robotics

MDPI

Article

A Case Study on Improving the Software Dependability of a ROS Path Planner for Steep Slope Vineyards

Luís Carlos Santos [1,2,*], André Santos [3], Filipe Neves Santos [1] and António Valente [1,2]

[1] CRIIS—Centre for Robotics in Industry and Intelligent Systems, Technology and Science, INESC-TEC—Institute for Systems and Computer Engineering, 4200-465 Porto, Portugal; fbsantos@inesctec.pt (F.N.S.); avalente@utad.pt (A.V.)
[2] ECT—School of Sciences and Technologies, UTAD—University of Trás-os-Montes and Alto Douro, 5001-801 Vila Real, Portugal
[3] HASLab—High-Assurance Software Laboratory, Technology and Science, INESC-TEC—Institute for Systems and Computer Engineering, 4710-057 Braga, Portugal; andre.f.santos@inesctec.pt
* Correspondence: luis.c.santos@inesctec.pt

Abstract: Software for robotic systems is becoming progressively more complex despite the existence of established software ecosystems like ROS, as the problems we delegate to robots become more and more challenging. Ensuring that the software works as intended is a crucial (but not trivial) task, although proper quality assurance processes are rarely seen in the open-source robotics community. This paper explains how we analyzed and improved a specialized path planner for steep-slope vineyards regarding its software dependability. The analysis revealed previously unknown bugs in the system, with a relatively low property specification effort. We argue that the benefits of similar quality assurance processes far outweigh the costs and should be more widespread in the robotics domain.

Keywords: robotics; safety software verification; agricultural robotics; path planning

check for updates

Citation: Santos, L.C.; Santos, A.; Santos, F.N.; Valente, A. A Case Study on the Software Dependability of a ROS Path Planner. *Robotics* **2021**, *10*, 103. https://doi.org/10.3390/robotics10030103

Academic Editor: Giulio Reina

Received: 5 July 2021
Accepted: 23 August 2021
Published: 26 August 2021

1. Introduction

The sector of agriculture has been changing over the years in response to the world's demands. The world's population has doubled in the last few decades and, by 2050, it is expected to double yet again. Estimates also indicate that 66% of the world's population will live in urban areas, and in 2014 this number was fixed at 55% [1]. Furthermore, human resources for agricultural labor are running low [2]. These statements indicate a need for increasing agriculture productivity, although in a way that is both sustainable and independent of intensive human labor.

The strategic European research agenda for robotics [3] states that robots will improve agricultural efficiency. A recent literature review found various approaches of robots in agriculture, especially for harvesting and weeding tasks [4]. This study infers that most approaches require further development and optimization. The terrains' harsh and unstructured environments present several challenges, especially in the steep slope vineyards located in the Douro Demarcated Region, a UNESCO Heritage place. These vineyards are unique, in that they grow along hills that block Global Navigation Satellite System (GNSS) signals and contain strong slopes, dangerous for robot navigation (see Figure 1).

An approach to the problem of robot navigation in uneven terrain with strong slopes was proposed in AgRobPP, a path planning system aware of the robot's center of mass [5,6]. Localization issues have been addressed with techniques to localize the robot independently of GNSS [7,8]. However, navigation is but one of many problems. Operating an autonomous robot in this region raises serious safety concerns; accidents may damage expensive robot equipment, centenary vine trees, or cause injury to human beings and animals. For obvious reasons, physical safety mechanisms, such as barriers, are not feasible in this type of terrain—especially not when vineyards may span up to 70 hectares (https://sogrape.com/tourism/quinta-do-seixo?slug=quinta-do-seixo, accessed on 22 August 2021). Ensuring safe robot operation is, thus, a responsibility of robotic software.

Figure 1. Typical Douro's steep slope vineyard.

The rise and standardization of middlewares and software ecosystems such as the Robot Operating System (ROS) [9] accelerated robotics research and development tremendously. ROS is a multi-lingual and open-source framework based on peer-to-peer architectures, consisting of several components (*nodes*) communicating via message-passing (*topics*, *services*) and shared variables (*parameters*). Still, robotic software is often developed by roboticists—people who typically have little background in software engineering best practices [10]. Fortunately, software engineering research is becoming more invested into dependability and quality assurance for robotics.

In this paper, we show the benefits of roboticists and software engineers joining forces. More concretely, our main contribution is a case study on improving the software dependability of AgRobPP, where we define a software analysis workflow that addresses various software development problems. This workflow is highly based on HAROS (https://github.com/git-afsantos/haros, accessed on 22 August 2021) [11], a framework specifically designed to analyze ROS software. HAROS contains a static analysis tool to verify the conformity of the developed code with programming style guides such as Google C++ Style Guide or ROS C++ Style Guide [11]. Besides, HAROS contains a framework to analyze the system behavior by checking if random inputs generate the expected system outputs [12]. Our report includes estimates of the required effort to apply this process, not only to demonstrate its efficiency and effectiveness but also to encourage the community in making similar techniques more widespread among roboticists. In particular, we show how, in a relatively short time, we were able to uncover behavioral bugs in the AgRobPP system.

