24-0006

Future Directions in Software Engineering for Autonomous Robots

Davide Brugali[1], Ana Cavalcanti[2], Nico Hochgeschwender[3],
Patrizio Pelliccione[4], Luciana Rebelo[4]

[1] Università degli studi di Bergamo - Bergamo, Italy - davide.brugali@unibg.it
[2] University of York - York, United Kingdom
[3] Universität Bremen - Bremen, Germany
[4] Gran Sasso Science Institute - L'Aquila, Italy

I. INTRODUCTION

Software Engineering for Autonomous Robots investigates the synergetic interaction of computing and robotic technologies; this synergy is both realistic and strategic. On one hand, software engineering techniques and methods have the potential to enhance the quality of the software that controls autonomous robots and the effectiveness of software development processes in the robotic industry. In addition, they can provide evidence of that improved quality to foster trustworthiness of the robot via advanced verification techniques based on testing, simulation, or even mathematical proof. On the other hand, autonomous robots represent a significant benchmark for software engineering techniques and approaches, and many technologies that have been successfully applied in robotics (e.g. from control theory to automated planning) are now a source of inspiration for software engineers.

Starting from the pioneer works on robotic software architectures at the end of the last century up to the wide and pervasive adoption of ROS (Robot Operating System) in today's development of robot control systems, there has been a steady increase in the robotic practitioners' awareness of the positive impact of software engineering methods in robotics. This is witnessed by some milestones. In 2005 the IEEE RAS Technical Committee on Software Engineering for Robotics and Automation[1] was established. In 2019 the euRobotics initiatives established a topic group focused on Software Engineering, Systems Integration and Systems Engineering[2]. Software engineering keywords, such as verification, validation, frameworks, and architectures, are now listed as topics of several robotics journals.

However, robotics still faces many software engineering challenges related to the development, integration, validation, verification, installation, and operation of robot control systems due to their cyber-physical nature. Verification and validation of robot behaviors are also concerned with human-robot interaction, besides cyber-physical features. The challenges are significant as trust is a concept that cannot be easily measured, especially when the robot is equipped with AI techniques that generate intelligent behavior.

We aim at raising awareness in software engineering for robotics with the objective of building bridges between the communities of software engineering and robotics. Specifically, we suggest a range of possible directions with new challenges for robot software engineering to be explored. We are based on recent studies on the state of the practice in software development for robotics, and on the discussion between the participants of the IEEE ICRA-23 Workshop on Robot Software Architectures (RSA23)[3] organized by the RAS Technical Committee on Software Engineering for Robotics and Automation and attended by over 100 participants from academia and industry. The following sections present four research directions that we believe will advance the state of the practice in software engineering for robotics and will mitigate those challenges.

II. Capturing Domain Expertise with Reference architectures

During the last twenty years, several software frameworks have been proposed by the research community to reduce the complexity of developing software control systems for autonomous robots, including SmartSoft, OROCOS, YARP, and ROS. They promote a component-based software development approach, in which two separate development cycles interact: development of fine-grained components (e.g. a motion planning package, a 3D perception package, etc.) and development of a variety of composites (e.g. a control

[1] TC on software engineering for robotics and automation, https://www.ieee-ras.org/software-engineering-for-robotics-and-automation, (last accessed 01 June 2024).

[2] Software Engineering, Systems Integration and Systems Engineering, https://sparc-robotics-portal.eu/web/software-engineering/home, 2023 (last accessed 01 June 2024).

[3] ICRA 2023 RSA Workshop. https://roboticsa.github.io/RoboticSA2023/ (last accessed 01 June 2024).

application for hospital logistics).

When a component is used in a large number of systems by different developers, the knowledge about the component usage, robustness and efficiency is available in the user community. So, component reuse promotes the development of maintainable, trustworthy, and affordable software systems. The weakness of component-oriented development is the lack of support for system architecture reuse.

Recent study reveals that the main reason (78.2%) among the survey respondents to not reuse existing software packages related to architectural problems (e.g., component's granularity does not fit, missing functionality, incompatible interfaces). As a consequence, to ensure the interoperability among heterogeneous or incompatible system components, practitioners tend to rely on ad hoc practices instead of principled engineering approaches.

