Testing Industrial Robotic Systems: A New Battlefield!

Arnaud Gotlieb, Dusica Marijan, and Helge Spieker

Abstract Industrial robotics is a field that evolves very fast, with ever-growing needs in terms of safety, performance, robustness and reliability. Nowadays, industrial robots are communicating cyber-physical systems that embed complex distributed multi-core software-systems involving intelligent motion control, anti-collision, and advanced force or torque control. The increased complexity makes these robots more fragile and more error-prone than they were previously. Failures can originate from many sources including system and software bugs, communication downtime, CPU overload, and robot wear and tear. Fortunately, advanced verification techniques such as constraint-based testing and validation intelligence are employed to cope with specification and development errors and ensure a better quality of delivered robots. In this chapter, we address the challenges of testing industrial serial robots and provide examples of Artificial Intelligence and Constraint Programming techniques being used to ease the automation of some parts of the robot testing processes. In particular, we present techniques for test generation, planning and execution for industrial robots, as well as the deployment of this technology into the real-world continuous integration process of a large robot manufacturing company. The presented techniques are complementary to other strong formal verification techniques such as model checking and theorem proving, and only the combination of these techniques will lead to an industrial manufacturing world where robots are safer and more reliable.

Arnaud Gotlieb
Simula Research Laboratory, P.O.Box 134, Lysaker, Norway, e-mail: arnaud@simula.no

Dusica Marijan
Simula Research Laboratory, P.O.Box 134, Lysaker, Norway e-mail: dusica@simula.no

Helge Spieker
Simula Research Laboratory, P.O.Box 134, Lysaker, Norway e-mail: helge@simula.no

1 Introduction

Robots are complex cyber-physical systems that are used nowadays in a range of safety-critical domains. In manufacturing, robots can perform tasks in full autonomy, for example on-demand painting with color and brush change [58], or they can collaborate with humans on the factory floor [57]. Robots are furthermore increasingly used in healthcare and medical operations [36, 3], education [42], and transportation [10]. Consequently, robots increasingly affect public safety [16], raising the interest of the community in their quality and safety assurance.

Due to the intrinsic complexity of robotic systems on the one hand [4], and the need of their error-proof interaction with the physical environment on the other [45], it is crucial to perform rigorous testing of these systems before their deployment into the field. This chapter focuses on industrial robots, which are robots used in factories to help improve worker productivity and safety by replacing or co-operating with humans. Examples of such robots are shown in Figure 1. On the left, a collaborative dual-arm Yumi robot is shown where the robot automatically sorts objects. On the right, a UR3 robot is shown from our laboratory setup for testing of learning robots.

Fig. 1 Examples of collaborative industrial robots from ABB and Universal Robots (Picture credits: Arnaud Gotlieb, Mohit Kumar Ahuja).

Industrial robotics is a field that advances quickly. Over the last few decades, industrial robots (IR) have gone from simple systems able to do easy pick-and-place tasks without any sensing capability to present-day automated and autonomous robots powered by Artificial Intelligence (AI), fully collaborative and able to dynamically adapt to their environment. While increasing manufacturing productivity by orders of magnitude, the underlying complex technology of modern IR introduces a vast set of challenges on IR robustness and safety. Comprehensive automated testing of IR is necessary to ensure both the early detection of potential robot faults before system deployment into operation, and to keep maintenance costs low, once the system is in operation [5]. However, comprehensive testing of IR is challenging, owing to multiple reasons, discussed in more detail in the next section.

IR increasingly support high levels of human-robot interaction (HRI) [56] and high levels of robot autonomy, which in turn affects the nature of HRI [8]. Indeed,

this context requires human imitation and demonstration capabilities [2, 84], rapid adaptation of the robots to changing environments, as well as safer robot behaviours [5]. Examples of industrial collaborative robots include single-arm robots that can work close to human workers (e.g., Universal Robot's UR3), and single-arm robots (e.g., CEA's SYBOT) or small-parts assembly robots (e.g. dual-arms ABB's YUMI) that cooperate with human co-workers.

To cooperate safely with humans, these robots have a special design where sensors and actuators are used to detect any human pressure or prevent dangerous robot actions. They are also equipped with perception systems that can detect obstacles and prevent collisions well in advance. However, the intelligent control systems that equip these robots have become more and more complex and they need to be frequently updated for maintenance. Hence, continuously controlling the safety and security of these systems is crucial [61, 5, 78].

Moreover, by using learning abilities, robots can optimize their trajectories to increase their speed and overall performance. Their actual behaviour is learnt instead of being carefully specified. Hence, controlling and testing the robot control systems has become very challenging as the precise expected behaviours of these collaborative robots are not known in advance.

IR can learn a range of skills, such as optimized trajectories, locomotion, grasping, obstacle detection and collision avoidance, or interactive abilities such as optimized object picking with human co-workers. Learning methods in IR use different theoretical frameworks of machine learning (ML), including imitation learning [12, 1, 52] where a robot-learner is trained to mimic human behaviours from demonstrations, reinforcement learning [69, 44] where a learner optimizes its actions by obtaining rewards or penalties from the environment, inductive programming [27] where a programming language allows learners to generalize rules from examples or instances, constraint acquisition [11] where constraint models are learnt by generalizing from given solutions and non-solutions, and deep learning (DL) [14] where multi-layers neural networks are trained over large corpus of data to efficiently recognize various type of signals. Learning skills can be given either through self-exploration of the space of trajectories or through the guidance of a human teacher.

Despite the considerable development of learning methods in the two last decades, testing IR faces specific challenges such as the limited availability of advanced robots to perform experiences, the unbearable cost of failures in robot deployment, the lack of systematic studies on how to evaluate robot learning, and the absence of appropriate test platforms. Furthermore, there is still no unifying learning framework for formalising ML in robotics. This unifying framework could be helpful to understand how different learning methods can be combined to deal with difficult tasks.

This chapter discusses the prominent challenges of testing IR, including the robot autonomy and collaborative capabilities. In Section 2, we provide an overview of recent advances in this area, with a focus on test generation and validation, and test planning under resource constraints. These topics are further discussed in Sections 3, 4 and 5 with a more in-depth presentation of selected methods and techniques in these areas. The chapter closes with a summary of the current state of testing IR and open questions for future work in Section 6.

2 Testing Industrial Robots: Challenges and Recent Advances

Robots are getting smarter, in terms of reasoning, sensing and adapting. They integrate a range of capabilities supported by AI and ML, to enable self-learning, advanced perception of the environment, synchronisation with humans and other robots, and motion control, to name a few. These capabilities make IR able to perform complex tasks *autonomously*, however, robot autonomy induces issues regarding robot dependability [37] and trust [6]. Consequently, these advanced capabilities also lead to increased complexity of testing IR. As Guiochet et al. [37] argue, one of the core challenges for deploying robots on a large scale will be ensuring their dependability and safety.

IR are increasingly *collaborative*, intended to interact with humans in a shared workspace. In contrast to conventional IR, which can work autonomously but are caged or placed at a safe distance from a human, collaborative IR perform joint object manipulation, responding to the human-worker motion in real-time. As such, these robots introduce the need to satisfy an increased level of safety requirements [45, 86], to ensure that human workers are not at risk when working collaboratively alongside robots.

A recent study [4] identified nine key challenges for automated testing of robots in practice. These challenges include: 1) unpredictable corner cases, 2) engineering complexity, 3) culture of testing, 4) coordination, collaboration, and documentation, 5) cost and resources, 6) environmental complexity, 7) lack of oracle, 8) software and hardware integration, and 9) distrust of simulation. The data for the study was collected from the interviews with 12 robotics practitioners from 11 robotics companies. While this study provides a comprehensive view of the practical and organizational challenges for testing robots, it does not consider several important technical challenges, which we discuss next.