As for structure, we start by comparing our approach to other case studies in robotics in Section 2. We present our case study subject, AgRobPP, in Section 3 and describe our analysis workflow in Section 4. Section 5 forms the core of the paper, where we explain how we analyzed and improved the system. Finally, Section 6 discusses our findings and lists some directions for future work.

2. Related Work

Applying formal methods and quality assurance processes to robotics software is not a novel concept. Several approaches have been adapted from general-purpose software engineering [11,13–16], while some were tailor-made for this particular domain [17–20]. Overall, with the increasing investment in robotics research and development, this topic is gaining more and more traction in the community. In this section, we list a number of relevant approaches to analyze the software of robotic systems and compare them to our own regarding application scope and required effort.

In general, we have observed that, oftentimes, teams approach software quality assurance in robotics with a single technique as their focus, such as model checking or runtime verification. The various approaches can be roughly categorized as being based on formal methods versus being informal or based on static analysis versus dynamic analysis. In contrast, we have employed a process that integrates multiple techniques into a single workflow.

At one end of the spectrum, we have approaches based on dynamic analysis, mostly runtime verification. This is a recurrent topic in the literature, e.g., as seen in [17–20], which demonstrates the efficiency and effectiveness of this technique. It operates on concrete implementations during normal operation and requires only an initial property-specification investment. Most approaches, however, specify properties in such a way that they are close to programming languages. The exception is [17], which provides a relatively high-level specification language. Our approach also includes a high-level property-specification language, but it serves multiple purposes. The same properties can be used for runtime verification, testing, or model-checking. In addition, as the language is pattern-based, it has a gentler learning curve, requiring less effort to specify properties.

At the other end of the spectrum, we have static analysis approaches, namely those based on formal verification. Trojanek et al. [13] translated three open-source robot navigation algorithms from C/C++ to SPARK, a language designed for the development of reliable software. With the toolkit available to SPARK, and after annotating the programs with verification conditions, they were able to unveil runtime errors in the original implementations and prove partial correctness of the translated algorithms. While our approach is unable to provide proof of correctness, it is able to operate on C++ code directly and requires considerably less manual work.

Another form of formal verification can be seen in [14], in which the authors model a high-level planning and scheduling system of Care-O-bot (https://www.care-o-bot.de/en/care-o-bot-3.html, accessed on 22 August 2021), for human–robot interaction scenarios. This is a rule-based system formally modeled and fed to a model checker to prove the correctness of four core behavioral properties (e.g., reminding a person to take medication at a given time). Compared to our approach, this approach is limited to models of the system's behavior. Our approach is able to handle the properties of the system's implementation and is able to capture other relevant aspects of software quality, such as architectural properties.

Mansoor et al. built a dependability case [15] to show that a family of surgical robots complies with a safety-critical property: that the surgeon is always aware of the position of the robotic arm within the patient. Their process involves constructing a formal model of the whole product line and its instances, capturing their collective and individual behavior. Then, resorting to a model finding tool, they are able to find counterexamples to use as the basis for tests. Our approach is unable to handle software product lines, but it benefits from automatic modeling and test generation steps.

Finally, on a different topic, we highlight the work Neto et al. [16], where the authors used the HAROS framework, as we do in this work, but applied to the FASTEN European Project—a complete stack of ROS-based software powering a mobile manipulator operating in an industrial environment. The analyses performed in this study aim to assess and improve the internal code quality of the project (e.g., compliance with metrics and coding standards). Successive iterations of analysis and improvement solved 25,440 reported issues, representing an overall reduction of 90%. Our work employs similar techniques in an initial phase and then moves on to the full range of features available to HAROS.

3. Case Study Subject

AgRobPP (https://gitlab.inesctec.pt/agrob/agrob_pp/, accessed on 22 August 2021) will be our case study subject. It is an open-source ROS-based framework for path planning in agricultural terrains, specifically for steep slope vineyards. AgRobPP was developed within the scope of the RoMoVi project (https://www.inesctec.pt/en/projects/romovi#about, accessed on 22 August 2021), and was recently published by the ROSIN European Project (https://www.rosin-project.eu/, accessed on 22 August 2021) under the AgRobIT project (https://www.rosin-project.eu/ftp/agrobit-mowing-robot, accessed on 22 August 2021). Typically, these vineyards are located along hills with irregular and sloped terrains representing a challenge for the path planning task; the ground inclination and robot's center of mass need to be considered during the path planning operation. Path planning operations may also be affected by the agricultural fields' dimensions, as a lot of computational memory could be required. For example, a steep slope vineyard from a small producer has an area of about 1 hectare, but larger farms span up to 70 hectares.

