H2020-MSCA-ETN-2018-SAS. Proposal Number 812788

Safer Autonomous Systems

European Training Network on

Safer Autonomous Systems (SAS)

Deliverable 2.1 – Report on first experiment of the virtual worlds generation on a robotic simulator

Authors: Luca V. Sartori[1], Jérémie Guiochet[1], Hélène Waeselynck[1], Rob Alexander[2], Magnus Albert[3]

[1]LAAS-CNRS, Université Toulouse III – Paul Sabatier, FR

[2]University of York, UK

[3]Sick AG, DE

This project has received funding from the European Union's EU Framework Programme for Research and Innovation Horizon 2020 under Grant Agreement No. 812.788

Summary

The deliverable reports the research path that led to the first experiment in the field of software testing for robotics using simulations. It highlights what has been achieved and how, during the first part of the PhD thesis of ESR6. It introduces the main concepts of software testing and of the simulation. It details the faced and foreseen challenges related to guiding the selection of test cases to improve the discovery of faults. It describes the methods used to solve the problems, the methods proposed for the future, and gives an overview of the work done, the results obtained, and the goals reached. The results include two case studies, one academic and one industrial, used to demonstrate the feasibility of the proposed methods and the tool developed at LAAS. The tool and its integration is also described. The industrial case study was given to the ESR during a secondment at the Naïo company in Toulouse. The state of the art is given with an overview of the most important aspects of software testing and the methodologies used by other researchers. The most important points, the achievements, and the future directions are remarked during the conclusions.

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

TABLE OF CONTENTS

1 Introduction

This deliverable will describe the result of more than one year of research in software testing. As it is dense of content, the background section will help to contextualize the deliverable in the broad area of simulation-based testing for robotics.

1.1 Background

Autonomous systems are becoming increasingly popular and are deployed in numerous tasks. To improve their reliability and to avoid critical failures that impact safety, their testing aims at ensuring that their behavior and decisions are acceptable even in scenarios that have not been foreseen by the developers. As testing of these systems is usually done through field testing, which is costly and is limited in the reproducible scenarios, system-level pre-validation can be done in virtual worlds through simulation, to discover faults and fix them before deploying the system in the real world. However, there is no current standard procedure to conduct simulation-based testing and to ensure satisfying coverage of the most critical scenarios for the system under test (SUT). The objective of the PhD thesis, of which this deliverable is a part, is to improve and automate the steps of the simulation-based testing related to the generation and selection of test inputs, the exploitation of the results, and the incorporation of dynamic agents in the tests. It is worth noting that simulation-based testing cannot substitute field testing, as it is limited in how it simulates physics and it can be computationally expensive. However, the two testing approaches complement each other and can guarantee better testing of the system software. Since just a portion of test cases will lead to failures, an important issue is the selection of test cases to run in the simulation, as the search space, i.e., the set of possible test cases generated by the generation parameters, e.g., size of the world, number of obstacles, increases exponentially in relation to the number of parameters that model the virtual worlds. Consequently, executing all the test cases is unfeasible and not efficient. Moreover, revealing as many failures as possible in simulation is key to discover the causing faults, which can then be analyzed and fixed before the robot is field tested. On this subject, at LAAS-CNRS, there is a framework of ongoing work[14, 15, 12]. The work has focused on the generation of virtual worlds, has demonstrated that it is possible to find bugs in low-fidelity simulation, and has shown that it is possible to coarsely adjust the difficulty level of the worlds for the system under test (SUT).

1.2 Terminology

This section will explain the main concepts and terminology that will be used in the deliverable.

What is a simulator? A simulator in robotics is a software that enables a virtual reproduction of the system under test (the robot), the environment in which the system operates, and most of the conditions that occur in a real environment. For the system, the perception, decision, and control of the robot are simulated. For the environment, it is possible to decide the level of physical fidelity, ranging from a coarse representation, to a level close to reality, which demands more computational power. The simulator can be tailored to the specific needs of the developers and testers. There are various simulators, so the choice of which simulator to use for testing depends on the needs of the

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

developers and on the limits of the specific simulator. A simulator can be used in software testing to perform simulation-based testing.

What is software testing? Software testing is an investigation conducted to make sure that the software is as fault-free as possible and suitable to use. It is done to assess the quality of the software. Software testing can use a testing loop to enhance the probability of finding faults.

