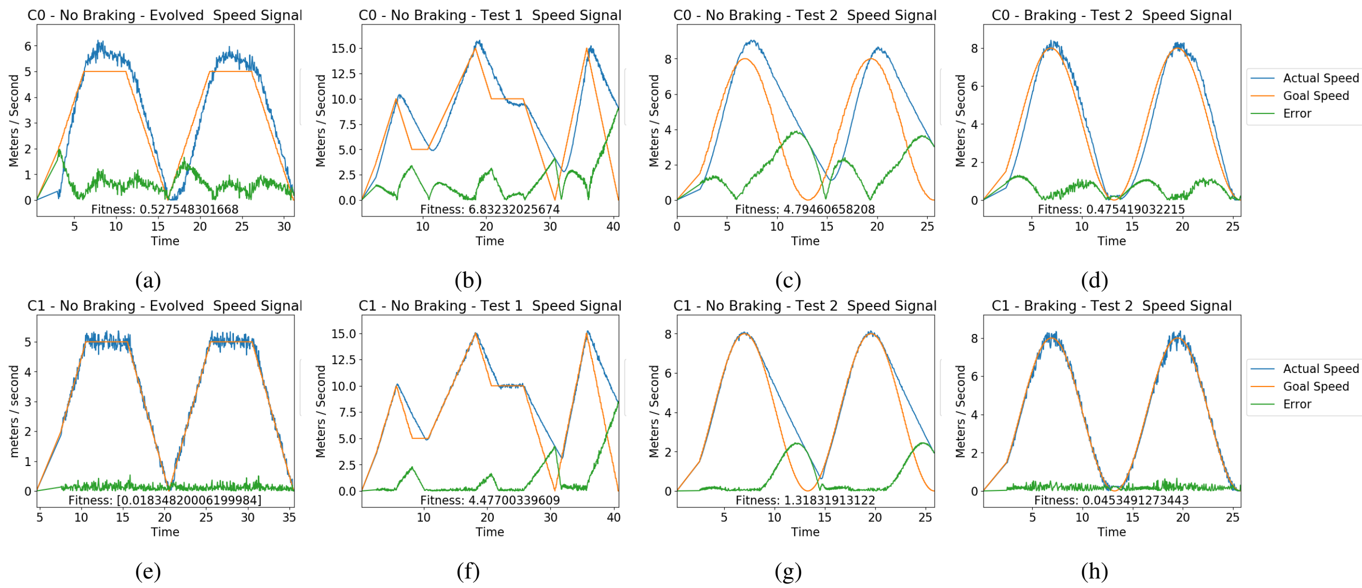

Fig. 5: Comparison of the default C_0 PID controller (top row) settings with the evolved C_1 PID controller (bottow row) settings over a variety of different speed signals. Subplots a) e) show the speed signal that the controller was evolved against, b) c) f) g) show performances against other random test signals and d) h) show performances on a test signal while braking is allowed.

improvement over a wider range of reference signals. Fig. 5 compares the performance of C_0 and C_1 on three other reference signals; C_1 was not evolved against these signals. It can be seen in Figs. 5b, 5f, 5c, and 5g that C_1 tracks acceleration in the reference signal better than C_0, but deceleration remains a challenge without the use of brakes. When braking is allowed, C_1 is able to track both the acceleration and the deceleration better than C_0.

TABLE I: PID controller settings.

PID Controller Settings				
	K_P	K_I	K_D	I_{max}
C_0	0.200	0.000	0.001	0.150
C_1	0.203	0.045	0.092	0.511
C_2	0.679	0.658	0.772	0.445

TABLE II: Evo-ROS configuration settings.