AgRobPP is composed of three main components:

1. AgRob Vineyard Detector—a satellite image segmentation tool [6,21];
2. AgRob Topologic Mapper—a tool to construct a topological map [6,22];
3. AgRob Path Planner—a path planning framework for uneven terrains [5];

AgRob Vineyard Detector is a machine learning tool based in Support Vector Machine (SVM), designed to detect vineyard lines and extract an occupation grid map without a previous robot visit to the terrain. AgRob Topologic Mapper takes this grid map and divides it into smaller zones, saving them in a graph structure. AgRob Path Planner is an A* search algorithm that navigates in the free cells of an occupation grid map, considering the robot's orientation and center of mass. Combined with AgRob Topologic Mapper, the search space gets reduced to the map's strictly necessary zones, making it more efficient in terms of memory.

4. Analysis Tools and Workflow

To improve the overall dependability of AgRobPP we rely on HAROS for our tooling. This framework incorporates multiple analyses in a single workflow, playing to the strengths of each, and is specially designed for roboticists:

- it takes source code as input, instead of models;
- it reverse engineers formal models as needed [23];
- it uses a high-level, pattern-based property specification language (https://github.com/git-afsantos/hpl-specs, accessed on 22 August 2021) that addresses ROS concepts directly;
- its reports use an interface that caters to ROS developers.

More concretely, our workflow consists of tackling three major (and mostly independent) fronts in the following order.

1. **Code quality analysis**, as seen in [16], aims to improve the overall *maintainability* of the project, i.e., make the code easier to read, share, change, and reuse.
2. **System architecture analysis**, rarely seen in the literature, aims to detect *orchestration* problems without executing the system, e.g., mismatching message types.
3. **System behavior analysis**, as seen in [13–15,17,20], aims to specify and verify diverse properties about the system's expected behavior, i.e., *correctness*.

We tackle code quality first, as it is mostly a ROS-agnostic and application-independent problem. While refactoring the code is laborious, the analysis is fast, and the results are immediate. This step unveils general programming mistakes and bad practices that, once fixed, will reduce the time spent on debugging the application-specific issues detected in the following steps as the code becomes more maintainable.

Next comes architectural analysis. This requires the model extraction capabilities of HAROS that, while mostly automatic, are unable to handle all cases in general. To remedy this, HAROS provides mechanisms for users to specify *extraction hints* [23] so that a sound and complete model is achieved. Naturally, there is room for iteration in this step. Besides manual inspection of the graphical models, a HAROS plug-in (https://github.com/git-afsantos/haros-plugin-pyflwor, accessed on 22 August 2021) enables user-defined queries that match and report problematic patterns in the model, such as two nodes exchanging messages but expecting different message types.

Lastly, there is behavioral analysis. Verifying the correctness, or functional safety, of a system is highly application-specific—it depends on what the system does, and on the concrete mission or application—and is one of the most sought-after analyses overall. HAROS encourages a dependability case approach, as seen in [15], in which users specify properties about the system's behavior, possibly break them down into smaller sub-properties, and then gather evidence of such properties using a variety of techniques. The properties are specified using the domain-specific HPL language. Then, coupling properties and extracted system models, different plug-ins handle the verification step. At the time of writing, plug-ins are available for runtime verification and property-based testing (https://github.com/git-afsantos/haros-plugin-pbt-gen, accessed on 22 August 2021), as well as model checking [24].

5. Analysis Process and Results

In this section, we describe the analysis process of this case study in terms of our approach, execution, and interpretation.

5.1. Code Quality Analysis

This type of static analysis is, as mentioned, independent of the concrete system. It is straightforward to execute, requiring only the installation of HAROS and its dependencies. No user input is required besides a set of target packages. The reported issues are mostly divided into *Coding Standards* and *Metrics*. As implied by the names, the former refers to non-compliance with a number of coding standards, while the latter warns against metrics that fall off expected thresholds.

We evaluated the three packages that make up the AgRobPP repository, a corpus consisting of 24,910 lines of C++ ROS-based code. An initial analysis resulted in a total of 7636 issues. Out of these, 7389 issues (96.77%) fall under Coding Standards, while 444 (5.81%) relate to Metrics—note the overlap of 197 issues. Fortunately, 4388 of these issues are related to code formatting, perhaps the easiest to fix.

Our approach to the formatting issues was simply to discard them. HAROS checks conformity with both the Google C++ Style Guide (https://google.github.io/styleguide/cppguide.html, accessed on 22 August 2021) and the ROS C++ Style Guide (http://wiki.ros.org/CppStyleGuide, accessed on 22 August 2021). In this case, we followed neither. Instead, we used the ClangFormat tool (https://clang.llvm.org/docs/ClangFormat.html, accessed on 22 August 2021) to automatically handle formatting, using a configuration provided by the ROS Industrial repositories (https://github.com/ros-industrial/industrial_calibration/blob/kinetic-devel/.clang-format, accessed on 22 August 2021). By ignoring this sub-category, we effectively had to handle 3248 issues.

