Lecture Notes in Computer Science **16504**

Advanced Research in Computing and Software Science

Subline of Lecture Notes in Computer Science

More information about this series at https://link.springer.com/bookseries/558

Elvira Albert · Corina Pasareanu
Editors

Fundamental Approaches to Software Engineering

29th International Conference, FASE 2026
Held as Part of the International Joint Conferences
on Theory and Practice of Software, ETAPS 2026
Turin, Italy, April 11–16, 2026
Proceedings

Springer

Editors
Elvira Albert
Complutense University of Madrid
Madrid, Spain

Corina Pasareanu
Carnegie Mellon University
Pittsburgh, PA, USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-032-22773-7 ISBN 978-3-032-22774-4 (eBook)
https://doi.org/10.1007/978-3-032-22774-4

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

ETAPS Foreword

Welcome to the 29th edition of ETAPS, which took place as an on-site event in Turin, Italy during April 11–16, 2026!

ETAPS 2026 was the 29th instance of the International Joint Conferences on Theory and Practice of Software (ETAPS). ETAPS is an annual federated conference established in 1998, and consists of four main conferences: ESOP, FASE, FoSSaCS, and TACAS. Each conference has its own Program Committee (PC) and its own Steering Committee (SC). The ETAPS main conferences cover various aspects of software systems, ranging from theoretical computer science to foundations of programming languages, tools and algorithms for system analysis, and formal approaches to software engineering. Organizing these conferences in a coherent, highly synchronized conference programme enables researchers to participate in an exciting event, having the possibility to meet many colleagues working in different directions in the field, and to easily attend talks of different conferences. In addition to its four main conferences, ETAPS 2026 also hosted fifteen satellite workshops and two colocated events, which together further attracted many researchers from all over the globe.

ETAPS 2026 received 456 submissions in total, 138 of which were accepted, yielding an overall acceptance rate of 30%. Out of the 138 accepted papers, 16 papers were selected as ETAPS distinguished papers. I thank all the authors of submitted papers for their interest in ETAPS, all the reviewers for their reviewing efforts, the PC members for their contributions, and in particular the PC (co-)chairs for their hard work in running this entire intensive process in a constructive, objective and timely manner. I congratulate all authors of the ETAPS 2026 accepted papers!

ETAPS 2026 featured the unifying invited keynotes by

- Monika Henzinger (Institute of Science and Technology Austria, Austria), delivering a talk about "Guarding Privacy Over Time: Challenges and Solutions in Continuous Data Observation",
- Einar Broch Johnsen (University of Oslo, Norway), discussing "Formal Methods Meet Digital Twins: Challenges and Opportunities".

ETAPS 2026 hosted the invited keynote speakers

- Christel Baier (Technische Universität Dresden, Germany) for FoSSaCS, presenting "Verification of Infinite-horizon Properties of Dynamic Bayesian Networks",
- Guy Van den Broeck (University of California, Los Angeles, USA) for TACAS, introducing "Symbolic Reasoning in the Age of Large Language Models".

The ETAPS 2026 invited tutorials were provided by

- Mieke Massink (CNR-ISTI Pisa, Italy) on "Model Checking in Space with Applications to Medical Image Analysis",
- Leonardo de Moura (Amazon Web Services, USA) surveying "The Lean Programming Language and Theorem Prover".

The ETAPS 2026 programme also featured a lively Ask-Me-Anything session, interactive tool demos, a Diversity, Equity, and Inclusion session, SV-Comp and Test-Comp community building events, and the ETAPS industry day. The goal of the ETAPS industry day is to bring industrial practitioners into the heart of the research community and to catalyze the interaction between industry and academia. The ETAPS 2026 industry day was organized by Giorgio Audrito (University of Turin, Italy), Sean Kauffman (Queen's University, Kingston, Canada), and Nikolai Kosmatov (Thales Research and Technology, Palaiseau, France).

ETAPS 2026 was organized by the Department of Computer Science of the University of Turin, which is the center for coordinating research, teaching, dissemination and technological transfer in computer science in Turin, Italy. The department covers both methodological and application oriented aspects of computer science, and performs research in several interdisciplinary areas. This is reflected in the collaborations with other research centers and companies in many scientific areas and in its participation in national, European and international projects.

ETAPS 2026 was further supported by the following associations and societies: ETAPS e. V. (the ETAPS Association), EATCS (European Association for Theoretical Computer Science), EAPLS (European Association for Programming Languages and Systems), and EASST (European Association of Software Science and Technology).

The ETAPS Steering Committee consists of an Executive Board, and representatives of the individual ETAPS conferences, as well as representatives of EATCS, EAPLS, and EASST. The Executive Board consists of Laura Kovács (TU Wien, chair), Andrzej Wąsowski (IT University of Copenhagen, vice-chair), Thomas Noll (RWTH Aachen, treasurer), Arnd Hartmanns (University of Twente, artifact evaluation coordinator), Barbara König (University of Duisburg-Essen, proceedings coordination), Caterina Urban (Inria, PhD activities), Elizabeth Polgreen (University of Edinburgh, social media), Jan Kofroň (Charles University Prague, organisational support, website), Jan Křetínský (Masaryk University Brno and TU Munich, diversity & inclusion), and Marieke Huisman (University of Twente, blog, awards). Further members of the ETAPS Steering Committee committee are: Robbert Krebbers (Radboud University Nijmegen), Azalea Raad (Imperial College London), Luís Caires (Tecnico ULisboa), Elvira Albert (Universidad Complutense de Madrid), Corina Păsăreanu (Carnegie Mellon University), Erika Ábrahám (RWTH Aachen), Marsha Chechik (University of Toronto), Marie-Christine Jakobs (LMU Munich), Nathalie Bertrand (Inria Rennes), Stefan Milius (Friedrich-Alexander Universität Erlangen-Nürnberg), Alexandra Silva (Cornell University), Joël Ouaknine (MPI-SWS Saarbrücken), Andrzej Murawski (University of Oxford), Sebastian Junges (Radboud University Nijmegen), Guy Katz (The Hebrew University of Jerusalem), Christian Schilling (Aalborg University), Naijun Zhan (Peking University), Joost-Pieter Katoen (RWTH Aachen and University of Twente), Dirk Beyer (LMU Munich), Fabrice Kordon (Sorbonne University Paris), Laure Petrucci (Université Paris 13), Peter Y.A. Ryan (University of Luxembourg), Claudio Menghi (University of Bergamo and McMaster University Hamilton), Mark Lawford (McMaster University Hamilton), Maurice ter Beek (CNR-ISTI Pisa), Ferruccio Damiani (University of Turin), Kim Guldstrand Larsen (Aalborg University), Bernhard Beckert (KIT Karlsruhe), Mattias Ulbrich (KIT Karlsruhe), Reiko Heckel (University of Leicester), Vladimiro Sassone

(University of Southampton), Anton Wijs (Eindhoven University of Technology), and Nikolai Kosmatov (Thales Research and Technology, Palaiseau).

The ETAPS 2026 local organization team consisted of Maurice ter Beek (CNR-ISTI Pisa, general co-chair), Ferruccio Damiani (University of Turin, general co-chair), Barbara Boni (Synesthesia Turin, local organization chair), Vincenzo Ciancia (CNR-ISTI Pisa, satellite events co-chair), Luca Paolini (University of Turin, satellite events co-chair), Maria Tacconi (Synesthesia Turin, satellite events co-chair and publicity co-chair), Francesco Brocero (Synesthesia Turin, web co-chair and volunteers co-chair), José Proença (University of Porto, web co-chair), Gianluca Torta (University of Turin, publicity co-chair and local proceedings co-chair), Lucy James (Synesthesia Turin, sponsor chair), Giovanna Broccia (CNR-ISTI Pisa, local proceedings co-chair), Giorgio Audrito (University of Turin, volunteers co-chair), Riccardo Sieve (UiO Oslo, volunteers co-chair), and Reiner Hähnle (TU Darmstadt, wine chair).

I would like to take this opportunity to thank all authors, keynote speakers, invited tutorial speakers, and attendees. Special thanks goes to the organizers of the ETAPS 2026 satellite workshops and colocated events. ETAPS 2026 is grateful for the generous support of Amazon Web Services, AccessiWay, Camera di Commercio Industria Artigianato e Agricoltura di Torino, the Department of Computer Science of the University of Turin, Springer Nature, and Turismo Torino e provincia Convention Bureau. I thank our general co-chairs Maurice ter Beek (CNR-ISTI Pisa) and Ferruccio Damiani (University of Turin), who made it all happen in Turin, and their local organization team for their enormous efforts to make ETAPS 2026 a fantastic event. I am especially grateful to Barbara Boni, Maria Tacconi, and Lucy James (Synesthesia Turin) for handling the organizational process in a smooth and reliable way. Last but not least, a big thanks to Jan Kofroň for all his help as an ETAPS Fellow and providing online presence support for the ETAPS conferences and the ETAPS Association.

I hope you all enjoyed ETAPS 2026!

April 2026

Laura Kovács
ETAPS SC Chair, President of the ETAPS Association

Preface

FASE 2026, the 29th International Conference on Fundamental Approaches to Software Engineering, was held from April 11–16, 2026, in Turin, Italy, as part of the 29th ETAPS International Joint Conferences on Theory and Practice of Software (ETAPS 2026). FASE serves as a premier venue for researchers, developers, and users to discuss innovations in software engineering.

The topics of interest for FASE include: requirements, design, architecture, and modeling of software systems, applications of AI to software engineering and applications of software engineering to AI-based systems, software quality, model-driven engineering, software processes, as well as software evolution.

There were four submission categories for FASE 2026:

1. Research papers, which clearly identify and justify a principled advance to the fundamentals of software engineering.
2. Empirical-evaluation papers, which evaluate existing software challenges or critically validate current proposed solutions with scientific means, that is, by empirical studies, controlled experiments, rigorous case studies, and simulations.
3. New Ideas and Emerging Results (NIER) papers, which seek to disrupt the status quo with forward-looking, thought-provoking, innovative research on the foundations of software engineering, as well as lessons learned from the past.
4. Tool demonstration and data showcase papers, which present a new tool, a new tool component, novel extensions to an existing tool, or a new dataset.