Evo-ROS Configuration Settings	
Generation Count	25
Population Size	25
Selection	Tournament - Size = 2
Elitism	False
Crossover	Two Point
Crossover Rate	0.5
Mutation	Gaussian - Sigma = 1.0
Mutation Rate	0.2
Fitness Metric	Minimize error (MSE)

B. Assessing the Controllers

To further assess the effectiveness of C_0, we have automatically generated sets of test reference signals. To evaluate Enki's ability to generate more challenging and/or diverse reference signals than a random generation method, we have generated $1,250$ reference signals using both Enki (S_{enki}) and a random method (S_{rand}). All signals were generated to cover a period of 60 seconds with speeds capped at a maximum of 10 m/s. The MSE was computed for the performance of C_0 on each reference signal.

The observed MSE distributions from both methods are displayed in Fig. 6. Regions are shaded by interquartile ranges, with green being the bottom quartile, blue being the middle two quartiles, and red being the top quartile. Signals from S_{rand} showed an average MSE of 1.047 (Fig. 6b) compared to an average MSE of 2.430 from signals in S_{enki} (Fig. 6a). Therefore, Enki is able to find more reference signals that cause the system to perform poorly. The random generation method creates reference signals by uniformly selecting values for each signal. When error-inducing reference signals occupy a small region of the domain of all possible reference signals, a uniform selection method is not likely to uncover such challenging signals. Instead, a random generation method will more likely result in a majority of reference signals that exhibit very similar errors. In contrast, Enki seeks out reference signals that produce unique types of error, and therefore, Enki can produce reference signals with a more uniform distribution of MSE, which may then lead to sets of more unique and challenging reference signals. Because our goal is to evolve a more resilient controller, we are interested in using Enki to discover the less common and more adverse reference signals.

To compare the effectiveness of C_1 to C_0 on the wider range of all possible reference signals, we used Enki to generate reference signals for C_0 and observed the error produced for each reference signal. Again, all reference signals were generated for 60 seconds and capped at 10 m/s. For Enki, the error was defined as the absolute difference between the actual speed of the vehicle and the reference signal. After running Enki with the settings listed in Table III, we generated $2,489$ signals (S_0), and Fig. 7 shows a comparison of the results from each controller. We found that the average MSE for C_0 was

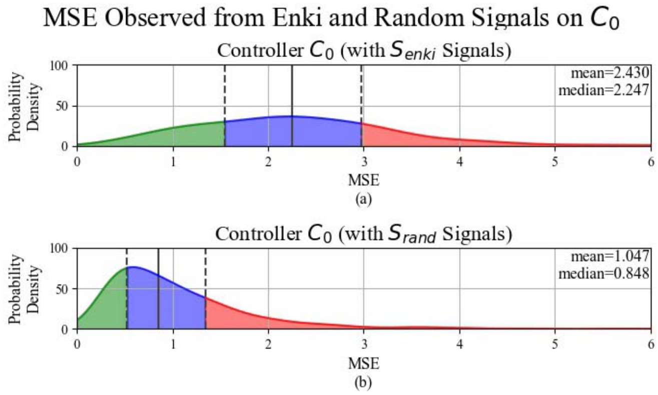

Fig. 6: Error observed on C_0 with signals from S_{enki} and S_{rand}. Subplot a) shows MSEs observed when using S_{enki}, and b) shows MSEs observed when using S_{rand}.

2.672 (Fig. 7a). When exposing C_1 to the same test reference signals, we found that the average MSE for C_1 was 2.250 (Fig. 7b). Each subplot shows a distribution of MSE observed by Enki for each controller. Since the whole distribution can be seen to skew left for C_1 when compared to C_0, C_1 has been observed to produce less error for any given reference signal. Therefore, our assessment with Enki further supports the claim that Evo-ROS did successfully evolve more resilient controller settings for C_1 than C_0.

TABLE III: Enki configuration settings.