Specifically, we discuss the challenges of automated testing of IR belonging to two broad categories: 1) test-input and output generation, and 2) test planning under resource constraints.

Furthermore, we highlight the importance of continuous testing of IR, where robot code components are tested incrementally, as they are developed. Continuous testing for IR aims to make the process of developing IR faster and less expensive by detecting potential errors at early stages of development, and thus avoiding expensive integration faults.

2.1 Test Generation

Test generation for IR is concerned with two basic tasks: test-input generation, and test-output generation. Next, we discuss techniques for test-input and output generation.

2.1.1 Test-Input Generation

When testing IR, the goal is to specify realistic, diverse, and complete test inputs, because IR operate in complex, highly configurable, and non-deterministic environments. Conventional approaches to this end include combinatorial sampling [54], which can generate a test-input set that is complete for a certain n-wise coverage. N-wise coverage means that all combinations of values of n parameters are included. Another approach is model-based test input generation [5], where authors first generate abstract test sequences, which are concretized in a simulator execution.

The test-input space for typical IR is vast, as present-day robots integrate a range of sensors, dealing with a large amount of data. This consequently creates a significant challenge of selecting an adequate set of inputs. Answering the question "when to stop testing" leads to the definition of specific coverage criteria for the Machine Learning (ML) model of an IR, which can guide the selection of inputs able to reveal false classification or incorrect regression during testing. This problem has been encountered in the software testing domain, especially in the area of fuzzing [53]. Fizzing is a testing technique where randomized inputs are provided to software under test with the goal to reveal vulnerabilities.

To address the challenge of test-input selection, several *test adequacy criteria* have been proposed. Source code coverage is one such metric, which makes it possible to decide when enough test cases have been collected and the test adequacy criterion is sufficiently fulfilled. Inspired by traditional code coverage metrics such as statement or decision coverage, neuron coverage [64] counts the number of activated neurons in the neural network when test inputs are submitted for classification. It is believed that a higher neuron activation coverage leads to a higher chance of detecting wrong classifications. Other studies have confirmed this initial result and also extended the proposed structural criteria by distinguishing neuron-level criteria from layer-level coverage criteria [50].

Instead of looking at the network structure, a different criterion called surprise adequacy has been introduced for measuring the diversity in training data [43]. The surprise of an input is defined as the difference between the input and the training dataset with respect to the behaviour of the ML model. It is then suggested that one should select inputs that are sufficiently surprising but still within the expected data distribution to compose an adequate test set. These criteria are specific to ML and can be used to guide the selection of test inputs, but additional techniques are needed to evaluate the correctness of results when test inputs are submitted to ML models.

2.1.2 Test-Output Generation

Expected test outputs are used in testing for checking the correctness of the actual outputs produced in a system execution. A mechanism for comparing the actual outputs against the expected outputs is known as a *test oracle*. The approaches to automated generation of test oracles can be broadly classified as *specified* and *derived* test oracles. The former exist if there is a specification (formal, semi-formal,

or informal) of expected outputs. If not, test oracles are derived from various artefacts such as documentation, program executions, known properties of the system under test, or invariants inferred from system executions.

Both approaches face challenges when dealing with sophisticated systems such as IR. For IR integrating AI and ML, generating test oracles has become intrinsically complex. This is because ML introduces probabilistic reasoning that may give rise to nondeterministic system behaviour. This means, in simple terms, that a robot might produce different output given a certain input. Because IR learn to predict outputs from training data, the correctness of the output in testing IR cannot be easily determined. Therefore, it becomes crucial to *ensure the quality of training datasets*.

The quality of the training dataset largely affects the performance of the ML model (IR system). This is even more the case with deep learning (DL) models, which have become a popular technology used in IR [65]. While DL is advancing robot capabilities in general, there is a strong dependence of DL on the training data, which makes it vulnerable to adversarial attacks [31], where small modifications to input data can cause misclassifications. As Sunderhauf [80] pointed out, the application of DL in robotics introduces new challenges that are specific for robotic vision and distinct from those present in computer vision. For example, computer vision takes images and translates them into information, while robotic vision translates images into actions. Consequently, these new challenges need to be adequately addressed in testing.

The behaviour of DL models follows the examples given in the training data. When collecting a training dataset for a new application, the data must be diverse enough to cover all variances that are expected to be encountered by the deployed model, i.e. the data distribution for which and on which the model is trained.

There are multiple potential failures here. First, the collected training data might be insufficient to train a model that can generalize to the real-world data. In this case, the training data should be expanded and the test set needs to be enriched by examples that the model does not generalize to. This will make it possible to identify these issues before deployment of a new or improved model.

Second, the training data might lead to a capable model for the real-world data, but over time the data encountered by the deployed model gradually changes and no longer fits the initial training distribution. This reduces the model performance and increases false predictions. An approach to address this challenge is to monitor the actual data encountered by the deployed model and measure how it differs from the training data. This monitoring is then followed by frequent model fine-tuning or retraining with an adjusted data set.

Other failures are caused by not carefully selecting the data and thereby introducing, for example, redundant data, adversarial data, or biases into the dataset. In case of biases, these can lead to a model that does not make fair predictions but is conditioned to repeat the biases encoded in the training data.

There are approaches inspired by *mutation testing* [41] proposed for evaluating the quality of test datasets for Deep Neural Network (DNN). DeepMutation [51] is one of the first works in this area. It specifies a set of mutation operators to inject

faults into training data. The idea is to, then, retrain ML models using the mutated training data, which will produce mutated models. In this way, faults are injected in the models, after which mutated models are tested using a test dataset. The quality of the test dataset is evaluated based on how many injected faults are detected. The drawback of DeepMutation is related to using basic mutation operators, which may seed faults that are not very representative of real faults [40]. Shen et al. [72] proposed MuNN, as another approach for mutation testing of neural networks. In this work, it was shown that neural networks of different depth require different mutation operators. An extended discussion of mutation testing is given in Chapter 11, where its application towards the testing of robotic systems is demonstrated.

2.2 Test planning under resource constraints

When testing IR, it is common that the resources available for testing are constrained. The physical test agent, on which the test cases are executed, are limited since they require space, maintenance efforts as well as initial acquisition costs. Therefore it is often economically infeasible to remove this bottleneck in IR testing, especially when having to consider a variety of IRs with different feature sets. The availability of these test agents is further restricted in practice, albeit due to defects, maintenance windows, or usage from other projects. Furthermore, in some cases, test agents need exclusive access to shared additional resources, such as measurement devices, conveyor belts, or network equipment. Finally, the time windows for testing are limited as well and testing should often be completed at a fixed deadline or with minimal total execution time. All these constraints increase the complexity of the test organization and test planning becomes necessary for efficient resource usage. While manual test planning is often state-of-the-practice, it is often also non-optimal and is less flexible as it has difficulty in adapting to varying scenarios where only subsets of test agents are available or the number of tests changes.

Following this description, test planning under resource constraints covers several smaller problems: deciding what to test, when to test and where to test. Within the software testing community, these problems are referred to as test case prioritization, test case reduction (also referred to as minimization or selection), and test case scheduling.

This assumes that the execution environment is generic or decoupled from the software, and the test suite can be executed on any test environment. For testing robotic systems it is further relevant to include the test environment into the test planning process as its resources need to be managed and considered, too. In 2012, Yoo and Harman published a major review on test selection and prioritization in software testing, and we refer the interested reader to this survey for an in-depth overview of the earlier literature [85], while we will discuss the specific challenges and developments in the industrial robotics context.