This year, 68 papers were submitted to FASE. Three papers were desk rejected. The remaining 65 papers were distributed in categories 1–4 as follows: 45 research papers, 9 empirical-evaluation papers, 8 NIER papers, and 3 tool-demonstration and data showcase papers. Each paper underwent a double-blind peer review process, where three program committee members reviewed each submission. The review process spanned 9 weeks, ensuring thorough evaluation and discussion of submissions. It was possible to submit an artifact for evaluation alongside a paper, if made long-term available and declared in the Data-Availability Statement. The program committee extensively discussed the papers and ultimately decided to accept 21 papers included in these proceedings: 15 research papers, 3 empirical studies, 2 NIER papers, and 1 tool paper, resulting in an acceptance rate of 32%.

Artifacts comprise tools, models, proofs, or other data for validating the results of a paper. The artifact evaluation committee (AEC) reviewed the artifacts based on their documentation, ease of use, and, most importantly, whether the results presented in the corresponding paper could be accurately reproduced. As in 2025 FASE offered a joint voluntary artifact evaluation together with ESOP and FoSSaCS to authors of accepted papers. All of the 8 artifact submissions that were linked with accepted FASE 2026 submissions met the requirements for the “Artifacts Available” badge. In addition, 2 submissions were awarded the Artifacts “Evaluated – Functional” badge and 5 submissions the Artifacts “Evaluated – Reusable” badge.

FASE 2026 was proud to host an invited tutorial by Mieke Massink from the C.N.R. –Area della Ricerca di Pisa– Ist. ISTI. These proceedings contain the invited paper supporting the tutorial.

FASE 2026 also hosted Test-Comp 2026, the 8th International Competition on Software Testing. This event evaluated 21 tools for automatic test generation for C programs, where 11 test-generation tools were registered and actively supported by development teams, including one tool that participated for the first time. One coverage validator was run in four different configurations to evaluate the coverage of the test-suites produced by the test-generation tools. In addition, the new test-suite validation tool TestCoCa participated for the first time. The FASE 2026 proceedings contain a competition report by the Test-Comp chair and 5 short papers selected by the competition jury. The short papers describe 5 out of the tools participating with active team support. The 5 short papers were reviewed by a separate program committee (jury); each was assessed by at least three jury members. Two sessions in the FASE 2026 program were reserved for Test-Comp: (1) a presentation session with a report by the competition chair and summaries by the development teams of participating tools, and (2) an open community meeting in the second session, jointly with SV-COMP.

We would like to thank all the people who helped make FASE 2026 successful. First, we thank the authors for submitting their papers. The PC members and additional reviewers did a great job: they contributed informed and detailed reports and engaged in the PC discussions. We thank Reiner Hähnle and Marie-Christine Jakobs, initial and current chairs of the FASE steering committee, and Marieke Huisman and Laura Kóvacs, also initial and current chairs of the ETAPS steering committee, for their valuable advice. Lastly, we would like to thank the overall organization team of ETAPS 2026. We extend our gratitude to Gianluca Torta and Giovanna Broccia, the local proceedings chairs, for their diligent oversight of the proceedings preparation. We also thank the Artifact Evaluation Committee (AEC) for their assessment of submitted artifacts and the Test-Comp 2026 jury members for their evaluation of competition submissions. Additionally, we appreciate the editorial support of Springer Nature, as well as the local organizers of ETAPS 2026 in Turin, Italy, for their efforts in facilitating this event. Finally, we note the Gold Open Access publication of these proceedings in the Lecture Notes in Computer Science (LNCS) series, ensuring unrestricted access to the contributions presented at FASE 2026.

February 2026

Elvira Albert
Corina Pasareanu
Dirk Beyer
Yannic Noller

Organization

Program Committee

Erika Abraham	RWTH Aachen University, Germany
Elvira Albert	Universidad Complutense de Madrid, Spain
Dirk Beyer	LMU Munich, Germany
Artur Boronat	University of Leicester, UK
Tevfik Bultan	University of California at Santa Barbara, USA
Ana Cavalcanti	University of York, UK
Marsha Chechik	University of Toronto, Canada
Priyanka Darke	Tata Consultancy Services, India
Stijn De Gouw	The Open University, UK
Bernd Fischer	Stellenbosch University, South Africa
Gordon Fraser	University of Passau, Germany
Divya Gopinath	NASA Ames (KBR Inc.), USA
Marie-Christine Jakobs	LMU München, Germany
Susmit Jha	SRI International, USA
Martin Jonáš	Masaryk University, Czechia
Sun Jun	Singapore Management University, Singapore
Thierry Lecomte	CLEARSY, France
Ravi Mangal	Colorado State University, USA
Raffaela Mirandola	Politecnico di Milano, Italy
Corina Pasareanu	CMU, NASA, KBR, USA
Luigia Petre	Åbo Akademi University, Finland
Cesar Sanchez	IMDEA Software Institute, Spain
Natasha Sharygina	University of Lugano, Switzerland
Dániel Varró	Linköping University, Sweden and McGill University, Canada
Andrzej Wasowski	IT University of Copenhagen, Denmark
Haoze Wu	Amherst College, USA
Jianjun Zhao	Kyushu University, Japan

Test-Comp Program Committee and Jury

Dirk Beyer (Chair)	LMU Munich, Germany
Kaled Alshmrany	Institute of Public Administration, Saudi Arabia
Max Barth	LMU Munich, Germany

Zhenbang Chen	National University of Defense Technology, China
Marie-Christine Jakobs	LMU Munich, Germany
Martin Jonáš	Masaryk University, Brno, Czechia
Matthias Kettl	LMU Munich, Germany
Thomas Lemberger	LMU Munich, Germany
Christophe Meudec	South East Technological University, Ireland
Marek Trtík	Masaryk University, Brno, Czechia
Henrik Wachowitz	LMU Munich, Germany
Chenfeng Wei	University of Manchester, UK

Additional Reviewers

Mohammad Afzal
David A. Anisi
Aren A. Babikian
Daniel Baier
Anna Becchi
Konstantin Britikov
Matías Brizzio
Margarita Capretto
Boqi Chen
Bharti Chimdyalwar
Achintya Desai
Mara Downing
Grigory Fedyukovich
Attila Ficsor
Dominik Frey
Carolina Gerlach
Marek Jankola
Matthias Kettl
Rick Koenders
Shrawan Kumar
Faezeh Labbaf
Fabrizio Leopardi
Marian Lingsch-Rosenfeld
Kumar Madhukar
Juraj Major
Logan Murphy
Sahar Nasimi Nezhad
Nico Naus
Matthew Peng
Venkatesh Ramanathan
Moeketsi Raselimo
Pedro Ribeiro
Andoni Rodriguez
Chia Sabah
Laboni Sarker
Stefano Schivo
Vincenzo Scotti
Md Shafiuzzaman
Jeroen Spaans
Jan Strejček
Alexandra van der Spuy
Erik Voogd
Henrik Wachowitz
Yiran Wang
Philipp Wendler

Contents

Testing and Verification

Competition on Software Testing (Test-Comp 2026)

Search-based Software Testing for Drone Applications: An Experience with the Simulink Environment

Annalisa Sergi[1], Yousef Ahmed Abdel Rahman Shoeib[1], Andrea Bombarda[1], Nunzio Marco Bisceglia[1], and Claudio Menghi[1,2]

[1] University of Bergamo, Bergamo, Italy
{a.sergi2,y.shoeib,n.bisceglia1}@studenti.unibg.it
{andrea.bombarda,claudio.menghi}@.unibg.it
[2] McMaster University, Hamilton, Canada

Abstract. Unmanned aerial vehicles (UAVs) are frequently used in monitoring and inspection of large and isolated areas, and often use line following techniques to guide their movement. The successful execution of this task greatly depends on the correct design of the software controller. Search-based software testing (SBST) is a widely used technique to check for software defects. It iteratively generates test cases until either violations of the system requirements are detected or the time budget is exceeded. However, the effectiveness of SBST strongly depends on the application domain. This empirical evaluation paper assesses the effectiveness of SBST in supporting the design of UAV applications by considering a rigorous case study. It considers three different versions of a drone software controller. It assesses the capability of SBST in generating failure-revealing test cases and the usefulness of the test cases. Our results confirm the effectiveness of SBST and the usefulness of the generated test cases.

Keywords: Drone Controller, Model Development, Simulink, Search-based Software Testing

1 Introduction

Unmanned aerial vehicles (UAVs), a.k.a. drones, obtained significant attention in recent years due to their wide range of applications [35]. Drones are cyber-physical systems used for aerial inspection, delivery services, defense [19], environmental monitoring, and search-and-rescue missions [51]. Many application scenarios where drones are employed are safety-critical or mission-critical: The development of reliable and adaptable control systems for such vehicles is crucial, particularly when operating in dynamic or partially structured environments.

Autonomous flying is a desirable feature for drones, as they are frequently used in monitoring and inspection of large and isolated areas [9, 54]. In such situations, the GPS signal is not guaranteed; hence, other methods of local position

E. Albert and C. Pasareanu (Eds.): FASE 2026, LNCS 16504, pp. 304–326, 2026.
https://doi.org/10.1007/978-3-032-22774-4_16

feedback are required. One of the most adopted position feedback methods is the line following [10, 5]: A line represents the track to be followed by the drone, which exploits cameras or sensors to track it and controls its motors as needed. Line-following is important benchmark for evaluating autonomous navigation capabilities [44, 45], sensor integration, and control software robustness.

Assuring the dependability and reliability of drones during line following is a critical research concern, as demonstrated by the existence of UAV testing competitions (e.g., [46]). The successful execution of this task is highly dependent on the correct design of the software controller and on the interaction between software components and their environmental conditions, making it particularly susceptible to subtle faults and performance degradation. For this reason, several testing techniques have been applied to drone software testing, such as simulation-based [31], metamorphic [23], and model-based [47] testing.

Search-Based Software Testing (SBST) employs metaheuristic optimization to generate test cases that aim at detecting failure-revealing test cases. It is widely applied to CPSs development [3, 28] in various domains, such as real-time, concurrent, distributed, embedded, and safety-critical systems [1]. SBST iteratively generates test cases until violations of the system requirements are detected [26] or the time budget for testing is exceeded. Despite being widely recognized as useful tools, the effectiveness and applicability of SBST test generators are strongly affected by their application domain [25]. An SBST tool that is effective in one domain may be less effective in another, or may require customizations that are not easily implementable.