What is a testing loop? The testing loop is a structure composed by steps of the process of software testing, and in this deliverable it is specific to simulation-based testing. The testing loop is composed by the following steps: generation of test inputs, execution of the simulation, gathering of the logs, analysis of the logs, pass/fail verdict (oracle), selection of the parameters of the new test cases to run, and the loop begins again. An example of testing loop is visible in Fig. 1. In the testing loop there are test cases, which can be seen as test inputs.

What is a test case? In the context of the deliverable, a test case becomes the combination of assigning a mission to the system and generating a virtual world, where the mission will take place.

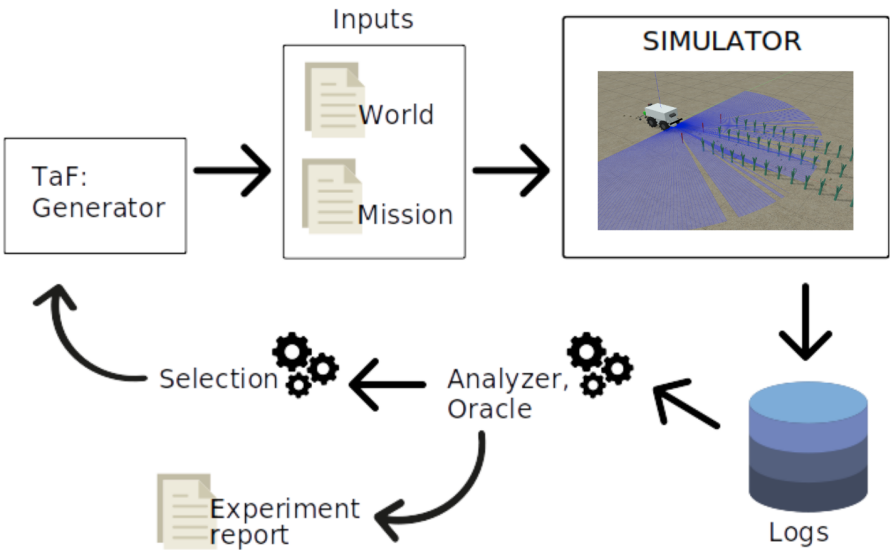

Figure 1: The testing loop.

1.3 Deliverable structure

The deliverable is structured as follows. Section 2 presents the challenges related to performing simulation-based testing for the scope of this deliverable. Section 3 gives an overview of the methods proposed to solve the challenges. The work done, the case studies, and the secondment are detailed in section 4. Section 5 shows the state of the art of software testing related to this deliverable. Lastly, section 6 summarizes the deliverable.

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

2 Challenges

In this section the research questions (RQs) are introduced to better frame the challenges related to simulation-based testing and more broadly software testing. It is only for context that the RQs are discussed, as the challenges are the focal point of this section. This PhD project aims at answering three research questions RQs, presented in the next paragraph, but since the focus of this deliverable is more on the first RQ, it will be the only one be discussed in detail. The research questions of this PhD are:

- **RQ1:** How to guide the selection of test cases in order to maximize the discovery of faults, taking into account test objectives and the analysis of previous runs.
- **RQ2:** How to exploit the results to characterize the high-level properties of the scenarios exhibiting failures.
- **RQ3:** How to incorporate a model of dynamic agents.

RQ2 and RQ3 are out of the scope of this deliverable and will be investigated in the future. Thus, the next session will talk only about the challenges of RQ1.

2.1 Generation of test cases (RQ1)

The space of possible test cases (the search space) that is possible to create is enormous. Thus, RQ1 focuses on finding an efficient technique (e.g., search-based, random testing, etc') to explore the search space, given the budget and time constraints. RQ1 faces the following challenges:

- How to exploit knowledge from previous test runs to guide the search of solutions (test cases), as it is preferable to avoid running test cases that have a low probability of failing, accelerating the convergence to cases that expose failures.
- How to accommodate multiple and conflicting objectives, because as their number increases, it is not possible to find a single solution that satisfies all of them, hence the need for a trade-off. The issue becomes to find a set of test cases where it is not possible to maximize more an objective without decreasing another one, i.e., the issue is to find the Pareto front.
- How to accommodate non-determinism, that is, the property of obtaining different results for repeated runs of the same test case, as in [14], due to the decisional capabilities of the robot. Additionally, different missions can have different degrees of non-determinism.
- How to select the algorithm among the many existing ones. When selecting the algorithm to analyze the search space, there are two key concepts to take into account: exploration—the diversity of solutions—and exploitation—the convergence to the optima of the search space. According to the search budget, different families of algorithms can be used based on their ability to show more diverse failures or more severe failures. It may be possible with a restricted time budget that a simple search algorithm is more efficient than a more sophisticated meta-heuristic algorithm at finding the optima.