The following rules give the most recurrent issues.

- *"Include all required headers for what you use."*
- *"Do not use integer types directly. Use size-specific `typedefs`, for instance from `<cstdint>`."*
- *"Maximum number of function lines of code of 40."*

The first rule is a simple yet solid practice that might be overlooked without static analysis tools. The code in question compiled successfully because the necessary libraries were transitively included in the headers we depended on. Should these headers drop the transitive dependencies in a future update, the AgRobPP code base would no longer build. The second rule advises against the use of platform-dependent integer types, e.g., `int`. It recommends the use of fixed-size types, such as `int32_t`, which bolster the portability of the project. All occurrences of `int` were manually replaced with an appropriately-sized integer type. This is a typical example of a tedious fix in the later stages of development. Had the analysis tools been part of the development process from the start, the `int` type would likely not have been used at all. Lastly, we have a metrics issue—functions that are over 40 lines of code are deemed too long. Fixing this issue after the fact requires care so that refactoring the code does not introduce new bugs. As such, it is still an ongoing process. However, this refactoring led us (indirectly) to uncover a critical fault in the path planner. Plans are published under two representations (that should always be equivalent): an array of poses (discrete), and an array of parametric curves (continuous). In some cases, the two outputs diverged.

Overall, after some back-and-forth iterations over the course of two weeks, we were able to go from 3248 non-formatting issues down to 1302—a reduction of 60%. We want to point out that it is easy to get carried away by the numbers with this kind of analysis. The goal, however, is not to get the number of issues down to zero (likely impossible, some tools are overzealous), but to get to a point where objectives are met (e.g., compliance with a standard) and the developers feel more confident about their code.

5.2. System Architecture Analysis

Our approach to architectural analysis was to start with a minimalistic configuration and progressively build up from there. To define a system configuration, HAROS requires, at least, the list of ROS *launch files* that orchestrate and deploy the executables. Choosing a minimalistic architecture to start enables us to add extraction hints as needed (and to understand the parsing limitations of HAROS), as well as iteratively fix any potential issues. Our starting architecture was the AgRob Path Planner node connected to a map server and a node that generates starting and goal poses in the map—i.e., a complete working example (Figure 2).

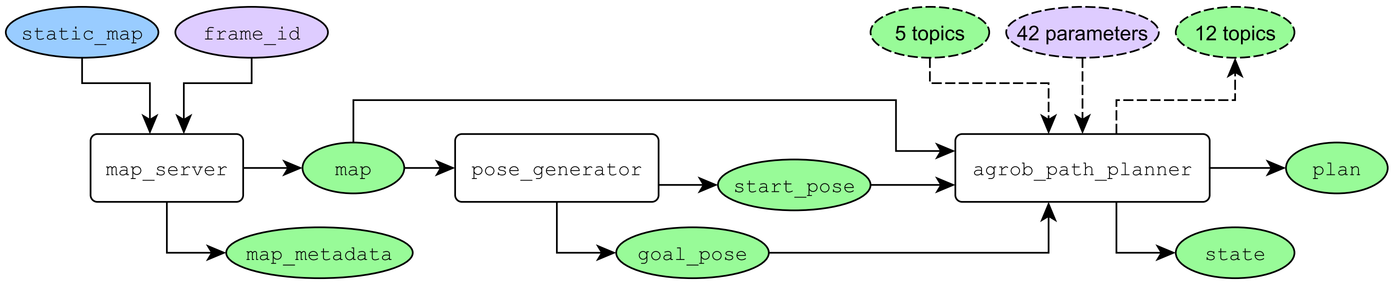

Figure 2. Minimal AgRobPP architecture, showing ROS nodes (rectangles), ROS topics (green) ROS services (blue), and ROS parameters (purple).

HAROS was mostly able to extract the architecture on its own, but it was unable to fully resolve the names of 25 ROS parameters. Fortunately, HAROS shows completion suggestions that help write down the necessary hints. In total, we wrote 29 lines of YAML data to fix 25 parameter names. In comparison, our full robot system required 320 lines of YAML data to fix 14 entities and create 61 missing entities.

The analysis itself did not find any relevant issues. Manual inspection of the models, coupled with several months of prior testing of the system had already ruled out orchestration mistakes. Regardless, we bolstered our confidence in the system with a small catalog of 14 queries, applied across all architectures, matching problematic patterns such as:

- topics with unbounded message queues, i.e.,
  `topics/publishers[self.queue_size == 0]|topics/subscribers [self.queue_size == 0]`;
- topics with multiple publishers, i.e.,
  `topics[len(self.publishers) > 1]`;
- missing publishers for starting or goal poses, i.e.,
  `topics[self.topic_name in ('goal_pose', 'start_pose') and not self.publishers]`.