To increase the use of SBST for drone design, engineers require precise indications of its efficiency and effectiveness in this domain. Practitioners need guidelines and lessons learned that discuss if, how, and when SBST is useful. They also need studies that assess different tools to understand their level of maturity [1]. The assessment of the efficiency of tools and the replication of experiments in different domains is widely recognized as a need by the research and industrial communities [1, 13, 34, 42, 48, 38, 14, 41]. This need is particularly relevant in the context of CPS model-driven development [6, 49, 7, 8], where models are typically created and maintained in an industrial contexts and not publicly available due to confidentiality agreements or license restrictions [4, 11].

In this empirical evaluation paper, we evaluate the effectiveness of SBST in the context of drone applications via a rigorous case study. We consider `S-TaLiRo` [2] as an SBST tool. It searches for test inputs violating a requirement of interest by applying evolutionary algorithms and/or stochastic optimization techniques (e.g., Monte-Carlo, Ant-Colony optimization, and Simulated Annealing). Our case study is the Parrot minidrone software controller we have been developing for the MathWorks Minidrone Competition [50]. More specifically, we iteratively and incrementally developed a line following controller for the drone under study. We assess how SBST supports testing our controllers.

We extensively used the SBST framework for testing three different versions of the drone controllers, including the last one that successfully participated in the MathWorks Minidrone Competition, obtaining remarkable results. By run-

Fig. 1: Parrot SA Mambo Fly Minidrone.

ning software-in-the-loop (SIL) experiments, we assessed (RQ1) how effective SBST is in generating failure-revealing test cases, and how different search algorithms compare, and (RQ2) how useful the failure-revealing test cases produced by SBST are. For RQ1, our results confirm the effectiveness of SBST in generating failure-revealing test cases and that Uniform Random (UR) is more effective than Simulated Annealing (SA) for version V3 of the controller and comparable for the other two versions. For RQ2, our results confirmed the usefulness of the failure-revealing test cases: All the behaviors identified by the framework were violating the requirements and helped identify weaknesses in the controller. Surprisingly, SBST identified requirements violations for version V3 of the controller, which was considered correct during manual validation. Finally, we present our reflections and lessons learned on applying SBST (and `S-TaLiRo`) to a new and significantly different problem.

This work is organized as follows. Section 2 describes our drone case study and introduces the development of control software. Section 3 presents SBST and describes how we have extended `S-TaLiRo` to support track generation. Section 4 reports our evaluation methodology and results. Section 5 discusses results, lessons learned, and the improvement of the state of practice. Section 6 presents related works. Section 7 concludes the work.

2 Drone Study Subject

This section presents our drone case study subject: The controlled system (Section 2.1), its requirements (Section 2.2), and the models of its controller we developed and tested (Section 2.3).

2.1 Controlled System

The controlled system is the Parrot SA Mambo Fly Minidrone from Figure 1. It is a drone platform produced by Parrot SA [43] and used in the MathWorks Minidrone Competition [50]. This competition offers a standardized environment in which participants design and implement control algorithms for a simulated or physical drone vehicle. The drone is modeled as a nonlinear dynamic system with six degrees of freedom. The drone actuators consist of four rotors. The thrust forces generated by the four rotors collectively determine translational movement along the three spatial axes as well as angular orientation through roll, pitch, and

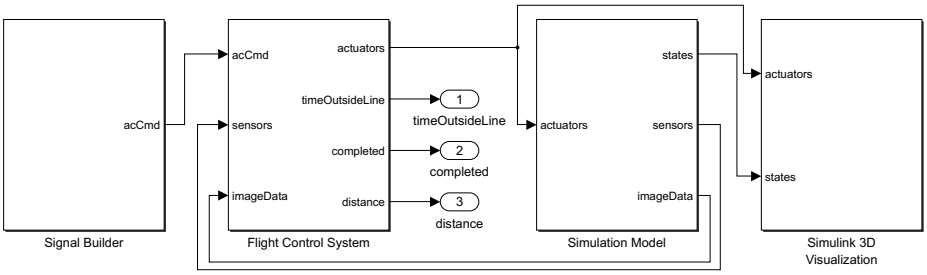


Fig. 2: The `Simulink` model.

yaw. Additionally, the controlled system possesses a downward-facing camera that can be used to gather information from the environment. Figure 2 shows the `Simulink` model of the drone used in the competition. The model consists of four components. The *Signal Builder* component is responsible for providing input commands for the simulation (*acCmd*) to the *Flight Control System* component. The *Flight Control System* component contains the flight control algorithm and has three input ports. It receives input from the cameras (*imageData*) for image processing, input commands (*acCmd*) from the *Signal Builder* component, and a signal (*sensors*) from the *Simulation Model* component for the path planning logic. Furthermore, the *Flight Control System* has four output ports. It sends commands to the actuators in the *Simulation Model* component (*actuators*), and provides three outputs used by our SBST framework: the current time the drone is off the line (*timeOutsideLine*), the track completion status (*completed*), and the current distance between the center of the drone and the closest border of the line (*distance*). The *Simulation Model* component encapsulates the models of the drone, the sensors, and the environment. It has one input port (*actuators*) for the commands to the actuators from the *Flight Control System* component, and three output ports: the data from the different sensors (*imageData* and *sensors*) is sent to the *Flight Control System* component, and a state signal (*states*) is sent to the *Simulink 3D Visualization* component, which also takes the actuator signal from the *Flight Control System* component to visualize the drone in the simulation environment.

2.2 Requirements

The competition requires participants to develop a line-following navigation algorithm on a track. The drone is tested across different unknown tracks (e.g., the one track from Figure 3). The track is represented by a 10 *cm* wide red line, and the drone is requested to land when the line ends on a circular marker having its center 25 *cm* from the end of the line and with a 10 *cm* radius. The number of sections that compose each track is always between 2 and 13. The angles between two contiguous sections are between 15 and 345 degrees. Intersections or very close (parallel) sections are not enabled in the competition.

The performance of the controllers is assessed by considering:

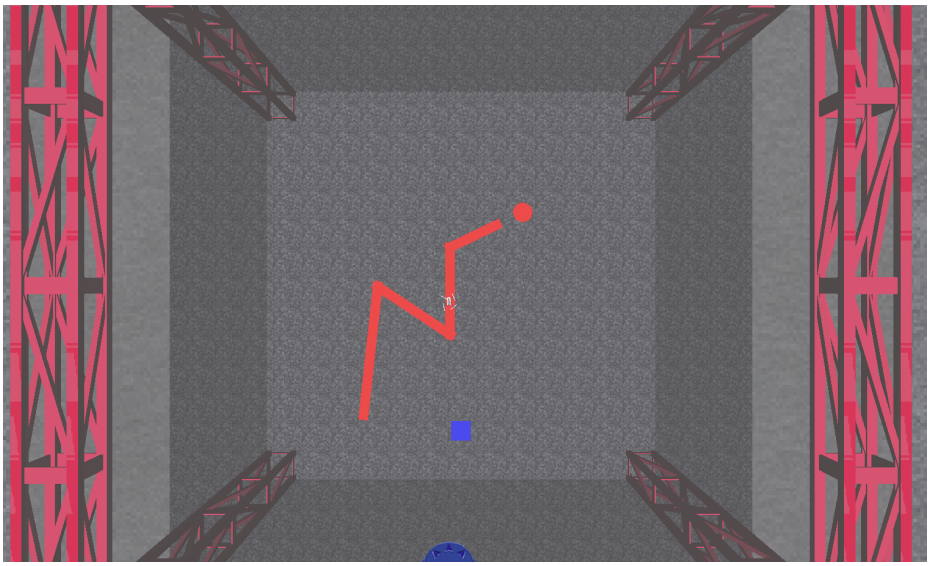


Fig. 3: The MathWorks Minidrone Competition arena.

1. the capability of the model to generate code to be uploaded to the drone,
2. the capability of the controller to land accurately and "softly": A landing is considered soft if the altitude is reduced gradually, and accurate if the drone lands within the circle with at least 20% of its total body, and
3. the accuracy of the drone following the track, measured with the number of straight sections completed (1 point per section).

In case of a tie between two or more teams, the deciding factor will be time (fastest wins).

To design the controller for our drone, we consider the requirements from Table 1. Specifically, Requirement R1 mandates the drone to be out of track for at most 1.25 seconds. This requirement is not explicitly mentioned in the competition rules: Adding this requirement helps distinguish between the accuracy of different versions of the controller. Requirement R2 mandates the drone to always land in the circular marker at the end of the track. Requirement R3 mandates the maximum horizontal distance between the center of the drone and the closest border of the track to be less than or equal to 12.5 *cm*. Indeed, since the drone is 13.21 *cm* wide, the distance of 12.5 *cm* is the one assuring an acceptable overlap between the drone and the track (both on the left or on the right).

2.3 Controller

The drone controller was developed by two of the authors, both undergraduate students in computer science engineering. It is embedded in the *Flight Control System* subsystem from Figure 2. The controller uses the camera to detect the

Table 1: Requirements considered for designing the controller of our drone.

ID	Description
R1	The drone shall be out of the track for at most 1.25 seconds.
R2	The drone shall always land in the circular marker at the end of the track.
R3	The distance between the drone and the track shall not exceed 12.5 *cm*.