Some research projects, such as EU FP7 BRICS and EU2020 RobMosys have addressed these issues by developing guidelines and recommendations emerging from successful reuse and integration of robotic software in a variety of research and industrial scenarios. Similarly, the ROS Wiki now hosts two new sections on "ROS Best Practices" and "ROS Use Patterns In robotics"[4].

A possible further step forward could be the definition, documentation, and systematic reuse of architectural solutions to recurrent design problems emerging from the robotic domain. While best practice recommendations and use patterns may be considered just cookbook recipes, a software architecture for a family of products (e.g., service robots) has the ability to capture the structural and behavioral commonalities in the overall design of similar systems (e.g., the navigation framework). This is commonly called a reference architecture and defines not only how components interact by means of communication and synchronization mechanisms (e.g., topic-based or service-based), but also the principles and policies that must be enforced by the set of interacting components.

Defining reference architectures for autonomous robots is challenging, because software development for robotics is characterized by the high variability in application requirements. The different hardware configurations, the dynamic operational environment, the variety of tasks are drivers of variability for software that implements robot functionalities. A promising approach, exemplified by the ROS 2 Navigation stack, consists in defining reference architectures for specific system abilities as defined in the EU2020 Multi Annual Roadmap, e.g., Manipulation, Perception, Navigation, Decisional Autonomy, etc.

III. Validation and verification: need for heterogeneous and compositional reasoning

The properties of a cyber-physical system, such as a robot, often depend on the behavior of its specialized hardware and environment. For autonomous service robotics we also need to deal with socio-technical concerns related to human stakeholders to validate and verify properties. In this context, traditional reasoning techniques for testing, simulation, and proof for software systems are relevant, but not enough.

We have two significant challenges when it comes to reasoning. The first is the presence of heterogeneous components, whose modeling and design are very different. Even development of simulations imposes a challenge when there are proprietary components involved. Testing becomes more difficult as an oracle needs to determine whether measurements of continuous data are good enough. Proof requires richer theories dealing with continuous time, data, and probabilities.

The second significant problem is scalability. Here, compositional reasoning is key. We need to take advantage of the many advances on architectures and variability to understand how component-level results are useful for system-level reasoning.

We ought to consider several forms of reasoning for validation and verification. Simulation is popular, but hampered by low-level, tool-dependent coding. So, behaviors observed in simulation may be a consequence of coding errors, rather than inadequate designs. Bringing heterogeneous simulation models together is also a problem. Possible solutions are automatic code generation and co-simulation.

Testing is a widely used form of reasoning. Tests generated and executed manually, however, have limited reasoning power. Model-based testing can decrease costs by automating the test-generation process, but, even more importantly, it is accompanied by a fault-detection guarantee and support for reproducing experiments.

Finally, the strongest form of reasoning is based on proof. It can use finite abstractions and automatic approaches, but scalability issues are exacerbated by continuous or hybrid models. More promising is the combination of model checking and theorem proving, which can deal with large or infinite models. Semantic integration is key, and automation is possible, but a challenge.

IV. From prototypes to products

Robots are not yet ubiquitous in our daily life, and impressive and capable robot prototypes often remain in laboratory environments. One reason for this is that little is known and publicly reported about successful robot deployment. How should the robot software be designed? How should the robot software be tested? There is not such a thing as a publicly available repository which helps to answer these questions and which harmonizes robot software engineering related best practices and insights. Although few robotic standards exist that consolidate software engineering practices by recommending design and implementation patterns to avoid common failure points, exploiting this

[4] [http://wiki.ros.org/BestPractices] (last accessed 01 June 2024).

knowledge systematically in the robot development process remains challenging, as it is not integrated into practice and robot software development tools. Turning capable robot prototypes to valuable products remains a challenge for another reason. Although robotic application developers can select from a wide range of robot components to perform advanced sensing, planning, and control tasks, little is known about their expected performance once they are integrated and deployed in robotic applications. This is a reality check that decides whether to make or break, and, too often, this is done in an ad hoc manner by manually checking within controlled scenarios whether the robot will operate at scale, without human intervention, and with the desired repeatability and precision.