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

In the next section, several techniques will be discussed to face the introduced challenges, to implement the test criteria, and to improve the testing loop.

3 Proposed methods

At the current stage of the thesis, for RQ1, some promising directions to maximize the number of critical test cases to run are random testing, combinatorial testing, and meta-heuristic, specifically evolutionary algorithms such as Genetic Algorithms, as it is possible to tune them to balance exploration and exploitation of the search space. Using Machine Learning to guide them is also a promising solution that could be explored in the future. Random testing is appealing because it is simpler to set up than other solutions, but still helps in discovering faults.

Regarding the **generation** step, the proposal is to start from the simpler method, random testing, and when the testing framework is in place, to start exploring additional techniques, in order of complexity to implement. In this case, combinatorial testing, in the form of pairwise testing or base choice testing, would significantly reduce the number of test cases to test, while maintaining a high fault discovery rate. After that, the inclusion of genetic algorithms would be the best choice to have diversity in the search space and try to reduce even further the number of test cases to run. The generation of test cases starts from a model with parameters describing the world and mission. The parameters can have constraints between themselves. To manage the generation from the model and the parameters into test cases, a tool developed at LAAS to generate test inputs has been selected as suitable. The tool has been developed with case studies similar to the ones faced during the time of the deliverable. This tool is called TaF and will be detailed in section 4. This tool and the techniques that have been mentioned in this paragraph are also linked with the selection step, as the results of the simulation can be used to guide the generation step of the next loop.

Physical fidelity and simulation. After generating the test inputs, the world will be loaded in the simulator, the mission will be assigned to the robot, and the simulation will start. It is advised to keep the physical fidelity to the minimum, as the focus is on testing the navigation of the robot and not the perception. Introducing realistic physics would be computationally expensive and would prolong the time to complete a simulation. So, for example, the grass or the wheel slip effect should not be included. An example of low physical fidelity simulation is visible in Fig. 2.

Logging. It is of paramount importance to record the log of the perception of the robot and the log of the simulator, the ground truth, as comparing the two logs is the only way during the analysis step to determine with accuracy what happened in case of a failure recorded in the simulation.

For the **analysis and pass/fail verdict**, a comparison of the robot and simulation logs must be performed. Deciding if the test has a fail or pass verdict should be based on many error detectors, each focusing on a simple property. It is proposed to follow the guidelines of the previous work at LAAS [15], which advises five broad classes of requirements to check: (i) requirements attached to mission phases, (ii) thresholds related to robot movement, (iii) absence of catastrophic events, (iv) requirements attached to error reports, v) perception requirements. From the verdict, the output parameters should be clustered using machine learning in order to converge to more critical test cases.

The **selection** step could also depend on the generation algorithm selected, as in the case of

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

genetic algorithms, where the output of the analysis is used to select the new parameters for the next generation. In general, it is advised to start with a selection that samples at random from the test cases that have yet to be explored, to have a well spread sampling, and later to introduce more sophisticated selection techniques to converge to critical cases or to preserve the diversity.

Multiple objectives. The objectives have to be weighted, with coefficients by, the developers based on their importance. An objective function can then be built, which permits to use the result of the test loop to create a minimization and maximization strategy to direct the search towards the objectives.

Non-determinism To compensate for the non-determinism of the simulation and the software of the robot, in the form of the decisions taken by the autonomous system, the proposed method is to repeat the same test case multiple times to obtain statistical significance.