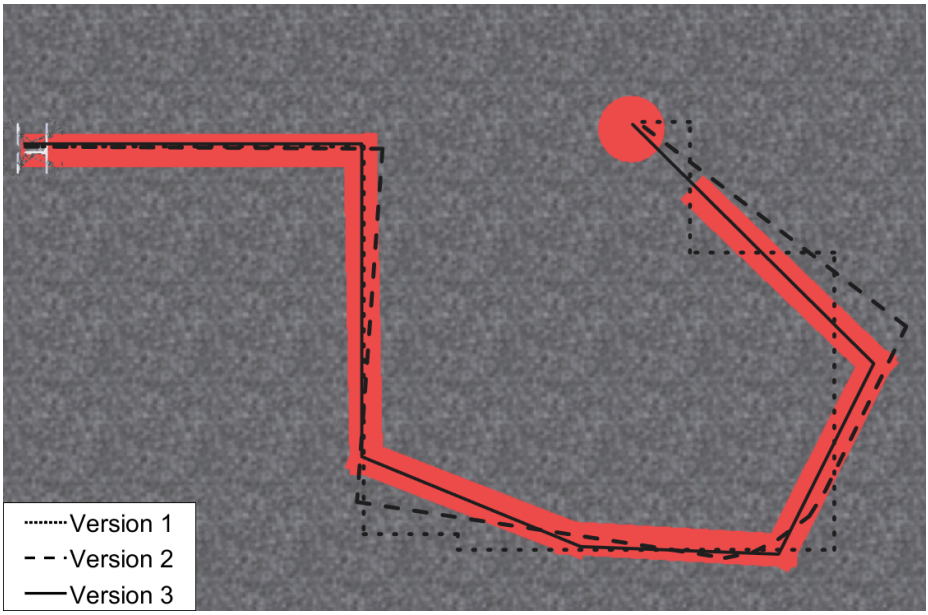


Fig. 4: Examples of paths for the versions V1, V2, and V3 of the controller.

position and orientation of the track relative to the drone's body frame. The input received by the camera is processed by the control algorithm, which computes appropriate rotor commands to maintain alignment with the track while ensuring stability of flight. Thus, the controlled system is a closed-loop interaction between the drone's motors, the environment, including the track, and the line-following control logic.

Table 2 summarizes the versions of the controller developed by the students. The development activity required approximately 400 hours. This time includes the time required for developing the different controllers and the time needed to test them. Figure 4 shows a track and three examples of paths obtained with the three different versions of the drone controller we have designed. The paths for versions V1, V2, and V3 are depicted in yellow, green, and blue.

The versions of the controller we developed are as follows:

– **Version V1**. Divides the input image from the camera into four portions (i.e., top, bottom, left, and right). For each portion, the number of pixels

Table 2: Identifier (ID), number of blocks (#B), and description and supported tracks for the controllers developed in this work.

ID	#B	Description and supported tracks
V1	47	Checks for the line in each of the 4 directions: forward, back, left, and right. Supports tracks with long segments and ideally 90-degree angles aligned with the x and y axes.
V2	44	Checks for the line in front of the drone until the end of the current segment is identified. Then, it rotates the image acquired by the drone until the next segment is in front of it. Supports tracks with medium-length segments and angles above 60 degrees.
V3	44	Checks in the section in front for the end of the current segment and the sides to know which direction to turn next. Turns the drone at the end of the segment until the next segment is in front of the drone. Supports the competition advanced tracks with a minimum distance between lines of 20 *cm* and a minimum angle of 15 degrees.

corresponding to the color of the line is computed. The drone uses this information to maintain or adjust its current direction and proceed exclusively along the X and Y axes, resulting in step-like trajectories and non-straight paths.

- **Version V2**. Divides the input image from the camera into three portions: two lateral portions (i.e., left and right) and a smaller area located at the front of the drone. The drone alternates between linear trajectories along the X and Y axes and rotations along the Z axis.
  This controller makes the drone follow a linear trajectory when the computed number of pixels in a 5×5 control matrix corresponding to the line color in the front portion is higher than a threshold of 4 pixels. When the number of pixels is lower than the threshold, the drone is approaching a corner: it stops, computes the number of pixels for the line color in the lateral portions (i.e., left and right), and performs a rotation of the image around the Z axis (i.e, counterclockwise and clockwise, respectively) depending on the pixels in the two areas. The rotation ends when a new line is revealed in the front portion, and the drone resumes a linear motion along the new direction.
  The controller regulates the flight speed based on the number of pixels corresponding to the line color revealed at the front, slowing down in proximity of a corner. Compared with version V1, the track navigation is smoother.
- **Version V3 (final)**. Relies on a finite-state machine with four states, as reported in Figure 5 and described in the following:
  - **State 0 (Take-off)**. The controller checks whether the drone is aligned with the line, counting the number of pixels in a 29×27 matrix sensed by the onboard camera under the center of the drone. If the number of pixels is between 324 and 360 pixels, the controller transitions to State 1; otherwise, it transitions to State 2.

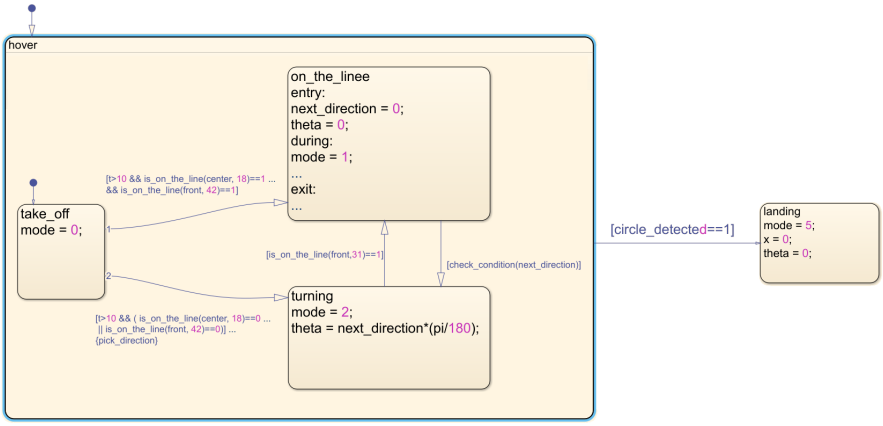


Fig. 5: State machine.

- **State 1 (On-the-line)**. The drone is facing the right direction and proceeds until the end of the current segment, using a matrix that checks the number of pixels corresponding to the line color revealed by the front drone's onboard camera. Furthermore, the new direction to follow is computed using lateral matrices. The idea of using a matrix in the front and lateral matrices comes from the previous version (V2) of the controller. Size and positions of the matrices are not the same since this controller can stay within the line more precisely: It splits the central matrix into 2 halves (left and right) and performs micro adjustments to stay exactly in the middle of the line.
- **State 2 (Turning)**. The drone is not facing the right direction or has reached the end of a segment, and no circle is detected. The end of a segment is detected when the pixels corresponding to the line color revealed at the front of the drone are zero, and the pixels with the color of the line detected by the lateral matrices are more than zero. The lateral matrix control was added since there is a break in the track between the landing spot and the end of the last segment. If this situation is detected, the controller should transition to State 3. Otherwise, the drone stops, rotates along the Z axis until it aligns with the new line, and transitions to State 1.
- **State 3 (Landing)**. Lands the drone.

We extensively used SBST to search for software behaviors of the versions V1, V2, and V3 of the controller that violate the specification from requirements R1, R2, and R3 as detailed in the next section. We emphasize that we employed SBST during development to uncover issues or vulnerabilities in the controller and to address them in the next version.

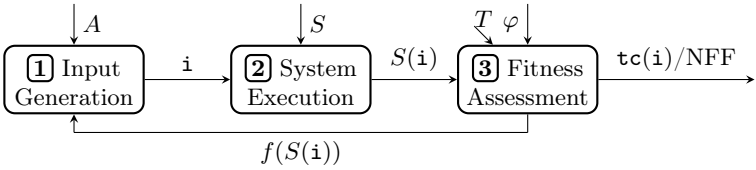


Fig. 6: SBST: An Overview.

3 Testing the Controller with SBST

SBST aims at generating failure-revealing inputs for the system under test. An overview of the SBST process is provided by Figure 6. Each box represents a step of SBST. Incoming and outgoing arrows represent inputs and outputs. Arrows with no source or destination represent inputs and outputs of the SBST framework.

When using SBST frameworks, users must provide a model of the system to be tested (S), an assumption on its inputs (A), a time budget (T), and a requirement (φ). SBST provides as output a failure-revealing test case ($\texttt{tc}(\texttt{i})$) or an indication that no failure-revealing test case was found (NFF — No Failure Found) within the time budget.

To produce this output, SBST iteratively performs the steps from Figure 6:

- *Input Generation* (1). The SBST framework generates an input ($\texttt{i}$) for the model (S) compliant with the assumption (A);
- *System Execution* (2). The SBST framework runs the system model (S) by providing the generated input ($\texttt{i}$) and obtaining a system execution ($S(\texttt{i})$);
- *Fitness Assessment* (3). The fitness value ($f(S(\texttt{i}))$) associated with the system execution ($S(\texttt{i})$) is evaluated by the SBST framework that assesses whether the fitness value is below a threshold value.

For each input ($\texttt{i}$), the SBST framework evaluates whether the associated test case ($\texttt{tc}(\texttt{i})$) is failure-revealing, i.e., (a) the input satisfies the assumption (A), and (b) the fitness value ($f(S(\texttt{i}))$) is smaller than a threshold value (typically the value 0). The fitness value is negative if the property is violated and positive otherwise. In addition, the higher the positive fitness value, the further the system is from violating its requirement; the lower negative values indicate that the system is further from satisfying its requirement [18, 17, 52, 40, 15].

The fitness value ($f(S(\texttt{i}))$) guides the search algorithm used by the SBST framework, which searches for an input associated with a negative fitness value. Specifically, SBST uses the fitness value computed in step 3 to drive the generation of the next input (1). The SBST framework terminates when a failure-revealing test case is detected or when the available time budget (T) expires and no failure-revealing test case is found. For the former case, SBST returns the failure-revealing test case. Note that, while many failure revealing test cases may exist for the S, SBST returns only one of them. For the latter case, SBST returns the NFF value.

The *Input Generation*, *System Execution*, and *Fitness Assessment* components are shared and implemented by many SBST tools, e.g., `ARIsTEO` [39], `ATheNA` [20], `S-TaLiRo` [2], `Breach` [12], `HECATE` [21], `FalStar` [16], `FalCAuN` [52], `falsify` [53], `FalStar` [16], and `ForeSee` [55].

SBST can be instantiated by considering different modeling formalisms. In this work, we assume that the CPS is modeled as a reactive system that receives inputs (produced by the *Input Generation* component) and produces outputs that can be monitored (used by the *Fitness Assessment* component). Alternative instances of SBST differ in the implementation of the *Input Generation*, *System Execution*, and *Fitness Assessment* components. In this work, we consider `S-TaLiRo` [2] as our SBST framework. `S-TaLiRo` is a toolbox for temporal logic falsification within the `MATLAB` environment. It searches for counterexamples to Metric Temporal Logic (MTL) properties in Simulink/Stateflow diagrams, minimizing a robustness metric. It uses stochastic optimization techniques, including Monte-Carlo methods and Ant-Colony Optimization, to perform a random walk over the initial states, controls, and disturbances of the system. The search returns the simulation trace with the smallest robustness value that was found. The lower the value of robustness, the closer the property is to being falsified by the tool. A simulation trace with a negative robustness indicates that the temporal logic properties were falsified.