Should any future changes or additions to the system violate any of these rules, there will be an automatic check in place.

5.3. System Behavior Analysis

Verifying a robotic system's behavior is arguably the most challenging and interesting task in the whole analysis workflow from a software engineering perspective. It is definitely the task that requires the most amount of work: (i) requirements must be analyzed for each system configuration; (ii) specifying properties requires careful thought, to avoid logic mistakes; (iii) verification techniques are harder to implement, and (iv) the verification process takes longer than previous analyses. In this case, we have applied behavioral analysis only to the minimal architecture shown in Figure 2.

As previously mentioned, HAROS encourages the construction of dependability cases. Our high-level property for this particular path planner is that *after receiving and loading a map, given valid starting and goal poses in this map, the AgRob Path Planner shall produce a valid plan from the starting pose to the goal, if a path between the two exists.* This is a complex property, spanning the whole system, that we have to break down into smaller, more manageable pieces. After gathering evidence that each sub-property holds, we can argue that the initial, high-level property does as well.

The property breakdown lends itself naturally to an iterative process. For instance, we can start by dividing the initial property into the following three pieces.

1. Receiving and *loading a map*.
2. Receiving *valid starting and goal poses* in the map.
3. Producing a *valid plan* if one exists.

Then, we must define what does it mean to *load a map*, etc., in terms of observable behavior, i.e., in terms of exchanged ROS messages. The specified properties are based on ROS messages because the HPL specification language operates on this concept, treating systems as black-boxes; we cannot depend on internal behavior, variables, or data structures.

The AgRob Path Planner is designed as a state machine that publishes its current state on the `state` topic upon entering a new state. Right away, we can define a number of safety and liveness properties that encode this state machine, e.g.,

```
1 globally: no state { not data in { "PLANNING",
2 "WAITING_FOR_MAP", "LOADING_MAP", "READY",
3 "PLANNING_OK", "PLANNING_FAILED" }}
4 after map: some state { data = "LOADING_MAP" }
5 within 2 s
6 after state { data = "READY" }
7 until state { data = "PLANNING" }: no state
```

The first property (lines 1–3) enumerates all valid states; there should be no message on `state` such that its `data` value is not contained in the specified enumeration. The second property (lines 4–5) states that after a `map` is published (there is only one published map in this configuration), a transition to `LOADING_MAP` should be observed in the next two seconds. The third property (lines 6–7) invalidates any state between `READY` and `PLANNING`. In other words, the only available transition from `READY` is to `PLANNING`. In total, there are 16 properties related to the path planning state machine.

Now, we can see that *loading a map* is simply observing a `map`, and a `state` sequence starting with `WAITING_FOR_MAP`, going through `LOADING_MAP` and terminating with `READY`.

Defining *valid starting and goal poses* is slightly more complex. Poses are valid if (i) they are located within the map's boundaries, and (ii) they fall on free space (i.e., not on top of a wall or obstacle). Maps are given by occupancy grids, such as the one in Figure 3. A pose is said to be on free space if, after converting from real-world coordinates to map coordinates, the corresponding grid cell is marked as empty.

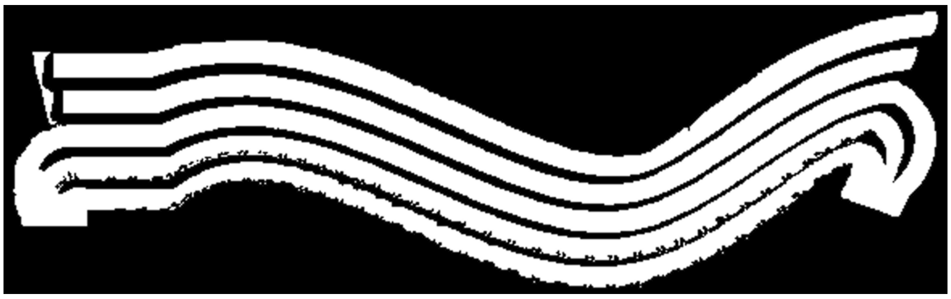

Figure 3. Occupancy grid map of a steep slope vineyard.

A *valid plan* is defined similarly. Each pose along the plan must be a valid pose. Figure 4 shows an example of two generated paths using the map represented in Figure 3. The blue path considers the robot's center of mass, while the red path is ignoring this feature.