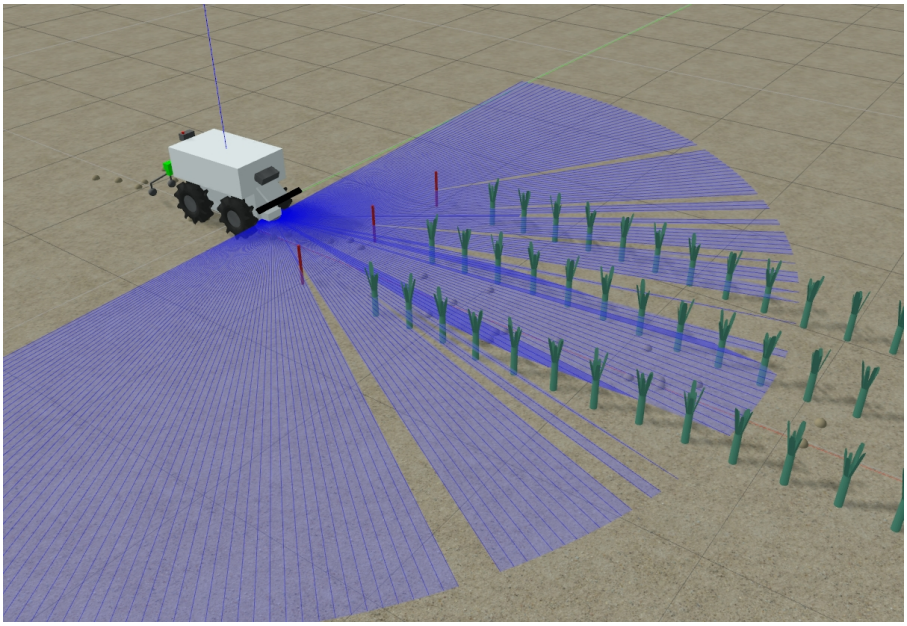

Figure 2: An agricultural robot performing weeding in a virtual field.

4 Work done

4.1 Preparation

The preparation started at the beginning of the PhD, where the ESR began the review of the bibliography related to software testing. Studying the testing methodologies lead to the exploration of the bibliography related to search-based software engineering and search-based software testing. From there the evolution strategies and genetic algorithms were studied. All these methods and algorithms are parts to the testing loop, which is the process used to test the robots software and assure the minimum amount of faults/bugs. The testing loop is composed by the following steps: generation of

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

test inputs, execution in the simulation, gathering of the logs for ground truth, analysis of the logs, pass/fail verdict, selection of the parameters of the new test cases to run, and the loop begins again. Visible in Fig. 1. The second part of the initial period was spent on studying the various simulators and testing tools that are usually used in the testing loop. One specific simulator, Gazebo [8] was then used in an academic case study.

Two case studies have been investigated before reaching this deliverable. The first one is the OSMOSIS[3], a simple academic case study of a robot for airport light inspection. The second case study is an industrial one related to the agricultural weeding robots of Naïo, a company based in Toulouse. One of the robots developed by Naïo is visible in Fig. 3.

Figure 3: The weeding robot Oz. Source: Naïo Technologies.

In the OSMOSIS case study the robot will follow the arcs of a graph to inspect the lights. The graph is composed by nodes that are connected by arcs and is laid out on a 2D google maps image representing the airport runway and apron. For this case study the ESR worked on generating new graphs that would represent different airports, because OSMOSIS, as of now, supports just the Blagnac Airport of Toulouse. The generation of new graphs was investigated starting from the extraction of graphs from real airports and by generating graphs of ”virtual” airports. The former method was deemed too complex, while the latter was working in generating planar graphs, but most of the resulting graphs had to be filtered, to obtain interesting graphs, as a percentage of the generated graph could not be considered airport-like, as the shapes were too complex or lacking details that a real airport would have. The ESR worked also on generating new missions. In this case a mission is a sequence of nodes, that the robot has to traverse, coupled with how to traverse the connecting arcs, so in light-inspection mode or not. The generation of new missions was done randomly, but the sequences of nodes had to be consistent with the properties of the graphs corresponding to the the runway and the apron of the airport. Thus, some constraints were enforced to obtain feasible and

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

coherent missions. After the experiments it was detected that, in very specific cases, the robot of OSMOSIS would go too far from an arc after a U-turn, violating the requirements of the mission set at the beginning.

The case of the robots of Naïo will be treated in the next subsection.

4.2 Secondment

To improve the testing loop, having an industrial case study is interesting because it has a broader scope than an academic one and faces problems that are relevant to the industry. Hence, the company Naïo Technologies supplied a case study related to the weeding robots that they are developing. The ESR did a secondment at the company, from February 2020 to April 2020, with half the secondment in smart working due to the lockdown restrictions. The overall objective of the secondment was to understand how to integrate the new testing methods and tools studied at LAAS in the workflow of the company, in order to improve the test cases, test scenarios, and faults discovery.

The goals were: 1) To create a draft of the framework for the testing loop and to integrate the TaF framework to generate test cases. The frameworks had to be integrated with the simulator and software used by the company. 2) To perform preliminary experiments with the draft framework and the software of the robots of the company. 3) To see if this approach could help to detect more faults/bugs in the software of the robots.