When creating a test plan, there are two distinct optimization goals. Either the test suite is fixed and the plan needs to minimize resource usage, e.g. total execution

time or the resource constraints are fixed and the test plan has to make the best use of these constraints, which can require to only select a subset of the test suite.

It is helpful in both cases to have a notion of relevance for each test case in the test suite to decide if it should be executed and how soon. The identification of this notion of relevance is called *test case prioritization* [68, 25], which has the goal to express an order of test case for high effectiveness, such that failing test cases are executed before passing test cases. The prioritization step can take a variety of information sources into account to prioritize a test case and establish a notion of how likely the test case might fail. These sources include source code coverage [25] and changes [30, 74] or historical test metadata [78, 46]. Shin et al. present a multi-objective test case prioritization approach for the acceptance testing of cyber-physical systems that includes both uncertainties in the test environment and potential hardware damages in the prioritization [75].

Once a priority has been assigned, a *test suite reduction* method selects the most relevant test cases for actual execution and thereby reduces the size of the test suite [83, 34, 73]. Reduction thereby performs the step to adjust the size of the initial test suite to the available, limited resources. For the case of sequential execution, the selection method can choose the most prioritized test cases that exhaust the available resources. Test suite reduction is also referred to as test suite minimization [39] as it minimizes the number of test cases from the full initial test suite to the test suite that is actually executed.

More complex approaches to test case reduction can consider additional constraints and requirements on the resulting test suite. For instance, they can demand certain coverage criteria to be fulfilled to cover the whole system even though the risk of failure in some subsystems has lower priority than in other subsystems. Additionally, it can be necessary to consider more resource constraints than just the available time, such as compatibility to available test agents or dependencies on external devices.

Finally, *test case scheduling* is the assignment of a test case to both a test agent and a time slot and is an application of constraint-based machine and project scheduling [13, 7, 38]. It differs from test case reduction, which selects a subset, by also making the additional assignment of time and execution location.

The scheduling model captures the constraints of the individual assignment between test cases and test agents as well as potential constraints on the execution orders of test cases. While test cases should generally be independent of each other, it can be necessary to group certain kinds of test cases to avoid costly setup times, or to avoid certain groupings of test cases because they rely on similar external resources. These constraints need to be developed together with domain experts, e.g. the quality engineers developing the tests, and need to be formalized and stored as metadata for the test cases.

3 Test Generation

This section presents two research leads that we pursue in the field of test generation for testing IR. The former subsection details the generation of test trajectories for IR and the latter presents our ongoing work to exploit Metamorphic Testing for testing robot's behaviours.

3.1 Stress test-trajectories generation with Constraint Programming

Serial IR embed complex multi-core software-systems with advanced motion control, collision avoidance, and intelligent torque control. As a result of the ever-increasing complexity of these systems and their interactions, IR have become more error-prone. Hence, generating tests that stress these robots to steer them to their limits is crucial to ensure quality and robustness. A goal in this area is to generate test trajectories of the end-effector of the robots that have the greatest potential to show deviations between the trajectory commanded to the robot and its actual trajectory. As an example, Figure 2 illustrates such a deviation for a 3D-trajectory specified to the robot. Generating test trajectories which exhibit such deviations is challenging

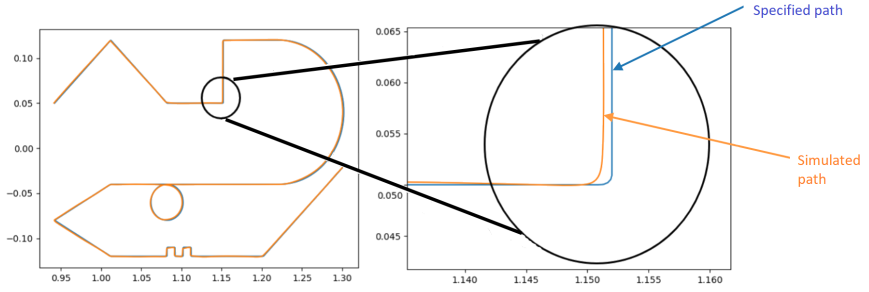


Fig. 2 2D-Projections of the robot specified test 3D-trajectory (in orange) and its executed trajectory in simulation (in blue). Both are very close of each other (on the left) but, by zooming in the figure, one uncovers a deviation between the specified and simulated path (on the right). Those deviations reveal potential software or hardware failures which shall be discovered before deploying the robot in operational contexts.

as there is no model of deviation and the space of possible trajectories is unbounded.

To the best of our knowledge, there is no fully-accurate analytic model that can solve the problem. The modelling of the robot's (inverse) kinematics involves solving a large number of multivariate polynomial equations. Typically, a 4-DoF[1] serial robot already requires to solve around 49 4^{th}-degree polynomial equations

[1] Degree of Freedom: typical industrial robots have 6-DoF

with 49 variables [55]. That is why, even though careful design of the robot using simulators is possible, the automatic generation of stress trajectories for finding and testing deviations between specified and simulated paths is crucial, to ensure high-quality of these industrial robots.

A possible approach for addressing this problem is to use *Constraint Programming*, which is a general-purpose framework [67] for solving combinatorial problems. Basically, these problems, called *Constraint Satisfaction Problems (CSPs)* are defined by a set of variables V, a domain D and, a set of constraints C, which are relations over V. Each variable v_i takes its possible values in a finite (or continuous) domain D_i, and D is the Cartesian product of all D_i. The goal of solving CSPs is just to find an assignment of all the variables to a single value of their domain such that all the constraints in C are satisfied. Sometimes, a cost-objective function can be added and when there are multiple solutions, the goal becomes to find one solution which optimizes that function. CP has been successfully used to solve many test generation problems [32], including applications in the IR domain [59, 60, 61].

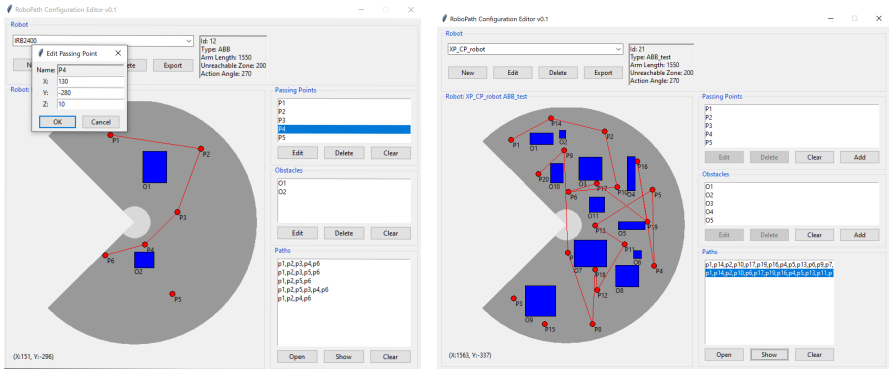
The workspace of an IR is usually materialized by a subspace of a 3D-space, a set of waypoints to be reached (in order to perform some dedicated tasks with the end-effector), and a set of obstacles. A valid trajectory in the workspace is a path that meets all or some of these waypoints at user-selected speeds, without hurting any of the obstacles. Generating trajectories to stress the robot involves finding valid trajectories that maximize the load of the robot, by changing directions and speed. The number of waypoints, obstacles and the complexity of the load-function to be maximized makes the problem hard to solve, as there are only a few valid trajectories in a huge search space of trajectories. In some cases, some way-points cannot been reached. Hence, the problem is actually to find trajectories which hit some of the waypoints (not necessarily all), while maximizing the load.