To use `S-TaLiRo`, we customized the *Input Generation* (1) component to generate tracks for the drones. This was done as follows.

3.1 Track Generation

We modified the *Input Generation* (1) component to generate different tracks according to the competition rules, as specified in Algorithm 1.

Algorithm 1 takes inpRanges and prevSample as input for the GENERATE-TRACK function. The first defines the minimum and maximum coordinates for each point that connect the track segments, and the latter is the previously generated track. Algorithm 1 behaves as follows. Line 2 generates a new track. Line 3 checks if the track is compliant with the rules of the competition. In particular, a track will be discarded if (a) the angle between two consecutive sections is lower than 15 degrees or higher than 345 degrees, (b) the distance between two segments is lower than 20 *cm*, and (c) the track has one or more intersections. If the track is compliant, Line 6 calls the INTERPOLATE function from Algorithm 2 and the track is returned (Line 7). Otherwise, Line 4 generates a new sample. In the first iteration, prevSample is null and the sample is randomly generated. From the second iteration, `S-TaLiRo` slightly updates the prevSample, while still complying with the possible input range (inpRange).

Once the coordinates of the track satisfy the constraints, the set of coordinates (curSample) is passed as input (Line 6) to the custom interpolation function INTERPOLATE from Algorithm 2. Indeed, `S-TaLiRo` allows users to pass a function pointer as the interpolation method. Algorithm 2 behaves as follows. Line 2 transforms the 10-coordinate array into a matrix with each row representing a point. Line 3 generates the coordinates of the points composing the track (tiles)

Algorithm 1 Input Generation component.

```
1: function generateTrack(inpRanges,prevSample)
2:     curSample=getNewSample(inpRanges,prevSample);
3:     while checkTrack(curSample) == 0 do
4:         curSample =getNewSample(inpRanges,prevSample);
5:     end while
6:     track=interpolate(curSample)
7:     return track
8: end function
```

Algorithm 2 Track Interpolation.

```
1: function interpolate(curSample)
2:     points = arrToPointsMatrix(curSample);
3:     [tiles,land] = linePatches(points);
4:     lines2wrl(tiles,land);
5: end function
```

and the center of the landing circle (land) from the points previously generated. Line 4 updates the virtual world of the competition arena using the function lines2wrl, receiving as parameters the lines and the final circular marker in simulation.

4 Evaluation

Our evaluation assesses the usefulness of SBST in testing drone control software, considering the following research questions:

RQ1: How *effective* is SBST in generating failure-revealing test cases, and how do different search algorithms compare? (Section 4.2)

RQ2: How *useful* are the failure-revealing test cases produced by SBST? (Section 4.3)

RQ1 assesses the performance of two different algorithms to assess how the choice of the algorithm influences the effectiveness of SBST within drone scenarios. RQ2 assesses the usefulness of SBST in detecting model failures.

4.1 Experimental Setup

To run our experiments, we formalized our requirements in MTL/STL as

$$\psi := \Box(timeOutsideLine \leq 250) \wedge \Diamond(completed = 1) \wedge \Box(distance \leq 25).$$

The symbols $\Box$ and $\Diamond$ represent the *always* and *eventually* temporal logic operators. The first part of the specification ($r_1 := \Box(timeOutsideLine \leq 250)$) encodes requirement R1 from Table 1 and requires that the time the drone is

Table 3: Configuration parameters for `S-TaLiRo`.

Parameter	**Value**
Search algorithms	UR, SA
Number of runs	10
Maximum number of iterations per run	1500
Time budget (T)	300 s

Table 4: Experimental results for the Drone case study.

	UR			**SA**		
Controller	**FR**	$\bar{\mathbf{S}}$	$\hat{\mathbf{S}}$	**FR**	$\bar{\mathbf{S}}$	$\hat{\mathbf{S}}$
Version 1	10/10	1.0	1.2	10/10	1.0	3.5
Version 2	10/10	1.0	1.4	10/10	1.0	1.5
Version 3	10/10	45.0	69.1	10/10	122.0	132.6

out of the track is less than 250 iterations. Since there are 200 iterations in a second, 250 iterations are equivalent to 1.25 seconds. The second part of the specification ($r_2 \coloneqq \Diamond(completed = 1)$) encodes requirement R2 from Table 1 and requires the drone to always land in the circular marker at the end of the track. If *completed* equals 1, the drone successfully landed; otherwise, *completed* is equal to 0. The third part of the formula ($r_3 \coloneqq \Box(distance \leq 25)$) encodes requirement R3 from Table 1 and requires that the maximum distance between the drone and the line shall always be less than or equal to 25 pixels (12.5 cm). The specification ψ is automatically mapped to a fitness function by S-Taliro [2].

We assume that the arena considered to generate the tracks is a 4 $m \times 4$ m. We generated tracks with five points represented by (X, Y) coordinates, which must be within the boundaries of the arena.

Table 3 reports the configuration parameters considered for our experiments. We considered two search algorithms: Uniform Random (UR) and Simulated Annealing (SA). To evaluate the effectiveness of our SBST framework implementation in generating failure-revealing test cases, we performed six (3 models $\times$ 2 search algorithms) experiments, each obtained by considering one version of the three controllers from Section 2.3 and one of the search algorithms.

We ran our SBST framework for 10 runs to account for the nondeterminism of the search algorithms. The experiments were executed on a personal laptop.[3]

4.2 Effectiveness and Comparison (RQ1)

Table 4 shows our experimental results. Each row represents the version of the drone controller. The table is divided into two parts related to the UR and SA search algorithms. For each experiment, the table outlines the falsification rate (FR), i.e., the number of times our SBST framework found failure-revealing

[3] Intel(R) Core(TM) i7-13700K 3.4 GHz CPU, 16 cores, 32 GB of RAM

test cases within the given time, and the average ($\hat{S}$) and the median number ($\bar{S}$) of iterations required to find the test cases. Note that `S-TaLiRo` does not indicate which specific requirement (e.g., r_1, r_2, or r_3) was violated and caused the falsification of ψ.

For all versions of the controller, our SBST framework could generate a failure-revealing test case. Therefore, all versions of the controller may cause the drone to violate one of the three requirements of Table 1. This confirms the effectiveness of SBST in generating failure-revealing test cases.

Both Uniform Random (UR) and Simulated Annealing (SA) showed a 10/10 falsification rate in all the experiments. While comparing the number of iterations required by UR and SA for versions V1 and V2 is meaningless since they both require a very limited number of iterations, the comparison is relevant for version V3. Surprisingly, UR performed better than SA for version V3 of the drone controller. The average ($\hat{S}$) and the median number ($\bar{S}$) of iterations required to identify the failure-revealing test case show that for version V3 of the controller, UR required fewer iterations (69.1 vs 132.6 for the average, and 45.0 vs 122.0 for the median) to identify the failure-revealing test cases than SA. This result is caused by our current definition of the fitness function, which cannot efficiently guide the search procedure employed by SA. This behavior is due to the improvements made to version V3 of the controller, which reduces the MTL specification to just the predicate R2, because the requirements R1 and R3 are always satisfied. In particular, the drone is always able to follow the track, so the time outside the line is always less than 250 iterations, meaning that R1 is always true. Furthermore, the distance between the drone and the line is always less than 25 pixels, meaning that R3 is always true. The only unsatisfied predicate remains R2, which, however, is not a good metric to efficiently guide the search because it remains false until the track is completed. This behavior makes the uniform random algorithm a better solution than using a fitness function.

Across all our experiments, the SBST framework took on average $32'\,38''$ (min=$20''$, max=$4h\,3'\,59''$, $StdDev$=$59'\,34''$) and 34.88 iterations (min=1, max=235, $StdDev$=63.80) to detect failure-revealing test cases.

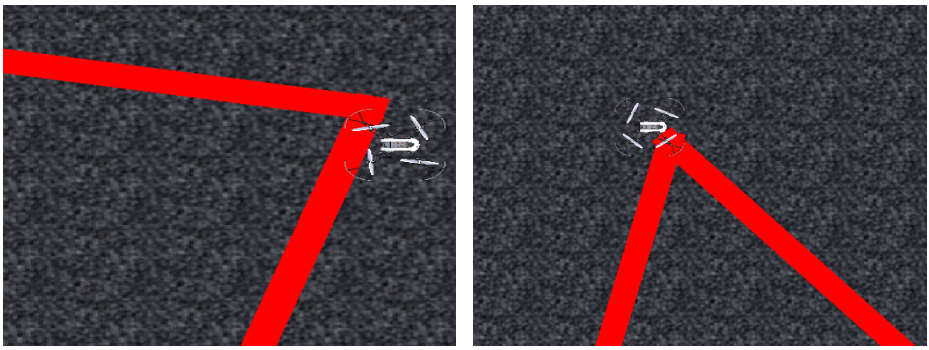
RQ1: Effectiveness

Our results confirm the effectiveness of SBST in generating failure-revealing test cases. They also show that, for version V3 of the controller, UR is more effective than SA, while for versions V1 and V2, the results are comparable.

4.3 Usefulness (RQ2)

We manually inspected the failure-revealing test cases returned by our SBST framework to analyze our software failures. In the following, we summarize our findings for each version of the controller.

Version V1. Version V1 could execute diagonal segments by proceeding exclusively along the X and Y axes, resulting in step-like trajectories. Therefore, the drone was unable to complete the entire track due to its inability to navigate



(a) Failure for Version V1. (b) Failure for Version V2

Fig. 7: Failures for versions V1 and V2 of the controller.

non-right angles, resulting in the drone getting stuck in corners and failing the requirements R1 and R2. As shown in Figure 7a, the drone is not able to find the next direction and is stuck in the corner.