The goals were reached, as 1) the draft framework was created and TaF was integrated successfully, 2) some preliminary experiments were performed, and 3) the experiment led to the discovery of new faults, for example one related to the robot having a wrong perception of its position and the other related to the robot crashing on the crops due a series of event starting after entering a new row. Additionally, the secondment gave precious ideas on how to improve the workflow, reduce possible issues during the next secondment, and integrate in an easier way the methods and tools developed during this secondment.

The secondment time was spent as follows. The first part of the secondment was dedicated to familiarising with the software, techniques, and tools that the company is using. Then the ESR started writing the framework for automating the testing loop. Third step was to focus on how to generate the missions and virtual worlds. For this step the tool developed at LAAS, named TaF, was used and an external module was written to use the tool for the generation. After that, a part of the simulator software was modified to improve logging, in order to have a ground truth and detect errors in the perception of the robot. Lastly, various tests have been executed to assess the fault revealing power of the work done till that point. It is worth noting that the framework developed during the secondment does not cover the complete testing loop and is not fully automated. In the future it will be updated to fully automated all the steps of the testing loop.

As a time-consuming task was solving technical issues, they will be discussed in the next subsection, along with the related mitigation strategies.

Simulation issues During the secondment various technical problems were encountered. They will be discussed at a high-level and in a general manner, as it is important to highlight them because they are common when using simulators and reducing their impact could help the industry to transition faster to simulation-based testing. The first issue is related to the complexity of the software used, as they are dependant on different frameworks, libraries, and software developed by other companies

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

or developers. These external components will be updated with new versions, which could be not compatible with the other suites, or could deprecate part of the code written by the company. This is called ”dependency hell” and means that an update of one of the multiple software at the base of the simulation stack or robot software could introduce faults or make the simulator unusable.

Regarding the dependency hell issue, the best mitigation is to use isolated virtual environments like virtual machines or Docker containers. This would solve some of the dependency issues with the host machine and will make it easier to roll back in case of updates that create compatibility problems. Unfortunately, this is not always possible, leading to updates breaking the system and forcing the developers to spend time to fix the new issues.

Another problem is the complexity of setting up a simulator with the desired physical fidelity. A lot of resources and time are needed to configure a simulator from scratch and made it compatible with the robot, as the developers have to learn how to use new tools and software and they have to integrate them with the software of the robot. For example, a separated message system could be needed to make the simulator communicate with the software of the robot. Every addition or modification to the simulator has also to be tested to make sure that it did not introduce new faults. Modifying a simulator with which the developer is not familiar with, requires errors and trials that consume time.

The time spent to select the simulator has also to be taken into account, as there are multiple simulators available and each one has advantages and disadvantages. Testing the simulators for the desired use means setting up all the subsystems that are connected to the robot and simulator, which is not a trivial task.

The simulator could also introduce spurious faults that are not due to the robot, but are due to the limits of the simulator or the approximations and trade-offs made in the physics.

Lastly, writing the tests for the acceptance testing could require another external test automation framework, consuming more developers’ time and introducing another step where faults could be introduced, because tests could have logic faults too.

4.3 TaF integration

TaF (Test Automation Framework) is a framework for automated testing that has been developed at LAAS. The tool aims at solving the problem of generating a large amount of quality test cases while taking into account the constraints between the input parameters. Specifically, it is a generic tool that generates constrained and diverse test cases, starting from the user input in the form of an XML-based model. TaF then uses an SMT solver and random generation to output the test cases. The user can extend TaF and its generator module to suit his needs.

TaF has been used during the secondment to generate the worlds and missions for the weeding robot. In that case, a world is a field composed by rows of crops and the mission for the robot is to pass between the rows, where theoretically the weed should be. TaF has simplified the generation, as in this specific case an XML model of the field and mission was created, then TaF has generated a draft export module with diverse parameters that respect the constraints, e.g., a constraint could be that the length of the next generated row is ±10% the length of the previous row. It was then possible to extended the export module to have as outputs a world file and a mission file that could be used with the simulator. TaF could then be executed again, resulting in new world and mission files. This

This project has received funding from the European Union’s
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

permitted to generate worlds more varied and more complex that the few standard worlds used by the company, resulting also in longer and more difficult missions.