In [49, 21], we proposed a method and a tool called Robtest to generate these trajectories by using CP. Robtest exploits a continuous-domain constraint solver called RealPaver [35] and the constraint language MiniZinc [63] with three backends solvers, namely, `gecode` [70], `chuffed` [20] and `SICStus` [17], for solving constraints over finite domains.

In CP, constraints with a non-fixed number of variables, called *global constraints*, are very successful in tackling difficult problems, as their solving is based on powerful domain-filtering techniques. Our model in Robtest exploits several global constraints such as Inverse, Subcircuit and Table to construct an effective approach, which generates stressful test trajectories.

Figure 3 shows the Robtest user interface where two cost-maximal trajectories have been automatically generated. Interestingly, these trajectories are automatically converted into scripts in a dedicated command language for the robot called Rapid. Robtest has been deployed and validated on real industrial robots and experimented with virtual workspaces containing more than 80 waypoints and 60 obstacles. For these workspaces, Robtest generated maximal trajectories in less than 5 minutes[2].

[2] The experimental benchmark and the CP model are publicly available at `www.github.com/Makouno44/Robtest`.



(a) With 2 Obstacles (b) With 10 Obstacles

Fig. 3 Examples of Robtest interface, which shows the 2D-projection of the robot workspace (in grey), waypoints and trajectories (in red) and obstacles (in blue). These optimal trajectories were generated by Robtest for 3D-workspaces with two obstacles (a) and ten obstacles (b). Some waypoints are not reached but the generated trajectories are maximal w.r.t. the load-function to optimize (Pictures by: Mathieu Collet).

To the best of our knowledge, there is no other model able to deal with so many waypoints and obstacles.

The next subsection introduces the notion of Metamorphic Testing and how it can be used to also generate new test cases for IR.

3.2 Metamorphic Testing of Robots

Metamorphic testing is a testing paradigm, in which a source test case is transformed into a new follow-up test case for which the exact expected outcome is unknown, but a relation between the source and follow-up test case is available [71, 19]. By execution of the follow-up test case it can be confirmed whether the system under test behaves according to the so-called metamorphic relation. If the relation is violated, a failure in the system has been identified. Metamorphic testing thereby addresses the oracle problem in software testing, where it is impossible or difficult to know the exact system output for a test case.

Examples of successful applications of metamorphic testing are search engines, where it is generally expected that additional keywords reduce the number of search results, or learning systems, where the exact reaction is unknown, but similar inputs should reveal similar results, given an adequate definition of similarity. The oracle problem [4] also applies to robotic systems and some of their components when it is difficult to precisely specify the final state of the system or the effect every individual action has.

Recent works have applied metamorphic testing to robotics-related computer vision systems, such as image classification [23, 24] and object detection [77]. By using metamorphic relations, the test images are modified and can be validated against the output of the original input images. This makes it possible to test the variability of the computer vision system without additional labeling efforts. While it has received increasing attention in the context of testing ML systems, metamorphic testing has not yet been widely applied to industrial robots in general.

An initial study by Lindvall et al. [48] showed success in metamorphic testing of autonomous drones in combination with model-based testing. Their test framework generates test cases from a model based on the drone specifications, for example, a full flight scenario with obstacle avoidance. The applied metamorphic relations span five equivalences of the drone system: 1) the behaviour is expected to stay constant over multiple runs, days, and reboots; 2) the behaviour is rotation-invariant, i.e. rotating the world does not affect the outcome; 3) the behaviour is translation-invariant, i.e. against the movement to a different location in the environment; 4) the exact location of the obstacle does not affect the result; and 5) the exact formation of the obstacles does not affect the result. The five equivalences can be combined and thereby create a wide variety of follow-up test cases from a single source test case, especially when further combined with model-based testing to even generate the source test cases.

To evaluate whether the behaviour matches the metamorphic relations, the authors outline different criteria, such as discretization of the sensor data to receive comparable values or the shape of the path the drone took, but the choice of this evaluation criterion is an open research question. Within their evaluation, they focus on the manual inspection of different flight paths for the generated scenarios, and the generated scenarios identified misaligned drone behaviour, leading to longer flight paths or irresponsible landing behaviour in some cases.

One potential outline for the application of metamorphic testing in robotics is related to the usage of domain randomization [81, 82] in the training of ML-based robotics. By introducing small randomizations in the training task, i.e. changing object colors, lightning conditions, or the physical model of the environment, the robot learns more robust behaviour. A similar approach can serve as a basis for metamorphic testing. The initial scenario is randomly modified in a similar way and expectations about the desired outcome are maintained. Maintaining the same expectations on the output, even though the test input has changed is the identity function, i.e. the result stays the same independent of the modification. However, metamorphic relations make it possible to define more complex relations over the inputs and outputs of changed test cases and can formulate acceptable or mandatory changes in the outcome.

In conclusion, metamorphic testing is a flexible testing technique and has already shown to be applicable in some cases. We expect that the interest and attention on the combination of robotics and metamorphic testing will increase in future work.

4 Test Planning

This section discusses selected methods for test planning under resource constraints. In the first part (Section 4.1), we will present a technique for optimal test suite reduction, i.e. the selection of a smaller test suite that maintains a set of coverage properties. In the second part (Section 4.2), we then proceed to the scheduling and assignment of a test suite to a number of test agents.

4.1 Test Suite Reduction

Testing ABB's industrial robots is performed as part of a continuous integration (CI) process [60]. In CI, the goal is to build, test and deploy software in frequent iterations, and thus avoid software integration faults by detecting software errors at an early stage of development. Each development iteration, also called a CI cycle, deals with developing and testing only a part of the overall robot system source code.

Consequently, in each CI cycle, only relevant test cases are selected from a larger set of test cases that cover the overall functionality of the robot under test. Since the duration of a CI cycle is time-constrained, the relevant set of test cases needs to also be minimal, so that minimal time is needed for its execution. Selecting the smallest subset of test cases from a larger set, while preserving certain properties of the test suite (e.g. requirements coverage) is challenging, and some commonly proposed approaches address this problem only with approximated solutions [66, 47]. Finding such a smallest set of test cases (a.k.a optimal test suite) is NP-complete [22, 29], and the time required to find the optimal test set grows exponentially with the size of the problem, in the worst case.

In their previous work, the authors of this chapter have studied the problem of optimal test suite reduction using network maximum flows [33] and Constraint Programming [67]. As an example, we consider a simple test suite reduction problem consisting of a set of three test cases covering a set of seven test requirements, shown in Figure 4. First, we encode the test suite reduction problem using a bipartite graph augmented by a source node S and a destination node D, shown in Figure 4 with special capacity constraints. S and D are special vertices of a flow network, the source and the sink. Capacity is associated with each arc in a flow network, as the maximum limit of flow that the arc can receive. The bipartite graph is directed from a node S to a node D, with l/c values on each arc denoting flow values and capacities respectively. Flow values of zero on all arcs mean that the flow is feasible but not maximum. To find maximum flows, the bipartite graph is first traversed using the Ford-Fulkerson method [28]. For the example test suite reduction problem of three test cases and seven test requirements, there are eight maximum flows, shown in Figure 5. The maximum flows correspond to different solutions of the test suite reduction problem. However, not all maximum flows are optimal, as they do not reduce the original test suite. To find the optimal flows (there can be more than one), CP is used to search for the flow that maximized the number of zeros on arcs from

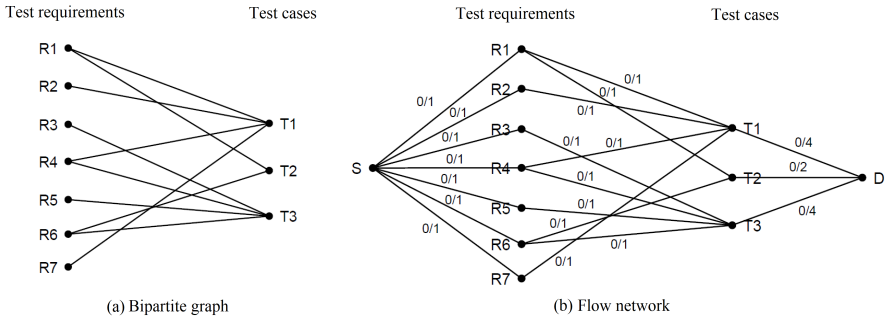


Fig. 4 Bipartite graph and the corresponding flow network for test suite reduction. The flow network consists of three test cases (T) covering seven test requirements (R). Flow values and capacities are denoted as l/c on each arc.

nodes T to D nodes [33]. As the final solution of our example test suite reduction problem, there are two optimal flows found.