Version 2. The problem from version V1 was fixed by rotating the image depending on the number of colored pixels in the left and right front (and lateral) sides of the drone. However, it was unable to navigate acute corners under 60 degrees due to reaching a position where there were no colored pixels in the left and right front (and lateral), i.e., the drone shall navigate "backwards". Figure 7b shows an example of such a track.

Version 3. Surprisingly, our SBST framework was effective in finding a failure-revealing test case also for this version. Table 5 summarizes the four failures identified by our test cases. For each failure, it reports the number of the experiment (Experiment) in which the failure was identified and describes the failure (Description). When the same failure is identified by multiple experiments, the ID of each experiment is reported in the first column. Figure 8 outlines examples of tracks that lead to the violation of the requirements. For example, experiments 1, 3, 4, 6, 7, 9, and 10 show that version V3 of the controller was unable to overcome two short segments that were very close to each other (see Figure 8a), i.e., connected by a very acute angle. However, the problem was not the angle but the length of the segments. The drone turned too much during the corner, treating the current segment as the next one, and returning to the start point. Thus, it failed to find the final circular marker. However, the rules of the competition did not specify any minimum length of the segments. The problem would be solved with longer segments.

RQ2: Usefulness

The manual inspection confirmed the usefulness of the failure-revealing test cases: All the behaviors identified by the framework were violating the requirements and helped identify weaknesses in the controller. Surprisingly,

Table 5: Description of the failure identified by each SBST experiment.

ID	Experiments	Description
F1	1, 3, 4, 6, 7, 9, 10	The controller fails on a track with a very short last segment and a tight angle between the last segment and the previous one, such as in Figure 8a. The drone reaches the last turn and starts to turn to align with the track. However, it aligns itself with the previous segment due to the limited length of the last one and the tight angle.
F2	2	The controller fails on the track in Figure 8b on the second segment, where the drone was not able to align itself properly before starting to turn towards the next segment. This results in losing the line and going back to the start of the track without being able to finish.
F3	5	The controller fails on the track in Figure 8c because of an artifact on the last turn, which results from the way the track is created between points by simply overlapping segments of the right width. The areas of the image acquired by the drone camera and used to check if the drone is turning in the right direction intersect that artifact on the right edge. This mistakenly makes the drone turn right and go back on the same segment it came from, thus failing to complete the track.
F4	8	The controller fails on the turn between the second and the third segment due to the fourth segment being very close to the end of the second one, as shown in Figure 8d.

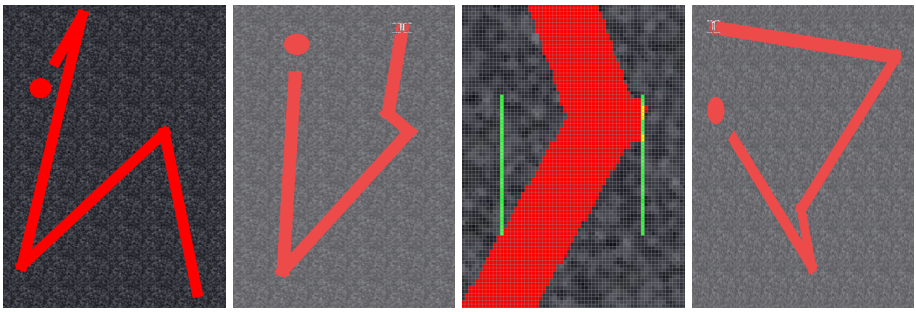
SBST identified requirements violations for version V3 of the controller, which was considered correct.

5 Discussion

We discuss lessons learned (Section 5.1) and threats to validity (Section 5.2).

5.1 Lessons Learned

Competition regulation. The SBST framework we established could identify a failure-revealing test case also for the final version (i.e., version V3) of our implementation of the Parrot minidrone software controller, which obtained remarkable results in the MathWorks Minidrone Competition. In Section 4.3, we discussed that the issue with the controller was the presence of very short segments connected by a very acute angle. The competition rules did not specify any minimum length of the segments or any other constraints, except for the minimum distance between two lines. We contacted MathWorks for clarification. Our objective is to understand if there is missing information in the competition rules, or if we should consider these test cases as failure-revealing. From



(a) Failure F1. (b) Failure F2. (c) Failure F3. (d) Failure F4.

Fig. 8: Failures for version V3 of the controller.

this experience, we learned that SBST is useful to identify failure-revealing test cases. However, in some cases, its usage can also pinpoint other problems (e.g., underspecified requirements or missing assumptions).

SBST effectiveness. We were confident about the drone's ability to follow any track for the last version of the controller. However, our experimental results showed that our SBST framework was effective in finding a failure-revealing test case (with a 10/10 falsification rate) for each experiment, identifying paths that do not allow the drone to reach the end of the track. From this result, we learned that engineers may be overconfident about their product. After investing a long time and significant effort in their design activity, they may overlook certain situations, configurations, and behaviors that lead to failures. Using SBST helps identify these behaviors and corner cases. Therefore, we expect similar SBST-based approaches to benefit all other competition participants. Since all competition participants develop within the same environment and under the same set of requirements, adapting the methodology presented in this paper to other drone controllers should be straightforward.

Tool customization. There are many SBST tools within the research literature (e.g., ARIsTEO [39], `ATheNA` [20], `S-TaLiRo` [2], `Breach` [12], `HECATE` [21], `FalStar` [16], `FalCAuN` [52], `falsify` [53], `FalStar` [16], and `ForeSee` [55]). However, applying these tools in different domains requires customization. For example, in our case, we had to modify the *Input Generation* (1) component to generate tracks (see Section 3.1). `S-TaLiRo` easily supports the customization of the *Input Generation*. However, track generation wasn't trivial and required considerable time to design a component that correctly generates tracks that satisfy the rules of the competitions. From this experience, we learned that while the high-level behavior of SBST is common and shared across many applications, different applications have peculiar needs that may require (non-trivial) customizations that can affect the effectiveness and efficiency of SBST. Similarly, we observed that tailoring SBST tools to different application scenarios can be labor-intensive, which may restrict the set of tools that can reasonably be considered in analyses such as the one presented in this paper.

SBST during development. In our case study, we implemented three versions of the Parrot minidrone software controller and assessed their ability to follow the lines using our SBST framework. The use of SBST during the development phase would have been beneficial in discovering different paths that lead the drone to fail to follow the track, improving the quality of the controller. For example, they would have been useful to identify the problem with version V3 of the controller. From this experience, we learned that using SBST during development can be beneficial to identify bugs and problems.

Fitness function design. Our results show that our fitness function could not efficiently guide the search procedure employed by SA. From this result, we learned that a proper definition of the fitness function is crucial for the effectiveness of the search process. The use of techniques such as fitness-landscape analysis could help this process. These techniques evaluate how the fitness value changes over the search space and usually support the fitness function design, helping understand the search process and its probability of success [20]. However, the fitness function design remains a complex activity and may require a combination of automated and manually defined fitness functions to guide the search of the input domain in order to increase the probability of finding failures.

5.2 Threats to Validity

The requirements we considered in this study subject could threaten the *external validity* of our results since it influences how our they can extend to other study subjects. However, the fact that the requirements were defined following the MathWorks Minidrone Competition mitigates this threat.

Our conclusions are specific to the competition environment and its regulations and may not generalize to other settings in which drones are used but no equivalent line-following task, or the same requirements, exists. However, the approach presented in this paper is general, and alternative requirements can be encoded to accommodate different use cases.

The values assigned to the fitness function and other configuration parameters in `S-TaLiRo` could threaten the *internal validity* of our results: Other values may lead to different results. However, the fact that we made a replication package publicly available mitigates this threat as it helps replicate the experiments with different parameters.

6 Related Work

Recent advancements in drone technology, driven by progress in areas such as artificial intelligence, computer science, and obstacle avoidance, have significantly expanded the capabilities of drones and opened new possibilities across various industries [35]. However, given the criticality of these systems, ensuring their reliability and safety is paramount.

Khatiri *et al.* [31, 32] implemented a search-based approach that replicates real drone flights and generates simulation-based test cases by analyzing the field

test logs and created an experimental platform designed to automate various aspects of UAV system testing (e.g., test generation) in a simulation environment. This platform was adopted during the first UAV Testing Competition [33]. However, we developed our drone by participating in the MathWorks Minidrone Competition with particular focus on generating challenging tracks that the drone may fail to follow, rather than realistic ones.

Vierhauser *et al.* [51] created an approach that defines structured and reusable field test scenarios for drone missions, executes field tests, and collects data for processing and analysis. Javed *et al.* [29] proposed a hybrid approach that combines model-based testing and SBST of unmanned aircraft system (UAS) software to automate the generation and execution of test cases, improving fault detection. Lindvall *et al.* [36] developed a framework that combines metamorphic testing and model-based testing to automatically generate test cases from models that encode testing scenarios of autonomous drones in simulation. Hildebrandt *et al.* [27] proposed a world-in-the-loop (WIL) approach to reduce the gap between simulation and reality, integrating real sensor and simulation data into a running UAV. Our work differs from these contributions, focusing on the falsification of the STL specification using `S-TaLiRo` to generate trajectories that violate line following requirements, rather than deriving test cases from models, logs, or sensor integration.

Search-based techniques are widely used in the automotive domain. Gambi *et al.* [24] proposed an approach that combines SBST and procedural content generation to generate virtual roads to test autonomous cars. Kendall *et al.* [30] proposed a framework based on deep reinforcement learning to generate random roads for autonomous driving. Similarly, SBST is used by Marzella *et al.* [37] to test the controller of an e-bike. Compared to their work, our systems under test are drones, and our aim is to evaluate different versions of the drone controller in both simulated and competition environments.

7 Conclusion

This paper evaluates search-based software testing (SBST) for drone control software applications. We considered `S-TaLiRo`, a well-known SBST tool. We customized `S-TaLiRo` for track generation and assessing three software controllers against three requirements, capturing off-track time, lateral distance, and track completion status. We considered two search strategies (Uniform Random and Simulated Annealing). SBST consistently produced failure-revealing test cases (10/10 runs), uncovering subtle violations even in the final, competition-ready controller (V3) that manual validation had previously accepted. The generated failure-revealing test cases exposed controller limitations on short segments with acute angles, alignment/turning edge cases, and artifacts from track construction. Our results show that (a) domain-tailored input generation is required to make SBST applicable in different CPS domains, and (b) integrating SBST throughout development mitigates overconfidence in manual testing. As future work, we plan to improve fitness functions for landing (e.g., by considering the distance between the coordinates where the simulation stops and the landing

point), compare additional SBST tools and algorithms, and perform hardware-in-the-loop testing to study the consistency between results from in vitro simulations and real-world experimentation. Additionally, we plan to conduct further experiments with other SBST tools to assess the effort needed to adapt them to scenarios involving drones, as the one we used in this paper.