TaF was chosen because creating an ad hoc solution for the generation is time consuming and is not a flexible solution. As TaF is extendable, in the future it is planned to use it to implement generation and selection strategies based on pairwise testing, base choice testing, and genetic algorithms. The first two strategies are types of combinatorial testing, while genetic algorithms are metaheuristics inspired by natural selection.

5 Related work

This section contains the concepts that have been studied to make this deliverable possible. The subjects are listed with small explanations to better frame them in the context.

The testing community is researching how to model and generate content, what to test, with which methods to test, how to evaluate the outputs of the tests, and how to automate the procedures. The selection of the simulator has also to be taken into account, as different simulators have different capabilities. Moreover, it is possible to customize the physical fidelity of the simulator, with a higher fidelity requiring more computational time and a bigger effort in the development of the simulator. Since the search budget, i.e., the time allotted for testing, is a constraint imposed by the company developing the system, simulations can have low physical fidelity, prioritizing the number of cases executed per search budget. There are various robotics simulators such as GAZEBO [8] and MORSE [6], and game engines like Unity [1] and Unreal Engine [2] that have been used to conduct simulations: a comparison of which is given in [13]. In [5], Procedural Content Generation (PCG)—an automated technique borrowed from the video games field [17]—is used to generate 2D environments to test the path-following algorithm with collision avoidance of a mobile robot, demonstrating that bugs can be found even with low physical fidelity. [16] came to the same conclusion, showing that the majority of the bugs of the ArduPilot System are exposed by trivial conditions, and do not require the reproduction of complex physical phenomena to be revealed. Previous work at LAAS reinforces this idea, as in the study on the navigation software of the mobile robot Mana [15], where out of 33 bugs, the only bug that needed high fidelity to be replicated was related to mechanical vibration, which was not included in the simulation. The same authors in [14] investigate the system level testing of Mana. In that experiment, the generation of test cases uses a world model specified in a UML class diagram, which manages the high-level generation parameters—the size, percentage of obstruction, and smoothness—demonstrating that it is possible to coarsely control the difficulty level of the test cases.

Parametrized world models define a search space for the selection of test cases, suggesting the use of search-based software testing (SBST). SBST, a subset of search-based software engineering [7], uses meta-heuristic to find satisfying solutions to a test objective. The quality of the solution, i.e., the generated test cases, is assessed through a fitness function. SBST has been applied to the generation of test cases for autonomous systems, with the heuristics ranging from a simple random search [5] to genetic algorithms (GA) [11], or to a hybridization of GA and machine learning (ML) [4]. In the latter work, the hybridization aims to characterize subsets—critical regions—of the search space with a higher probability of containing critical test scenarios. Specifically, the proposed algorithm

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

consists of two nested loops, where the results of GA from the inner loop allow ML in the outer loop to identify combinations of parameters that are more likely to induce critical scenarios, yielding critical regions that deserve more exploration by GA, which refines the learned critical regions, and so on.

After the simulations are completed, assigning a pass/fail verdict to a test case is specifically challenging for autonomous systems, because the decisional aspects make it difficult to determine the right output for a given input, which is known as the test oracle problem. Moreover, the test oracle is usually based on a set of high-level requirements, like the absence of collision. Previous work at LAAS on the Mana robot identified five broad classes of requirements to check: (i) requirements attached to mission phases, (ii) thresholds related to robot movement, (iii) absence of catastrophic events, (iv) requirements attached to error reports, v) perception requirements. Other authors have considered metamorphic properties relating a test case to follow-up cases derived from it. In [9], metamorphic testing is applied to autonomous drones, by rotating a generated world to create follow-up cases. The drone should behave the same way in the original and rotated worlds. The comparison involves the path that the drone takes, the mission success, and a sequence of system states reconstructed from sensors values, giving a high-level view of the evolution of the system. Finally, some approaches consider a quantitative assessment of test results rather than a binary pass/fail verdict. For the cleaning agent in [11], quality thresholds are proposed, which are for example associated with battery levels. In [10], for cyber-physical system models, a quantitative test oracle provides a measure of the degree of satisfaction or violation of requirements.