Enki Configuration Settings	
Generation Count	50
Archive Size	50
Population Size	50
Selection	Tournament - Size = 3
Crossover	Single Point
Mutation Rate	0.25
Mutation Shift	0.2
Novelty Metric	Diversify error (absolute difference)

C. Further Enhancing the Controller

To explore whether the proposed framework can iteratively improve the resiliency of the controller, a second set of controller settings (C_2) was evolved. C_2 was created by Evo-ROS in a process similar to C_1 (see Section IV-A). However, for C_2, instead of only evolving the controller against the reference signal shown in Figs. 5a and 5e, the controller was also evolved against the top 5 most unique reference signals found by Enki when assessing C_1. After deriving the values for C_2, we evaluated it against the signals in S_0, and the results are displayed in Fig. 7c. We found that C_2 further reduced the average MSE to 2.148, with the overall MSE distribution skewed slightly more left. The reduced error exhibited by C_2 shows that by assessing the controller with Enki, we can find scenarios to further harden the controller to a wider range of reference signals.

D. Possible Limitations

Since this paper's framework relies on simulation, it is assumed that the simulator is capable of accurately matching reality. Any deviation in the simulator from the physical vehicle and its environment will be reflected in the scenarios generated

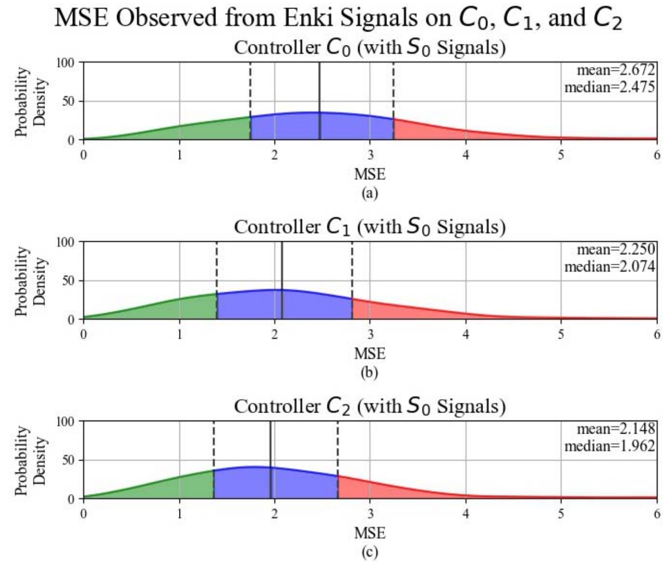

Fig. 7: Error observed on controllers when tested against signals from S_0. Subplot a) shows MSEs observed from C_0, and b) shows MSEs observed from C_1. b) shows MSEs observed from C_2.

by the framework. Our future work includes validating the results with our physical AutoRally platform, currently under construction. Additionally, for our experiments, we have only considered reference signals with a maximum speed of 10 m/s, and therefore, the comparative performance of the controllers may not be valid when the speed is allowed to exceed 10 m/s. Finally, the intent of this paper has been to assess the potential value of the proposed framework as a proof-of-concept, but since evolution-based techniques contain stochastic elements, multiple trials will be required to determine the statistical relevance of these results.

V. Conclusions and Future Work

We have demonstrated that the evolution-based techniques in this paper are capable of discovering more resilient configurations for an autonomous system with limited input by the user. In future work, we aim to introduce machine learning techniques that take the output from Enki and learn to infer execution mode boundaries for a given system configuration. One objective will be to construct an adaptive system that can switch between predetermined system configurations when the current configuration is no longer applicable to the operational context. Our goal is to apply this to a physical system that can independently and effectively detect changing adverse environmental conditions, such as sharp inclines or high slippage terrain, and safely transition into a better suited system configuration or mode to navigate efficiently through the adverse environment.

Acknowledgments

This work was supported in part by grants from the U.S. National Science Foundation and the Air Force Research Laboratory.

References

[1] G. Calikli and A. Bener, "Empirical Analyses of the Factors Affecting Confirmation Bias and the Effects of Confirmation Bias on Software Developer/Tester Performance," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10, 2010.

[2] W. J. Gutjahr, "Partition Testing vs Random Testing: The Influence of Uncertainty," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 661–674, Sep 1999.

[3] I. Ciupa, B. Meyer, M. Oriol, and A. Pretschner, "Finding Faults: Manual Testing vs. Random+ Testing vs. User Reports," in *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, Nov 2008, pp. 157–166.