4.2 Test Scheduling and Distribution

In this section, we discuss two test scheduling and distribution problems. Both problems have in common that they require to assign test cases to test agents under consideration of additional constraints. The first problem are scheduling problems where test cases should be assigned to test agents while minimizing the total execution time of the final test plan. The second problem is to assign the test cases to the test agents such that in subsequent test cycles every test case is executed on a different test agent. Our discussion includes both a description of the problem as well as the presentation of solutions for these problems from the earlier work by the authors of this chapter.

4.2.1 Test Scheduling with Global Resources

We consider a scheduling problem where test cases should be assigned to test agents while minimizing the total execution time of the final test plan. This scheduling problem is a variant of the generic machine scheduling problem [15] and covers a wide range of practical applications. Our discussion will be based on the work by Mossige et al. [62] and will contain the additional notion of exclusive access to shared global resources.

Some test cases may only be executable by a subset of the agents, e.g. a test agent might not provide a certain feature set, and some test cases can require exclusive access to a shared global resource. The final schedule will have to consider that only one test case at a time can make use of this global resource. It is furthermore

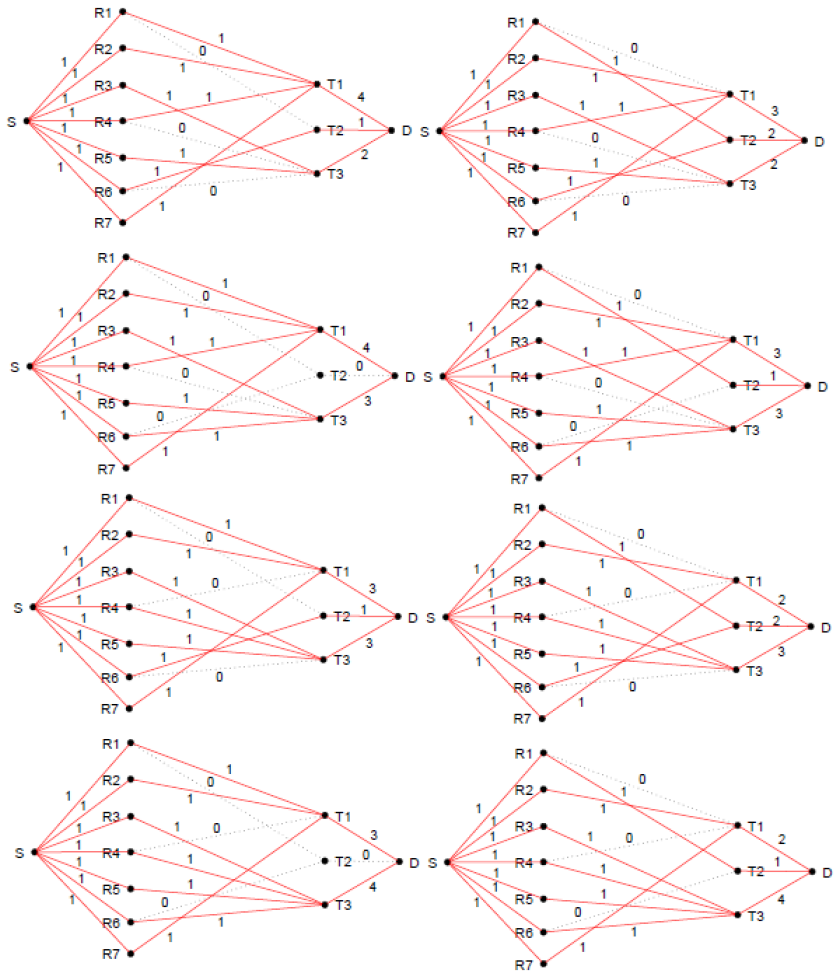


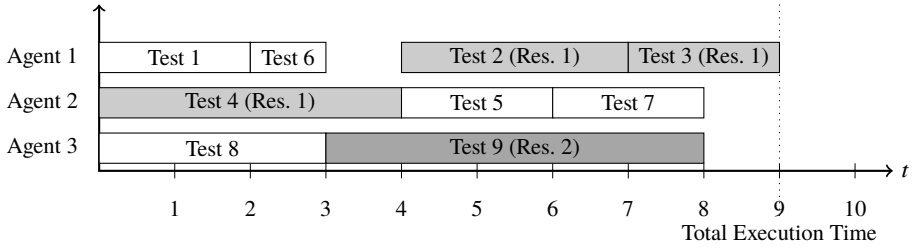
Fig. 5 Feasible solutions of the test suite reduction problem as maximum flows (from [33])

necessary for the scheduling process to know the execution time of each test case. Because there can be some variation in the actual execution time, a practical approach is to maintain the execution time of earlier runs and provide an overestimation of the average execution time as a conservative estimate.

An example problem is shown in Figure 6. A test suite of nine test cases has to be scheduled among three test agents (Figure 6a). Most test cases are compatible with all agents, except test cases 6–9, which require some specific test agent functionality. Test cases 2–4 further require one specific global resource and test case 9 depends on the other global resource. The resulting optimal schedule (Figure 6b) avoids the

Test Number		1	2	3	4	5	6	7	8	9
Duration		2	3	2	4	2	1	2	3	5
Executable on Test Agent	1	X	X	X	X	X	X			X
	2	X	X	X	X	X		X		
	3	X	X	X	X	X			X	X
Use of Global Resource	1		X	X	X					
	2									X

(a) Test Suite Overview



(b) Optimal Schedule

Fig. 6 Test Scheduling with Global Resources: An example problem with 9 test cases, 3 test agents, and 2 global resources (adapted from [62]).

overlap of test cases 2–4, which leads to the critical path of the total test plan and the final test execution time.

The model proposed by Mossige et al. [62] uses constraint-based scheduling [7] to model and solve the test scheduling problem in Constraint Programming. The cumulatives global constraint [9] supports the definition of the basic machine scheduling problem such that resource constraints on each machine, i.e. sequential execution of only a single task per time and non-preemptive execution without interruption, are followed.

By using an effective selection of global constraints, the CP solver can apply optimized filtering techniques that reduce the search space for the branch-and-bound tree search and thereby also the time until the first, and later the optimal, solution is found. Additionally, the search strategy for the tree search process affects the performance of the solver.

The search procedure attempts to assign values to variables such that all the constraints are satisfied and the cost function is minimized. Many different strategies can be used to explore the search space, however, it is known that the most effective approach is a search strategy depending on the characteristics of the scheduling problem to be solved [76].