Thus, establishing a methodology to support engineers not only to boost their robot components to a higher Technology Readiness Level (TRL) but also to systematically deploy prototype systems into reliable products could be a promising future research direction for robot software engineering. This methodology would support engineers in collecting the required evidence that their robots operate correctly and safely in their targeted environments, and at the same time, increase the trustworthiness of deployed robot applications. Examples of evidence include, to name a few, field test reports, standard conformance assessments, and other quality assurance artifacts such as code and design reviews. However, such boosting methodology is not currently available for robotics. Preferably, the methodology also considers the complete lifecycle of robotic systems and applications, including continuous maintenance and improvement, which is a topic of high relevance for learning-enabled robots operating over a long period.

A potential contribution to such a boosting methodology can be found in the field of performance evaluation and benchmarking. Robotics has a long tradition of performance evaluation and benchmarking through scientific competitions. Over the past decades, teams from across the world have competed with their robots during notable events such as the DARPA Grand Challenge and RoboCup. Competition tasks range from application-oriented scenarios in domestic, manufacturing, logistics, and disaster environments to game-oriented scenarios such as soccer. Because teams compete regularly under replicable conditions, it is possible to monitor and assess scientific progress. Although the focus of these competitions is different from, for example, a company aiming to turn a prototype into a product, there are opportunities to generalize insights, practices, and evaluation protocols gained from competitions to boost robot prototypes to products. We argue that competition already encodes a huge body of knowledge on how and at which level of detail acceptance criteria of robotic systems are specified and how to measure and assess these acceptance criteria. Detailing these acceptance criteria into robot software tools and possibly automating their evaluation in the context of real product scenarios could be a component in incrementally boosting the TRL of robotic systems and, at the same time, a promising future direction for robot software engineering.

V. Engineering trustable robots

With robots becoming an integral part of our daily lives, answering the following question becomes more and more urgent: How to engineer robots that humans will trust?

We will not accept anymore robots that expose (i) behaviors that are not aligned with our social and human values, like safety, fairness, ethics, and privacy, and (ii) opaque behaviors, i.e., without explaining the rationale for choices and actions. The use of AI in some robotic modules exacerbates the problem of trust. This is testified by the recent AI Act released by the European Union, which is the first law to regulate the use of AI in Europe. It provides a framework that delineates different requirements and obligations for both the use and provision of AI systems within the European Union (EU). It classifies AI systems based on risk for humans and society in four categories, and regulates if and how AI-based systems can be produced, used or deployed.

While accomplishing a mission, a robot will be required to obey laws and follow specific regulations, not only concerning safety or security, but also social and human values, as, e.g., regulated by the AI act. For instance, let us consider a robot in the assistive healthcare domain. The patient or operators supporting the patient in this activity, should be able to specify preferences like consensus about the treatment, risk management assessment, balance for benefit or moral attitude. Then, there is the need of instruments to guarantee that the behavior of the robot(s) will be compliant with the specified ethical preferences. This can be obtained by automatically transforming the ethical specification into an unambiguous and precise formulation, for instance, in some form of modal or temporal logic, which can be programmatically exploited, e.g., to automatically generate the correct-by-construction logic needed for coordinating the robots and their interactions with humans, as well as the environment.

Explainability and monitorability are important methods to contribute to the perceived trustworthiness by users, industry, and society. As described in[5], 'explainable, causal and ethical AI' is a potential key driver of adoption. Despite the ideal of a human-centric AI and the recommendations to empower the users via transparency and accountability of decisions, the power and the burden to preserve the users' rights still remain in the hands of the (autonomous-) systems producers. There is the need for theory and software technology to equip persons with a shield that empowers them during their interaction with the digital world in accordance with their views about privacy and ethics.

[5] https://www.mckinsey.com/industries/healthcare/our-insights/transforming-healthcare-with-ai (last accessed 01 June 2024).

24-0006

VI. Conclusions

We sketched four research directions in software engineering for autonomous robots that end-users and other humans will trust. These directions cover the complete life-cycle from design, to validation, and deployment of robot software and are not complete, yet a starting point for making robots more trustworthy.

Acknowledgments

This work was supported in part by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU. We would like to thank all the participants of the ICRA 2023 workshop on Robot Software Architectures.