Data Availability A replication package containing all of our data, test results, and scripts is publicly available [22]. An additional artifact is available at `https://doi.org/10.5281/zenodo.18173673`.

References

1. Ali, S., Briand, L.C., Hemmati, H., Panesar-Walawege, R.K.: A systematic review of the application and empirical investigation of search-based test case generation. IEEE Transactions on Software Engineering **36**(6), 742–762 (2010). https://doi.org/10.1109/TSE.2009.52
2. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 254–257. Springer (2011)
3. Arrieta, A., Wang, S., Markiegi, U., Sagardui, G., Etxeberria, L.: Search-based test case generation for cyber-physical systems. In: IEEE Congress on Evolutionary Computation (CEC). p. 688–697. IEEE Press (2017). https://doi.org/10.1109/CEC.2017.7969377
4. Badreddin, O., Lethbridge, T.C., Elassar, M.: Modeling practices in open source software. In: Open Source Software: Quality Verification. pp. 127–139. Springer (2013)
5. Basso, M., Pignaton de Freitas, E.: A UAV guidance system using crop row detection and line follower algorithms. J. Intell. Robot. Syst. **97**(3-4), 605–621 (Mar 2020)
6. Boll, A., Brokhausen, F., Amorim, T., Kehrer, T., Vogelsang, A.: Characteristics, potentials, and limitations of open-source simulink projects for empirical research. Software and Systems Modeling **20**(6), 2111–2130 (2021)
7. Boll, A., Kehrer, T.: On the replicability of experimental tool evaluations in model-based development: Lessons learnt from a systematic literature review focusing on matlab/simulink. In: Systems Modelling and Management. p. 111–130. Springer (2020). https://doi.org/10.1007/978-3-030-58167-1_9
8. Boll, A., Vieregg, N., Kehrer, T.: Replicability of experimental tool evaluations in model-based software and systems engineering with matlab/simulink. Innovations in Systems and Software Engineering **20**(3), 209–224 (2024)
9. Bollard, B., Doshi, A., Gilbert, N., Poirot, C., Gillman, L.: Drone technology for monitoring protected areas in remote and fragile environments. Drones **6**(2) (2022). https://doi.org/10.3390/drones6020042
10. Brandão, A.S., Martins, F.N., Soneguetti, H.B.: A vision-based line following strategy for an autonomous uav. In: 2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO). vol. 02, pp. 314–319 (2015)
11. Ding, W., Liang, P., Tang, A., Vliet, H.v., Shahin, M.: How do open source communities document software architecture: An exploratory survey. In: International Conference on Engineering of Complex Computer Systems. p. 136–145. ICECCS, IEEE Computer Society (2014). https://doi.org/10.1109/ICECCS.2014.26

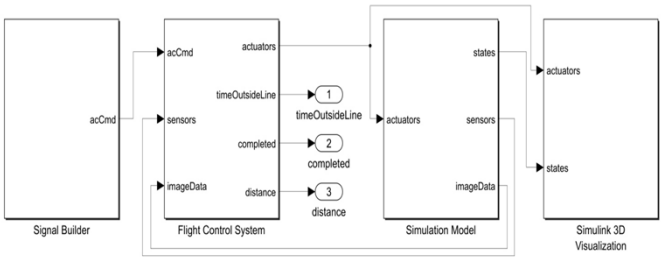
12. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P.B. (eds.) Computer Aided Verification. pp. 167–170. Springer, Berlin, Heidelberg (2010)
13. Dyba, T., Kitchenham, B.A., Jorgensen, M.: Evidence-based software engineering for practitioners. IEEE software **22**(1), 58–65 (2005)
14. Engström, E., Petersen, K.: Mapping software testing practice with software testing research — serp-test taxonomy. In: IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). pp. 1–4 (2015). https://doi.org/10.1109/ICSTW.2015.7107470
15. Ernst, G., Arcaini, P., Fainekos, G., Formica, F., Inoue, J., Khandait, T., Mahboob, M.M., Menghi, C., Pedrielli, G., Waga, M., Yamagata, Y., Zhang, Z.: ARCH-COMP 2022 Category Report: Falsification with Ubounded Resources. In: Frehse, G., Althoff, M., Schoitsch, E., Guiochet, J. (eds.) International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22). EPiC Series in Computing, vol. 90, pp. 204–221. EasyChair, Bramhall, Stockport (2022)
16. Ernst, G., Sedwards, S., Zhang, Z., Hasuo, I.: Fast falsification of hybrid systems using probabilistically adaptive input. In: Parker, D., Wolf, V. (eds.) International Conference on Quantitative Evaluation of Systems: (QEST). pp. 165–181. Springer, Cham (2019)
17. Fainekos, G., Hoxha, B., Sankaranarayanan, S.: Robustness of specifications and its applications to falsification, parameter mining, and runtime monitoring with s-taliro. In: Finkbeiner, B., Mariani, L. (eds.) International Conference on Runtime Verification. pp. 27–47. Springer, Cham (2019)
18. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications. In: Formal Approaches to Software Testing and Runtime Verification, pp. 178–192. Springer (2006)
19. Foreman, V.L., Favaró, F.M., Saleh, J.H., Johnson, C.W.: Software in military aviation and drone mishaps: Analysis and recommendations for the investigation process. Reliability Engineering & System Safety **137**, 101–111 (2015). https://doi.org/10.1016/j.ress.2015.01.006
20. Formica, F., Fan, T., Menghi, C.: Search-based software testing driven by automatically generated and manually defined fitness functions. ACM Transactions on Software Engineering and Methodology (TOSEM) (2023)
21. Formica, F., Fan, T., Rajhans, A., Pantelic, V., Lawford, M., Menghi, C.: Simulation-based testing of simulink models with test sequence and test assessment blocks. IEEE Transactions on Software Engineering (2023)
22. FOSELAB: Replication package for the paper "Search-based Software Testing for Drone Applications: An Experience with the Simulink Environment". `https://github.com/foselab/ParrotMinidroneCompetition` (2025), accessed: August 8, 2025
23. Fredericks, E.M., Jacobs, M., DeVries, B.: Towards a metamorphic testing architecture for software-defined drone systems. In: 2024 11th International Conference on Software Defined Systems (SDS). pp. 170–177 (2024). https://doi.org/10.1109/SDS64317.2024.10883896
24. Gambi, A., Mueller, M., Fraser, G.: Automatically testing self-driving cars with search-based procedural content generation. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 318–328 (2019)
25. Harman, M., Jia, Y., Zhang, Y.: Achievements, open problems and challenges for search based software testing. In: IEEE International Conference

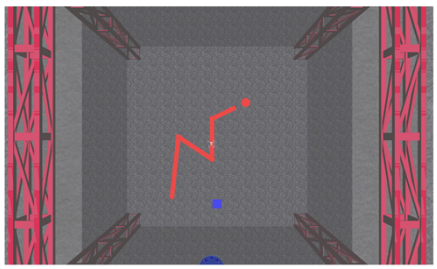
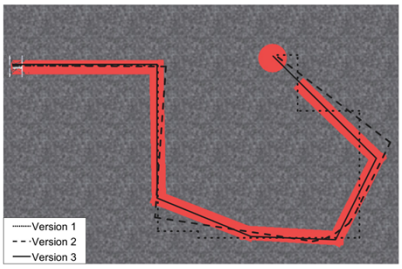
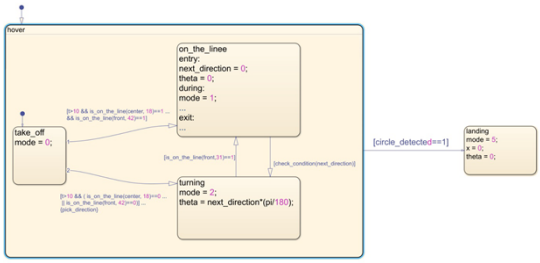
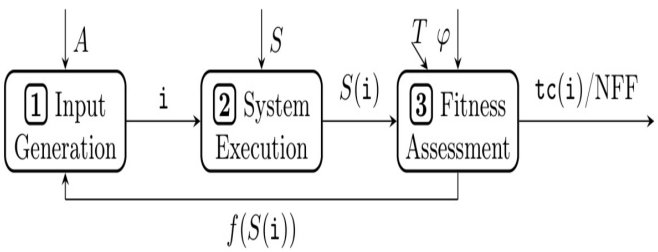
on Software Testing, Verification and Validation (ICST). pp. 1–12 (2015). https://doi.org/10.1109/ICST.2015.7102580
26. Harman, M., Jones, B.F.: Search-Based Software Engineering. Information and Software Technology **43**(14), 833–839 (2001). https://doi.org/10.1016/s0950-5849(01)00189-6
27. Hildebrandt, C., Elbaum, S.: World-in-the-loop simulation for autonomous systems validation. In: 2021 IEEE International Conference on Robotics and Automation (ICRA). pp. 10912–10919. IEEE (2021)
28. Humeniuk, D., Khomh, F., Antoniol, G.: A search-based framework for automatic generation of testing environments for cyber–physical systems. Information and Software Technology **149**, 106936 (Sep 2022). https://doi.org/10.1016/j.infsof.2022.106936
29. Javed, Z., Iqbal, M.Z., Khan, M.U., Usman, M., Jilani, A.A.A.: A hybrid search and model-based approach for testing the self-adaptive unmanned aircraft system software. Computer Standards & Interfaces **93**, 103959 (2025). https://doi.org/10.1016/j.csi.2024.103959
30. Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.M., Lam, V.D., Bewley, A., Shah, A.: Learning to drive in a day. In: 2019 international conference on robotics and automation (ICRA). pp. 8248–8254. IEEE (2019)
31. Khatiri, S., Panichella, S., Tonella, P.: Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights. In: 2023 IEEE Conference on Software Testing, Verification and Validation (ICST). pp. 281–292 (2023). https://doi.org/10.1109/ICST57152.2023.00034
32. Khatiri, S., Panichella, S., Tonella, P.: Simulation-based testing of unmanned aerial vehicles with aerialist. In: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings. p. 134–138. ICSE-Companion '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3639478.3640031
33. Khatiri, S., Saurabh, P., Zimmermann, T., Munasinghe, C., Birchler, C., Panichella, S.: Sbft tool competition 2024-cps-uav test case generation track. In: Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing. pp. 29–32 (2024)
34. Kitchenham, B., Dyba, T., Jorgensen, M.: Evidence-based software engineering. In: International Conference on Software Engineering. pp. 273–281 (2004). https://doi.org/10.1109/ICSE.2004.1317449
35. Le, N.B.v., Thai, H.D., Yoon, C.W., Huh, J.H.: Recent development of drone technology software engineering: A systematic survey. IEEE Access **12**, 128729–128751 (2024). https://doi.org/10.1109/ACCESS.2024.3454546
36. Lindvall, M., Porter, A., Magnusson, G., Schulze, C.: Metamorphic model-based testing of autonomous systems. In: 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET). pp. 35–41. IEEE (2017)
37. Marzella, M., Bombarda, A., Minervini, M., Bisceglia, N.M., Gargantini, A., Menghi, C.: Test case generation for simulink models: An experience from the e-bike domain. In: Search-Based Software Engineering. pp. 1–15. Springer Nature Switzerland, Cham (2025)
38. Melo, S.M., Carver, J.C., Souza, P.S., Souza, S.R.: Empirical Research on Concurrent Software Testing: A Systematic Mapping Study. Information and Software Technology **105**, 226–251 (2019)
39. Menghi, C., Nejati, S., Briand, L.C., Parache, Y.I.: Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system