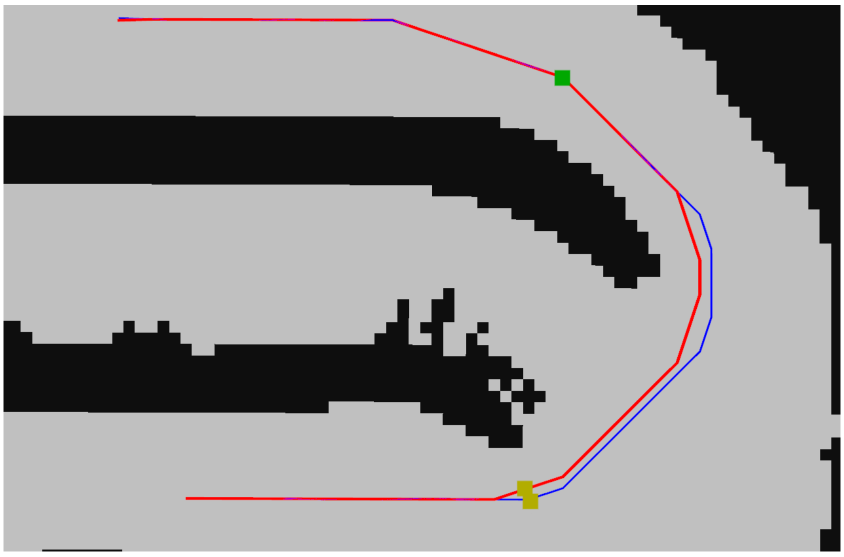

Figure 4. AgRobPP path generation example. Blue—considering centre of mass; Red—without center of mass awareness. Reprinted with permission from ref. [5]. Copyright 2019 Cambridge University Press.

Besides, for each consecutive pair of poses in the plan, we define safety thresholds for the distance between their positions and the difference between their orientations. Thus, a `plan` is validated with 6 properties.

Overall, the breakdown of a single high-level property led us to specify 30 HPL properties. However, note that this is not a full specification of the AgRob Path Planner. The node also publishes the generated path in a ROS format suitable for visualization, publishes the same path with a parametric curve representation for trajectory controllers, and may also publish local plans to adapt to new obstacles along the way. Specification for these features is still ongoing work. We estimate that, for the 30 properties we have so far, it took about two weeks of fine-tuning and on-and-off discussion between roboticists and software engineers until both parties were satisfied.

After specification comes verification, and, for this task, HAROS provides plug-ins based on model checking, runtime verification, and property-based testing. The model checking plug-in is of limited use for this AgRob configuration. It specializes in system-wide properties (properties that span multiple components at once) and takes node-specific properties as axioms. Given this architecture's small dimension, most of our properties are axioms rather than assertions to be checked. Once we delve into more complex configurations, the plug-in will become increasingly more helpful, even though it is unable to handle some language features, such as real-time constraints. Thus, for this step, we focus mostly on property-based testing, which, in turn, uses runtime verification to produce observers (and oracles).

Simply put, this plug-in generates tests that attempt to falsify properties by repeating test cases over and over, hundreds (or thousands) of times, and by generating semi-random inputs for each attempt. Once a counterexample is found, it uses heuristics to try to minimize it, so that only a relatively small (ideally, a minimal) example input is displayed to the user. In this case, the input is a trace of messages to be published on topics that the system subscribes, but for which there are no publishers (e.g., the five omitted topics subscribed by the AgRob Path Planner in Figure 2). Tests do not tamper with topics for which the system provides publishers.

We could test the path planning node in isolation (a *unit test*), but, in this case, it is advantageous to test the full architecture at once. Unit tests would delegate the generation of maps and poses to the test infrastructure. Generating random maps is of little value (too much noise), and specifying rules for realistic maps takes too much effort. It is simpler to include the `map_server`, and repeat the process for different maps, such as the one shown in Figure 3.

After spending a full working day (about 8 h) on testing, discussion, and interpretation of test results, we have found 2 properties that needed further fine-tuning (false assumptions), and 4 bugs without immediate critical effects.

Bug #1: In this simple configuration, some of the published topics of the AgRob Path Planner should not be used. By stimulating the right inputs, the testing infrastructure found instances in which unexpected messages could be observed. This is not a critical bug, because the output of these topics feeds debugging components and unexpected messages will not imply a danger action from the robot.

Bug #2: The planner should only transition to `PLANNING` once it receives a starting pose and a goal pose in the `READY` state. The lack of proper pre-conditions in the message handler meant that poses were always being processed, even, e.g., when loading the map. Thus, poses sent prior to entering the `READY` state would still trigger a transition to `PLANNING`. The consequences of this bug may be severe. The planner may generate a random path, generating an unexpected behavior with unpredictable outcomes.

Bug #3: Under some circumstances, a `plan` would be published, but the planner would not enter `PLANNING_OK`. This bug will not bring a robotic safety issue but reduces the reliability of the robot. The entire software may block and freeze the robot.

Bug #4: We expected no `plan` to be published when on `PLANNING_FAILED`, but the planner did publish empty plans. This may cause unexpected severe behavior in the trajectory controller software (3rd party) that is usually ready to send velocity commands when a plan is received in a ROS topic.