In the case of test scheduling that is to first assign the most demanding tasks to a test agent, i.e. here those that require the most global resources and then take the longest time. The assignment is made by assigning the earliest possible start time. This procedure is repeated for all test cases while trying to first distribute the test

cases among the available machines. By initially preferring a different machine from the one that the previous test case was assigned to, we accept a compromise between the solving time of the schedule and the execution time of the schedule for the first found solution. Afterwards, branch-and-bound search minimizes the total execution time, but under resource constraints and with only short available time windows, it is often infeasible to solve every schedule to the global optimum rather than accepting a good enough, near-optimal solution.

The outlined approach is versatile and the general process persists for other variations of the scheduling. The constraint model and search strategy can be adapted to environment-specific conditions that require further constraints or requirements on the final schedule. However, the scheduling model is focused on a single test plan and cannot consider constraints and expectations that affect subsequent test schedules, e.g. from day to day. This issue will be discussed in the next section.

4.2.2 Test Scheduling with Rotational Diversity

Due to resource constraints in testing, trade-offs often have to be made between test coverage or completeness and resource usage. For example, each test is only executed once even though different test agents with different feature sets are available and for a thorough and complete test result it would be desirable to run each test on every compatible test agent.

One way to mitigate this trade-off is to follow the restrictions for a single test schedule, but address the distribution of test cases to different test agents over multiple test cycles, e.g. when running a form of continuous integration where the test process is repeated frequently. In every test cycle, the test schedule is automatically planned anew and the optimization function is adjusted so that it prefers the assignment of test cases to test agents that have not been recently made. This is especially relevant when the availability of test cases and test agents can change due to maintenance, new test cases, or restricted access to certain agents.

A solution to this problem of *rotational diversity* has been proposed in our earlier work [79]. As shown in Figure 7, the main idea is to split the overall problem into two sub-problems. The inner problem is the actual test scheduling or test selection problem to be solved at every cycle to produce the test plan according to an optimization objective. The outer problem then adjusts the optimization weights of each test case in the schedule so that it incorporates information about the time since it has been assigned to each test agent, the so-called affinity. The higher the affinity, the better the solution if the assignment between these test case and test agent are made.

The decomposed solution approach allows adopting the method for other problems or problems with a dedicated set of additional constraints, as long as the notion of an optimization objective and individual optimization weights per test case and test agent are given.

Several strategies for combining the original optimization weight with the affinity value have been proposed and compared [79]. Experimental results show that, in

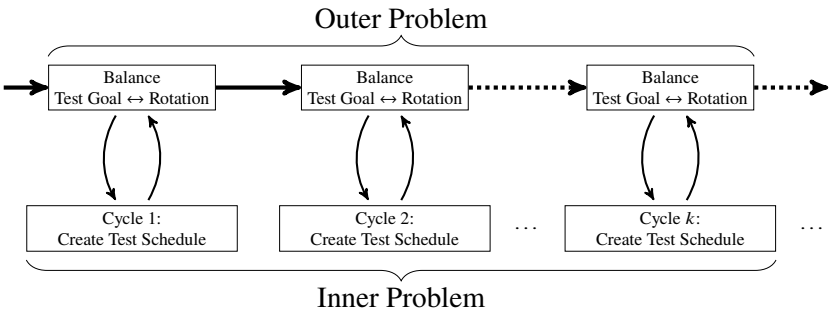


Fig. 7 Test Scheduling with Rotational Diversity: The problem is divided into the actual scheduling problem per cycle and a balancing problem to steer the scheduling goal towards a trade-off between rotation and effectiveness (adapted from [79]).

a scenario where a maximum time limit for testing should be exhausted, a product of both factors is a simple, but effective strategy. Such a strategy makes it possible to produce test plans that frequently rotate all test cases over their compatible test agents while only reducing the original optimization objective by on average less than 4%.

In this section, we discussed methods to distribute test cases onto physical test agents with the goal to make best use of limited resources. The next section will look into the actual execution of these test cases on the physical hardware through the example of two case studies at our industrial partner ABB Robotics Norway.

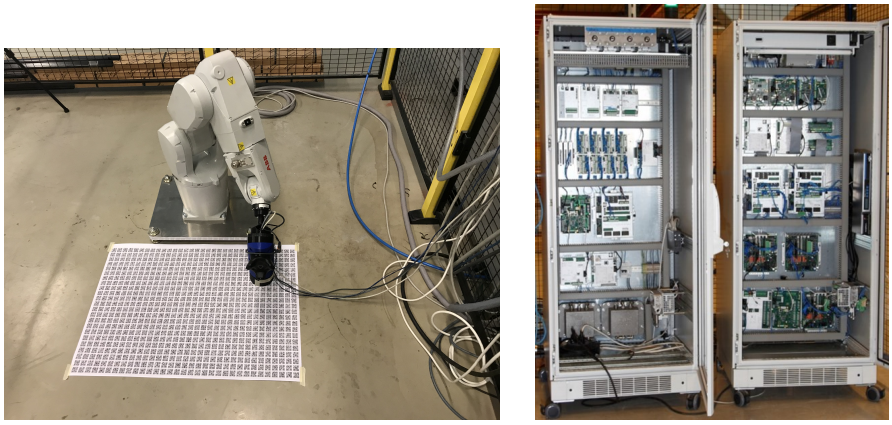
5 Test Execution

This section discusses test execution environment for robots at ABB (Section 5.1), in presents a method for executing test cases for an integrated robot painting system (Section 5.2).

5.1 Automated Regression Testing for Industrial Robots

Test execution for IR is always challenging as it involves not only to perform an appropriate combination of testing in simulated environments and testing of real robots in safe environments but also to automate the testing process in continuous testing and delivery. Simulated environments for IR setup and testing include for instance ABB's Robotstudio or Gazebo[3], but the design and automation of input scenarios for these environments requires skilled robotics engineers who are familiar with both the mechanics and electronic of the robots. Besides, these environments

[3] http://gazebosim.org/



(a) Robot Testing (b) Robot Testbench

Fig. 8 Actual robot testing versus testbenchs for testing IR software systems (Pictures by: Morten Mossige).

have usually not been created for repeated test execution and they include neither test-scripting facilities nor test-results checking features.

Automating test execution for real robots is challenging as it involves robot motions (e.g., arm motions) that must be accurately synchronized with other subsystems, to perform specific tasks, such as painting or glueing. It is a usually labor-intensive step to prepare the robot environment for executing the tests, as strict safety regulations are enforced when humans are setting up the environment that involves moving machinery and sometimes dangerous paint fluids and gases [26]. Due to high cost, this type of test execution is usually performed during the final verification step and aims to detect subsystems synchronization problems or performance issues. Test-execution setups based on real robots have the advantage of providing very accurate results, but they require long labor-intensive preparation efforts.

Among the faults that are sought, software defects (as opposed to hardware defect) are usually prominent and finding them is crucial before delivering the robots to their end-users. Fortunately, correcting software defects usually does not require long and costly procedures. Thus, there is a trend that consists in testing software systems independently from the physical robots, by using *testbenches*, such as the one shown in Figure 8. These testbenches include layers of motherboards with actual subsystem CPUs, which enable extensive automated *regression tests execution*.