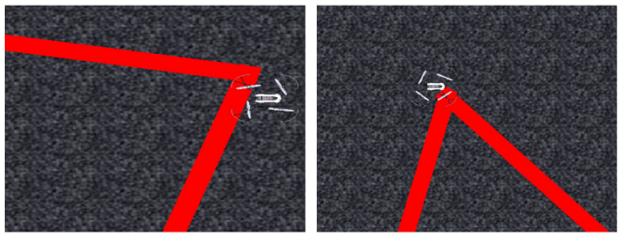
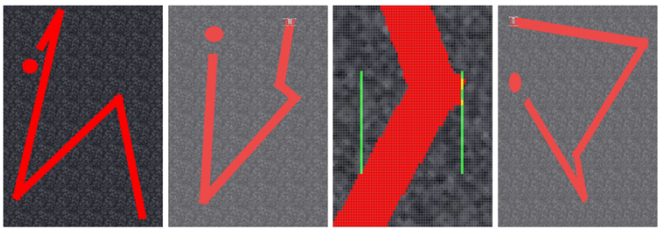
identification. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. pp. 372–384 (2020)
40. Menghi, C., Nejati, S., Gaaloul, K., Briand, L.C.: Generating automated and online test oracles for simulink models with continuous and uncertain behaviors. In: European software engineering conference and symposium on the foundations of software engineering. pp. 27–38. ACM (2019)
41. Mezhuyev, V., Al-Emran, M., Ismail, M.A., Benedicenti, L., Chandran, D.A.P.: The acceptance of search-based software engineering techniques: An empirical evaluation using the technology acceptance model. IEEE Access **7**, 101073–101085 (2019). https://doi.org/10.1109/ACCESS.2019.2917913
42. Panichella, A., Kifetew, F.M., Tonella, P.: A large scale empirical comparison of state-of-the-art search-based test case generators. Information and Software Technology **104**, 236–256 (2018)
43. Parrot SA: Parrot sa. `https://www.parrot.com/en/drones` (2025), accessed: August 8, 2025
44. Rihem Farkh, K.A.: Vision navigation based pid control for line tracking robot. Intelligent Automation & Soft Computing **35**(1), 901–911 (2023). https://doi.org/10.32604/iasc.2023.027614
45. Roy, A., Noel, M.M.: Design of a high-speed line following robot that smoothly follows tight curves. Computers & Electrical Engineering **56**, 732–747 (2016). https://doi.org/10.1016/j.compeleceng.2015.06.014
46. Sajad Khatiri: UAV Testing Competition. `https://github.com/skhatiri/UAV-Testing-Competition?tab=readme-ov-file` (2025), accessed: August 8, 2025
47. Sartaj, H., Muqeet, A., Iqbal, M.Z., Khan, M.U.: Automated system-level testing of unmanned aerial systems. Automated Software Engineering **31**(2) (Aug 2024). https://doi.org/10.1007/s10515-024-00462-9
48. Sayyad, A.S., Goseva-Popstojanova, K., Menzies, T., Ammar, H.: On parameter tuning in search based software engineering: A replicated empirical study. In: International Workshop on Replication in Empirical Software Engineering Research. p. 84–90. RESER, IEEE (2013). https://doi.org/10.1109/RESER.2013.6
49. Shrestha, S.L., Chowdhury, S.A., Csallner, C.: Replicability Study: Corpora For Understanding Simulink Models & Projects . In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 1–12. IEEE (2023). https://doi.org/10.1109/ESEM56168.2023.10304867
50. The Mathworks Inc: Minidrone competitions. `https://www.mathworks.com/academia/students/competitions/minidrones.html` (2025), accessed: August 8, 2025
51. Vierhauser, M., Meixner, K., Biffl, S.: Scenario-based field testing of drone missions. In: 2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 10–17 (2024). https://doi.org/10.1109/SEAA64295.2024.00012
52. Waga, M.: Falsification of cyber-physical systems with robustness-guided black-box checking. In: International Conference on Hybrid Systems: Computation and Control. ACM (2020)
53. Yamagata, Y., Liu, S., Akazaki, T., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. IEEE Transactions on Software Engineering **47**(12), 2823–2840 (2021). https://doi.org/10.1109/TSE.2020.2969178
54. Zhang, M., Li, X.: Drone-enabled internet-of-things relay for environmental monitoring in remote areas without public networks. IEEE Internet of Things Journal **7**(8), 7648–7662 (2020). https://doi.org/10.1109/JIOT.2020.2988249

55. Zhang, Z., Lyu, D., Arcaini, P., Ma, L., Hasuo, I., Zhao, J.: Effective hybrid system falsification using Monte Carlo Tree Search guided by QB-robustness. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification. pp. 1–24. Springer, Cham (2021)

Alternative Texts for Your Images, Please Check and Correct them if Required

Page no	Fig/Photo	Thumbnail	Alt-text Description
	11fig	$\psi := \Box(timeOutsideLine \leq 250) \wedge \Diamond(completed = 1) \wedge \Box(distance \leq 25).$	The image displays a mathematical formula using logical and temporal operators. The formula is: [psi := Box (timeOutsideLine leq 250) land Box (completed = 1) land Box (distance leq 25)] Here, (psi) is defined using the box operator (Box) indicating a condition that must always hold. The formula involves conditions on variables: (timeOutsideLine), (completed), and (distance).
3	fig1.jpg		A small quadcopter drone with a white and black body, featuring four propellers and protective guards. The drone has green lights on the front, and it appears to be in flight.
4	fig1.jpg		Flow chart illustrating a simulation process. It includes four main components: Signal Builder, Flight Control System, Simulation Model, and Simulink 3D Visualization. Arrows indicate data flow between components. Signal Builder sends "acCmd" to the Flight Control System, which outputs "actuators," "timeOutsideLine," "completed," and "distance" to the Simulation Model. The Simulation Model processes these inputs and sends "states" to Simulink 3D Visualization. Data loops back from the Simulation Model to the Flight Control System, completing the cycle.

Page no	Fig/Photo	Thumbnail	Alt-text Description
5	fig1.jpg		A 3D plot depicting a zigzagging red line with a circular endpoint, positioned above a blue square on a textured gray surface. The scene is enclosed by red structural frames, suggesting a spatial or architectural context.
6	fig1.jpg		A sketch illustrating three versions of a path on a textured background. The paths are represented by different line styles: dotted for Version 1, dashed for Version 2, and solid for Version 3. The paths converge at a red circle on the right side. A legend in the bottom left corner identifies the line styles corresponding to each version.
8	fig1.jpg		Flowchart illustrating a process with four main states: "take_off," "on_the_linee," "turning," and "landing." Each state is represented by a box with associated conditions and actions. Arrows indicate transitions between states based on conditions such as line detection and direction checks. The "take_off" state has mode 0, transitioning to "on_the_linee" with mode 1. The "turning" state has mode 2, with theta calculated as next_direction times pi divided by 180. The "landing" state has mode 5, triggered by circle detection.
9	fig1.jpg		Flow chart illustrating a three-step process. Step 1: "Input Generation" with input labeled (A) leading to (i). Step 2: "System Execution" with input (S) leading to (S(i)). Step 3: "Fitness Assessment" with inputs (T) and (arphi), resulting in (tc(i)/NFF). An

Page no	Fig/Photo	Thumbnail	Alt-text Description
			arrow loops back from Step 3 to Step 1 labeled (f(S(i))).
14	fig1.jpg	 (a) Failure for Version V1. (b) Failure for Version V2	Two-panel image showing a drone navigating around red obstacles. Panel (a) labeled "Failure for Version V1" depicts the drone colliding with a red line. Panel (b) labeled "Failure for Version V2" shows the drone encountering a similar issue with a red line. Both panels illustrate navigation failures in different versions.
16	fig1.jpg	 (a) Failure F1. (b) Failure F2. (c) Failure F3. (d) Failure F4.	Four-panel figure illustrating different failure modes in a sketch format. Each panel shows a red geometric shape on a textured gray background. Panel (a) depicts a zigzag line with a circle; panel (b) shows a bent line with a circle; panel (c) features a wide, red diagonal band with green vertical lines; panel (d) displays a triangular shape with a circle. Each panel is labeled with a failure type: F1, F2, F3, and F4, respectively.
23	fig1.jpg		Creative Commons license symbols indicating "Attribution" (BY), "Non-Commercial" (NC), and "No Derivatives" (ND). The symbols include a person, a crossed-out dollar sign, and an equals sign, respectively.