6 Conclusions

This report summarizes the research and the work that led to the first experiments on the subject of the PhD thesis, which is simulation-based software testing for autonomous robots. The main research question of this deliverable is how to guide the selection of test cases to maximize the discovery of faults when there is a constrained search budget, so the goal is to find as many critical test cases as possible during a constrained time slot. The challenges related to this RQ are focusing on allowing the automatic world generation to be guided by test objectives and analysis of previous test runs, while taking into account non-determinism and multiple objectives. To see the feasibility of the proposed methods, they have been used in two case studies, one academic and one industrial. The second case study started with the secondment at the company that is developing the weeding robots. The main objective was reached, i.e., understanding how to integrate new testing methods and the TaF tool, developed at LAAS, in the workflow of the company. A draft of an automated testing framework for the testing loop was created and used to generate diverse test cases. The use of this framework showed that is is possible to find faults that remained undetected during the usual testing procedure of the company. This work is a continuation of previous research at LAAS. In the next future, the work will continue in line with the efforts at LAAS and will introduce more complex generation and selection strategies. After that, the focus will shift to RQ2 and RQ3, which are related to the analysis step of the testing loop and to another case study.

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

References

[1] *Unity*, https://unity.com/, accessed: 2020-04-25.

[2] *Unreal Engine | What is Unreal Engine 4*, https://www.unrealengine.com/, accessed: 2020-04-25.

[3] *What is OSMOSIS? | Open-Source Material fOr Safety assessment of Intelligent Systems*, https://osmosis.gitlab.io/, accessed: 2020-04-25.

[4] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter, *Testing Vision-based Control Systems Using Learnable Evolutionary Algorithms*, Proceedings of the 40th International Conference on Software Engineering (New York, NY, USA), ICSE '18, ACM, 2018, event-place: Gothenburg, Sweden, pp. 1016–1026.

[5] James Arnold and Rob Alexander, *Testing Autonomous Robot Control Software Using Procedural Content Generation*, Computer Safety, Reliability, and Security (Friedemann Bitsch, Jérémie Guiochet, and Mohamed Kaâniche, eds.), Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 33–44 (en).

[6] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, *Modular open robots simulation engine: MORSE*, 2011 IEEE International Conference on Robotics and Automation, May 2011, pp. 46–51.

[7] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang, *Search-based Software Engineering: Trends, Techniques and Applications*, ACM Comput. Surv. **45** (2012), no. 1, 11:1–11:61.

[8] N. Koenig and A. Howard, *Design and use paradigms for Gazebo, an open-source multi-robot simulator*, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), vol. 3, September 2004, pp. 2149–2154 vol.3.

[9] M. Lindvall, A. Porter, G. Magnusson, and C. Schulze, *Metamorphic Model-Based Testing of Autonomous Systems*, 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET), May 2017, pp. 35–41.

[10] Claudio Menghi, Shiva Nejati, Khouloud Gaaloul, and Lionel Briand, *Generating Automated and Online Test Oracles for Simulink Models with Continuous and Uncertain Behaviors*, arXiv [cs] (2019).

[11] Cu D. Nguyen, Simon Miles, Anna Perini, Paolo Tonella, Mark Harman, and Michael Luck, *Evolutionary testing of autonomous software agents*, Autonomous Agents and Multi-Agent Systems **25** (2012), no. 2, 260–283 (en).

[12] C. Robert, *First Insights into Testing Autonomous Robot in Virtual Worlds*, 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), October 2017, pp. 112–115.

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/

[13] Francisca Rosique, Pedro J. Navarro, Carlos Fernández, and Antonio Padilla, *A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research*, Sensors **19** (2019), no. 3, 648 (en).

[14] T. Sotiropoulos, J. Guiochet, F. Ingrand, and H. Waeselynck, *Virtual Worlds for Testing Robot Navigation: A Study on the Difficulty Level*, 2016 12th European Dependable Computing Conference (EDCC), September 2016, pp. 153–160.

[15] T. Sotiropoulos, H. Waeselynck, J. Guiochet, and F. Ingrand, *Can Robot Navigation Bugs Be Found in Simulation? An Exploratory Study*, 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS), July 2017, pp. 150–159.

[16] C. S. Timperley, A. Afzal, D. S. Katz, J. M. Hernandez, and C. Le Goues, *Crashing Simulated Planes is Cheap: Can Simulation Detect Robotics Bugs Early?*, 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), April 2018, pp. 331–342.

[17] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, *Search-Based Procedural Content Generation: A Taxonomy and Survey*, IEEE Transactions on Computational Intelligence and AI in Games **3** (2011), no. 3, 172–186.

This project has received funding from the European Union's
EU Framework Programme for Research and Innovation
Horizon 2020 under Grant Agreement No. 812.788.
https://etn-sas.eu/