Regression tests are test scripts that are systematically executed for testing a new software release of an already deployed system. With these tests, if any of the test verdicts is failed, then the software release has detected a regression fault, meaning that some kind of added feature has broken previous correct behaviours of the system. The obvious advantage of using these testbenches for regression testing is that they test actual software components to be deployed on the robot (instead of simulated components), which allows test engineers to fully automate the test-

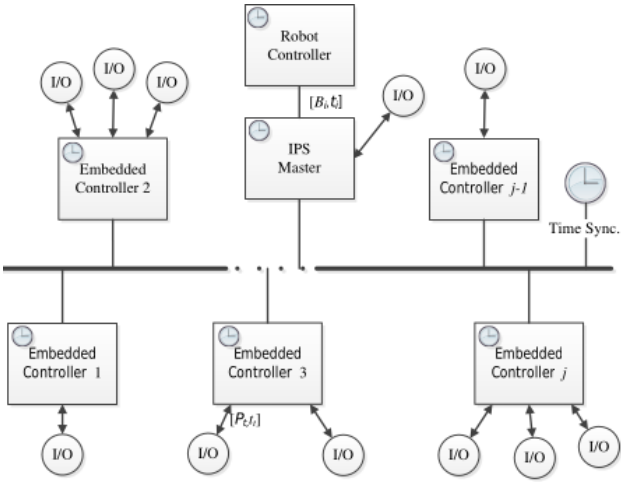


Fig. 9 ABB's Integrated Paint System as a distributed system (from [60]). Different subsystems are synchronized using a unique shared clock and communication bus. Implemented timing functions are used to trigger subsystems at the appropriate time-stamped but this needs to be heavily tested.

execution process. These testing principles have been deployed at ABB Robotics with great success [60].

5.2 Generating and Executing Test Cases on an Integrated Painting System

ABB's Integrating Painting System (IPS) is a distributed control system that embeds various real-time embedded controllers for performing high-quality painting. These controllers are useful, for example, to control the air flow, the air pressure, the pump pressure in paint flow, or the high-voltage for electrostatic charging used in painting car bodies. One of the main challenges encountered while testing IPS is related to the testing of timing characteristics of the distributed control system. Typically, the IPS is configured with embedded controllers that execute time synchronization protocols to keep their clocks synchronized, as shown in Figure 9. As there are many possible configurations for the IPS, test configurations selection is required. Note that some configurations may involve more than 20 embedded controllers.

In [59, 58], we have introduced a constraint model that is based on input channels, which are responsible for controlling exactly one physical process, for example, air or paint, involved in generating a spray pattern, as shown in Figure 10.

The IPS input is a sequence of paint-spray paths, that is, a sequence of time-stamped events (B_i, t_i), denoting the i-th paint-spray path B_i and its application time t_i. The output of the model is given by variables that represent physical values for

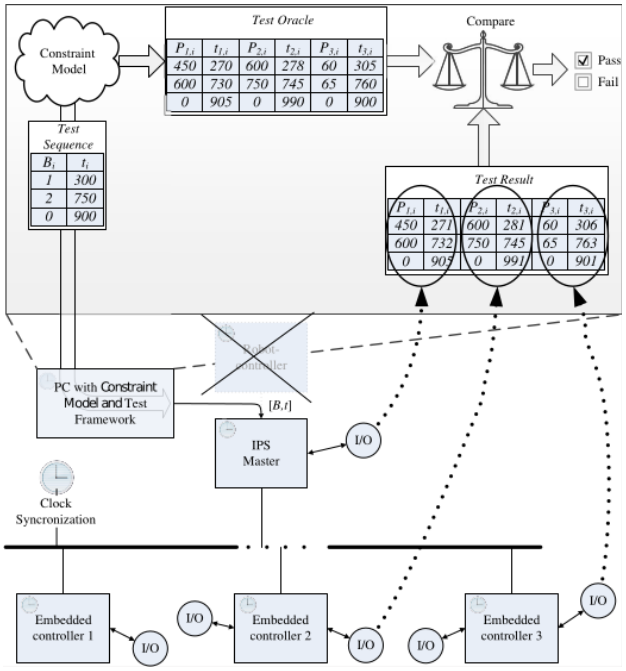


Fig. 10 A CP Model for Testing ABB's Integrated Paint System (from [60]). The *Constraint Model* generates inputs under the form of a sequence of time-stamped events (*Test Sequence*) and, based on its calculations, also produces expected outputs (*Test Oracle*). In parallel, the events are processed by the distributed system (*IPS Master*), which produces observed outputs (*Test Result*). By comparing expected outputs and observed outputs, failures are automatically detected and pointed out to the test engineer (*Compare Function*).

.

each channel j, $(P_{j,i}, t_{j,i})$. A detailed presentation of the constraint model and its constraint solving method can be found in [58, 60, 61].

Using this constraint model has been beneficial to generate test scenarios that can find subtle faults such as failing overlaps, burst scenarios or invalid shutdowns. The model has been deployed at ABB Robotics for generating such test scenarios, while test execution is typically triggered by a build server on a testbench.

In brief, the IPS software system is built every night and all the embedded controllers are upgraded with the new build release. The IPS is then configured by using either customer configurations or on-purpose test configurations. A set of simple tests, so called *smoke tests*, is then executed before solving the constraint model for new test-scenario generation. These tests are then executed by applying the generated time-stamped events sequence. As the constraint model also generates the expected outputs, the failed verdicts can be reported back to the test engineer. A detailed presentation of these steps is shown in Figure 10.

The constraint model is solved with the finite-domain constraint solving library of SICStus Prolog, named `clpfd` [18]. The solver is called through a Python front-

end layer that allows test engineers to easily integrate the test generator engine with existing build and test servers based on MS Team Foundation Server. A schematic overview of the architecture is shown in Figure 11.

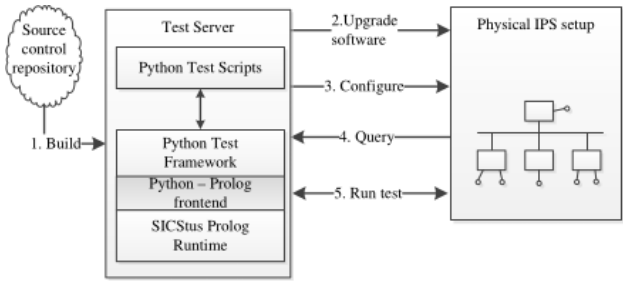


Fig. 11 Communication between the testbench and IPS (from [60]).

During the initial deployment of the constraint model, five critical bugs were discovered and immediately corrected, and several dozens of non-critical bugs were detected and progressively corrected. Today, the model is used on a daily basis and allows test engineers to continuously improve the quality of delivered IR.

6 Discussion & Conclusion

Continuous testing has emerged from the ever-growing complexity of industrial robots to become a crucial activity of the development process. An industrial robot is not only tested once before deployment, but its testing process involves frequently executing test suites to assess its robustness, performance and to quickly identify regression faults. One goal is therefore to identify or generate effective test cases that induce fault-revealing behaviour, and another goal is to continuously improve the testing process for a better exploitation of available resources.

Throughout this chapter, we discussed techniques for these two aforementioned goals of testing industrial robots. We discussed approaches for the automated testing of industrial robots both in regards to the generation of diverse and complex test scenarios as well as to the test organization under resource constraints.

The view on testing robotic systems presented in this chapter is complementary to the traditional view of validation and verification using formal methods, as presented in Chapters 7,8 and 9. By designing a formal model of the system, model checking techniques can be used to formally prove or disprove some properties of the system under verification. Interestingly, such a formal model can also be used to generate test cases for finding faults in approaches such as simulation-based testing as explained in Chapter 5 or mutation-based test case generation as presented in Chapter 11.

In software and system testing, as discussed here, generated test cases are specifically defined scripts, often independent from an exact formal model, that are part of a continuous integration process, as described in Section 3.