[4] J. Lehman and K. O. Stanley, "Abandoning Objectives: Evolution Through the Search for Novelty Alone," *Evolutionary Computation*, vol. 19, no. 2, pp. 189–223, Jun 2011.

[5] G. A. Simon, J. M. Moore, A. J. Clark, and P. K. McKinley, "Evo-ROS: Integrating Evolution and the Robot Operating System," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2018, pp. 1386–1393.

[6] J. Lehman, "Evolution Through the Search for Novelty," Ph.D. dissertation, University of Central Florida, 2012.

[7] J. Lehman and K. Stanley, "Novelty Search and the Problem with Objectives," *Genetic Programming Theory and Practice IX*, 2011.

[8] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg, "AutoRally: An Open Platform for Aggressive Autonomous Driving," *arXiv preprint arXiv:1806.00678*, 2018.

[9] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating Software Engineering as a Search Problem," *IEE Proceedings - Software*, vol. 150, no. 3, pp. 161–175, Jun 2003.

[10] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based Software Engineering: Trends, Techniques and Applications," *ACM Comput. Surv.*, vol. 45, no. 1, 2012.

[11] P. McMinn, "Search-Based Software Testing: Past, Present, and Future," in *Proceedings of the 4th International Conference on Software Testing, Verification and Validation Workshops*, ser. ICST '11, 2011.

[12] M. Harman and J. Clark, "Metrics Are Fitness Functions Too," in *Proceedings of the 10th International Symposium on Software Metrics, 2004. Proceedings.*, 2004.

[13] G. Fraser and A. Arcuri, "EvoSuite: Automatic Test Suite Generation for Object-oriented Software," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11, 2011.

[14] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective Automated Testing for Android Applications," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016, 2016.

[15] D. Floreano, P. Husbands, and S. Nolfi, "Evolutionary Robotics," in *Handbook of Robotics*. Berlin: Springer Verlag, 2008.

[16] J. Bongard and H. Lipson, "Automated Robot Function Recovery After Unanticipated Failure or Environmental Change Using a Minimum of Hardware Trials," in *Proceedings. 2004 NASA/DoD Conference on Evolvable Hardware, 2004.*, Jun 2004, pp. 169–176.

[17] ——, "Active Coevolutionary Learning of Deterministic Finite Automata," *J. Mach. Learn. Res.*, vol. 6, pp. 1651–1678, Dec 2005.

[18] B. Kouchmeshky, W. Aquino, J. Bongard, and H. Lipson, "Co-evolutionary Algorithms for Structural Damage Identification Using Minimal Physical Testing," *Int. J. Numer. Meth. Eng.*, vol. 69, pp. 1085–1107, Jan. 2007.

[19] N. Jakobi, "Running Across the Reality Gap: Octopod Locomotion Evolved in a Minimal Simulation," in *Proceedings of the First European Workshop on Evolutionary Robotics*. Springer-Verlag, 1998, pp. 39–58.

[20] S. Koos, J. B. Mouret, and S. Doncieux, "Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers," in *Proceedings of the 2010 ACM Genetic and Evolutionary Computation Conference*, 2010, pp. 119–126.

[21] A. J. Clark, B. DeVries, J. M. Moore, B. H. C. Cheng, and P. K. McKinley, "An Evolutionary Approach to Discovering Execution Mode Boundaries for Adaptive Controllers," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016.

[22] A. J. Ramirez, A. C. Jensen, B. H. C. Cheng, and D. B. Knoester, "Automatically Exploring How Uncertainty Impacts Behavior of Dynamically Adaptive Systems," in *Proceeings of the IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '11, 2011.

[23] A. J. Ramirez, "Applying Evolutionary Computation Techniques to Address Environmental Uncertainty in Dynamically Adaptive Systems," Ph.D. dissertation, Michigan State University, 2013.

[24] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 2015.

[25] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, an Open-source Multi-robot Simulator," pp. 2149 – 2154 vol.3, 04 2004.