In summary, with some specification investment and a proper analysis workflow, we were quickly able to find subtle bugs that had eluded developers for months at this point.

6. Conclusions

This paper reports on the joint effort between roboticists and software engineers to improve the dependability of robotic software. We defined a software analysis workflow based on the HAROS framework and applied it to the AgRobPP path planner for steep slope vineyards. Even though our process relied on HAROS, for its overall convenience and suitability for the task, the methodology matters more than the tools. The same workflow can, in theory, be replicated with other tools and, likely, some additional effort.

Transitioning from an ad hoc quality assurance process to a well-defined workflow had immediate effects on the product's overall quality. The source code became more modular and easier to maintain. In addition, 4 bugs in the system's behavior were detected and fixed. The benefit–cost ratio is definitely positive, considering the late adoption of the quality assurance process, and considering that, overall, it took only about 180 man-hours to achieve these results. We argue, thus, that similar workflows should be more widespread in the open-source robotics ecosystem, and the relationship between roboticists and software engineers should be further exploited.

In terms of future work, our first goal is to achieve a full specification of the AgRob Path Planner, or as close to it as possible. Not only will this improve our confidence in a core component of the robotic system as a whole, it will also serve as a form of documentation. Our second goal is to scale the presented process to more complex architectures so that more components of the system benefit from it. We will look into integrating further analyses into the workflow. For instance, we can use software model checking to verify intra-node functional safety, something we currently lack. We also intend to perform a statistical evaluation of the code's performance before and after the bug's correction. Lastly, we will start the transition of the robotic software to ROS2, which uses DDS (Data Distribution Service), supporting real-time aspects and QoS (e.g., Deadline and Reliability). These features will open several possibilities for the development of the HAROS framework in the future.

Author Contributions: Conceptualization, L.C.S., A.S. and F.N.S.; methodology, A.S. and L.C.S.; software, A.S. and L.C.S.; validation, L.C.S. and A.S.; formal analysis, A.S.; investigation, L.C.S. and A.S.; resources, L.C.S. and A.S.; data curation, L.C.S. and A.S.; writing—original draft preparation, L.C.S. and A.S.; writing—review and editing, A.S., L.C.S., F.N.S. and A.V.; visualization, F.N.S. and A.V.; supervision, F.N.S. and A.V.; project administration, F.N.S.; funding acquisition, F.N.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work is financed by the ERDF—European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation—COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT—Fundação para a Ciência e a Tecnologia within project PTDC/CCI-INF/29583/2017 (POCI-01-0145-FEDER-029583). Luís Carlos Feliz Santos thanks the FCT for the Ph.D. Grant DFA/BD/6152/2020.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Leeson, G.W. The growth, ageing and urbanisation of our world. *J. Popul. Ageing* **2018**, *11*, 107–115. [CrossRef]
2. Leshcheva, M.; Ivolga, A. Human resources for agricultural organizations of agro-industrial region, areas for improvement. In Proceedings of the Sustainable Agriculture and Rural Development in Terms of the Republic of Serbia Strategic Goals Realization within the Danube Region: Support Programs for the Improvement of Agricultural and Rural Development, Belgrade, Serbia, 14–15 December 2017; Thematic Proceedings 2018; pp. 386–400.
3. Robotics, E. Strategic Research Agenda for Robotics in Europe 2014–2020. 2014. Available online: Eu-robotics.net/cms/upload/topicgroups/SRA2020SPARC.pdf (accessed on 21 April 2018).
4. Santos, L.; dos Santos, F.N.; Pires, E.J.S.; Valente, A.; Costa, P.L.; Magalhães, S. Path Planning for ground robots in agriculture: A short review. In Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Ponta Delgada, Portugal, 15–17 April 2020; pp. 61–66. [CrossRef]
5. Santos, L.; dos Santos, F.N.; Mendes, J.; Costa, P.; Lima, J.; Reis, R.; Shinde, P. Path Planning Aware of Robot's Center of Mass for Steep Slope Vineyards. *Robotica* **2020**, *38*, 684–698. [CrossRef]
6. Santos, L.C.; Aguiar, A.S.; Santos, F.N.; Valente, A.; Petry, M. Occupancy Grid and Topological Maps Extraction from Satellite Images for Path Planning in Agricultural Robots. *Robotics* **2020**, *9*, 77. [CrossRef]
7. Santos, L.; de Aguiar, A.S.P.; dos Santos, F.N.; Valente, A.; Ventura, J.B.; Sousa, A.J. Navigation Stack for Robots Working in Steep Slope Vineyard. In *Intelligent Systems and Applications (IntelliSys)*; Advances in Intelligent Systems and Computing; Springer: Cham, Switzerland, 2020; Volume 1250, pp. 264–285._21. [CrossRef]
8. Aguiar, A.S.; dos Santos, F.N.; Sobreira, H.; Cunha, J.B.; Sousa, A.J. Particle filter refinement based on clustering procedures for high-dimensional localization and mapping systems. *Robot. Auton. Syst.* **2021**, *137*, 103725. [CrossRef]
9. Quigley, M.; Conley, K.; Gerkey, B.P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. 2009. Available online: http://www.cim.mcgill.ca/~dudek/417/Papers/quigley-icra2009-ros.pdf (accessed on 22 August 2021).
10. Ingibergsson, J.; Schultz, U.; Kuhrmann, M. On the Use of Safety Certification Practices in Autonomous Field Robot Software Development: A Systematic Mapping Study. In *PROFES 2015, Proceedings of the 16th International Conference, Bolzano, Italy, 2–4 December 2015*; LNCS; Springer: Cham, Switzerland, 2015; Volume 9459, pp. 335–352.

11. Santos, A.; Cunha, A.; Macedo, N.; Lourenço, C. A Framework for Quality Assessment of ROS Repositories. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; pp. 4491–4496. [CrossRef]
12. Santos, A.; Cunha, A.; Macedo, N. Property-based Testing for the Robot Operating System. In Proceedings of the ACM SIGSOFT International Workshop on Automating Test Case Design, Selection, and Evaluation (A-TEST@ESEC/SIGSOFT FSE), Lake Buena Vista, FL, USA, 5 November 2018; pp. 56–62. [CrossRef]
13. Trojanek, P.; Eder, K. Verification and testing of mobile robot navigation algorithms: A case study in SPARK. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, USA, 14–18 September 2014; pp. 1489–1494.
14. Webster, M.; Dixon, C.; Fisher, M.; Salem, M.; Saunders, J.; Koay, K.L.; Dautenhahn, K.; Saez-Pons, J. Toward Reliable Autonomous Robotic Assistants Through Formal Verification: A Case Study. *IEEE Trans. Hum.-Mach. Syst.* **2016**, *46*, 186–196. [CrossRef]
15. Mansoor, N.; Saddler, J.A.; Silva, B.; Bagheri, H.; Cohen, M.B.; Farritor, S. Modeling and Testing a Family of Surgical Robots: An Experience Report. In Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE), Lake Buena Vista, FL, USA, 4 –9 November 2018; pp. 785–790. [CrossRef]
16. Neto, T.; Arrais, R.; Sousa, A.; Santos, A.; Veiga, G. Applying Software Static Analysis to ROS: The Case Study of the FASTEN European Project. In Proceedings of the Iberian Robotics Conference—Advances in Robotics (ROBOT), Porto, Portugal, 20–22 November 2019; Advances in Intelligent Systems and Computing; Springer: Cham, Switzerland, 2019; Volume 1092, pp. 632–644._51. [CrossRef]
17. Adam, M.S.; Larsen, M.; Jensen, K.; Schultz, U.P. Rule-based Dynamic Safety Monitoring for Mobile Robots. *J. Softw. Eng. Robot.* **2016**, *7*, 120–141.
18. Luo, C.; Wang, R.; Jiang, Y.; Yang, K.; Guan, Y.; Li, X.; Shi, Z. Runtime Verification of Robots Collision Avoidance Case Study. In Proceedings of the IEEE Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; IEEE Computer Society: Washington, DC, USA, 2018; pp. 204–212. [CrossRef]
19. Ulus, D.; Belta, C. Reactive Control Meets Runtime Verification: A Case Study of Navigation. In Proceedings of the International Conference on Runtime Verification (RV), Porto, Portugal, 8–11 October 2019; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2019; Volume 11757, pp. 368–374._21. [CrossRef]
20. Lesire, C.; Roussel, S.; Doose, D.; Grand, C. Synthesis of Real-Time Observers from Past-Time Linear Temporal Logic and Timed Specification. In Proceedings of the International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 597–603. [CrossRef]
21. Santos, L.; Santos, F.N.; Filipe, V.; Shinde, P. Vineyard segmentation from satellite imagery using machine learning. In Proceedings of the EPIA Conference on Artificial Intelligence, Vila Real, Portugal, 3–6 September 2019; Springer: Cham, Switzerland, 2019; pp. 109–120.
22. Santos, L.; dos Santos, F.N.; Magalhães, S.; Costa, P.; Reis, R. Path Planning approach with the extraction of Topological Maps from Occupancy Grid Maps in steep slope vineyards. In Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Porto, Portugal, 24–26 April 2019; pp. 1–7. [CrossRef]
23. Santos, A.; Cunha, A.; Macedo, N. Static-Time Extraction and Analysis of the ROS Computation Graph. In Proceedings of the IEEE International Conference on Robotic Computing (IRC), Naples, Italy, 25–27 February 2019; pp. 62–69. [CrossRef]
24. Carvalho, R.; Cunha, A.; Macedo, N.; Santos, A. Verification of System-Wide Safety Properties of ROS Applications. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 25–29 October 2020.