Today, a solid methodological foundation exists for software and system testing and for how to design and implement tests for industrial robots, but the practical adoption of these techniques is not yet state-of-the-practice. Although the goals and challenges are acknowledged, both cultural and economic resistance against their adoption have been observed [4]. A direction for future work is therefore to maintain the connections and collaborations between the testing community, often based within the domain of software engineering, and the robotics community, especially with industrial partners, to present the benefits of automated testing techniques and establish them within robotics research and practice. Examples of existing initiatives towards the principled design and engineering of robotic systems is further discussed in other chapters of this book. In Chapter 10 a software framework for distributed robotics is presented, whereas Chapter 1 discusses the adoption of Software Product Lines towards the design of robotics software. An overview of domain-specific languages for designing robotic systems and missions is given in Chapter 12. All these topics have seen their origin or strong applications within software engineering and now find similar application in robotics.

At the same time, even though the foundation is solid, there are still opportunities for methodological developments and improvements. Additional dedicated focus should be given on modern types of industrial robots like learning robots, which employ trained ML models or even continuously adapt their behaviour through learning, and collaborative robots, that work in close proximity and collaborations with humans. While ML-based components share many similarities with traditionally developed components, their testing has challenges that go beyond their basic functionality as measured through accuracy. The receptiveness for adversarial data, i.e. weaknesses due to manipulated inputs, needs to be considered as well as encountering situations outside the distribution of training data. Also traditional metrics for test completeness, such as code coverage or execution traces, are not as reliable anymore, Although there is work to adopt similar metrics for testing of ML systems, their meaning is not as intuitive or well-understood at this time and will need more consideration by the research community. Conclusively, the design of test cases and their efficient execution raises new challenges: How to construct efficient test cases that test robot components not only in isolation but also jointly? How to produce the necessary diversity in learning experiences to test the robot's learning capabilities and potential outcomes? How to model human interaction and human uncertainty?

In conclusion, the task of testing industrial robots includes open challenges for the future, even though several achievements have been observed in terms of test generation, test planning and, test execution.

Given the ongoing trend of flexible automation using lightweight, collaborative robots on the one hand and the continuing use of large-scale fixed industrial robots in automated assembly lines on the other hand, the variety of applications for testing increases. It will be the goal to transfer the existing body of knowledge and methods towards these open problems and to promote their adoption among practitioners.

Acknowledgements The authors would like to thank Morten Mossige, Mohit Kumar Ahuja, and Mathieu Collet for their tremendous contributions to the research works presented in this chapter. This work is supported by the Research Council of Norway under the T-Largo grant agreement no. 274786 (Testing of Learning Robots) and by the European Union under grant agreement no. 825619 (AI4EU).

References

[1] Abbeel P (2008) Apprenticeship learning and reinforcement learning with application to robotic control. Stanford University Stanford, CA

[2] Abi-Farraj F, Osa T, Peters NPJ, Neumann G, Giordano PR (2017) A learning-based shared control architecture for interactive task execution. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, pp 329–335

[3] Adhami L, Coste-Maniere E (2002) Positioning tele-operated surgical robots for collision-free optimal operation. In: Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292), vol 3, pp 2962–2967 vol.3

[4] Afzal A, Le Goues C, Hilton M, Timperley CS (2020) A study on challenges of testing robotic systems. In: Proceedings of the International Conference on Software Testing, Verification and Validation (ICST), ICST, vol 20

[5] Araiza-Illan D, Western D, Pipe AG, Eder K (2016) Systematic and realistic testing in simulation of control code for robots in collaborative human-robot interactions. In: Alboul L, Damian D, Aitken JM (eds) Annual Conference Towards Autonomous Robotic Systems, Springer International Publishing, Cham, pp 20–32

[6] Avizienis A, Laprie J, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1):11–33

[7] Baptiste P, Le Pape C, Nuijten W (2001) Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems, 1st edn. International Series in Operations Research & Management Science 39, Springer US

[8] Beer JM, Fisk AD, Rogers WA (2014) Toward a framework for levels of robot autonomy in human-robot interaction. J Hum-Robot Interact 3(2):74–99, DOI 10.5898/JHRI.3.2.Beer

[9] Beldiceanu N, Carlsson M (2002) A New Multi-Resource Cumulatives Constraint With Negative Heights. In: Principles and Practice of Constraint Prog. (CP'02), pp 63–79

[10] Bernard M, Kondak K, Maza I, Ollero A (2011) Autonomous transportation and deployment with aerial robots for search and rescue missions. J Field Robotics 28:914–931

[11] Bessiere C, Koriche F, Lazaar N, O'Sullivan B (2017) Constraint acquisition. Artificial Intelligence 244:315–342

[12] Billard A, Calinon S, Dillmann R, Schaal S (2008) Robot Programming by Demonstration, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 1371–1394. DOI 10.1007/978-3-540-30301-5_60

[13] Blazewicz J, Lenstra JK, Kan AR (1983) Scheduling Subject to Resource Constraints: Classification and Complexity. Discrete Applied Mathematics 5(1):11–24

[14] Bousmalis K, Irpan A, Wohlhart P, Bai Y, Kelcey M, Kalakrishnan M, Downs L, Ibarz J, Pastor P, Konolige K, Levine S, Vanhoucke V (2018) Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, Brisbane, QLD, pp 4243–4250, DOI 10.1109/ICRA.2018.8460875

[15] Brucker P, Knust S (2006) Complex Scheduling (GOR-Publications). Springer-Verlag New York, Inc., Secaucus, NJ, USA

[16] Brumfield B (2015) Car assembly line robot kills worker in germany. URL \url{http://www.cnn.com/2015/07/02/europe/germany-volkswagen-robot-kills-worker/}

[17] Carlsson M, Ottosson G, Carlson B (1997) An open-ended finite domain constraint solver. In: Programming Languages: Implementations, Logics, and Programs, 9th International Symposium, PLILP'97, Including a Special Trach on Declarative Programming Languages in Education, Southampton, UK, September 3-5, 1997, Proceedings, pp 191–206
[18] Carlsson M, Ottosson G, Carlson B (1997) An Open-Ended Finite Domain Constraint Solver. In: Proc. of the 9th Int. Symp. on Prog. Languages, Implementations, Logics, and Programs (PLILP '97), pp 191–206, DOI 10.1007/BFb0033845
[19] Chen TY, Kuo FC, Liu H, Poon PL, Towey D, Tse TH, Zhou ZQ (2018) Metamorphic Testing: A Review of Challenges and Opportunities. ACM Computing Surveys 51(1), DOI 10.1145/3143561
[20] Chu G, Stuckey PJ, Schutt A, Ehlers T, Gange G, Francis K (2016) Chuffed, a lazy clause generation solver
[21] Collet M, Gotlieb A, Lazaar N, Carlsson M, Marijan D, Mossige M (2020) Robtest: A cp approach to generate maximal test trajectories for industrial robots. In: In Proc. of the 26th Int. Conf. On Principles of Constraint Prog. (CP-20), LNCS 12333, Louvain-La-Neuve, Belgium
[22] Cormen T, Leiserson CE, Rivest RL, Stein C (2001) Introduction to Algorithms. The MIT Press, 2 edition
[23] Ding J, Kang X, Hu XH (2017) Validating a Deep Learning Framework by Metamorphic Testing. Proceedings - 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing, MET 2017 pp 28–34, DOI 10.1109/MET.2017.2
[24] Dwarakanath A, Ahuja M, Sikand S, Rao RM, Bose RPJC, Dubash N, Podder S (2018) Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pp 118–128, DOI 10.1145/3213846.3213858
[25] Elbaum S, Malishevsky A, Rothermel G (2002) Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering 28(2):159–182, DOI 10.1109/32.988497
[26] European Parliament and Council of the European Union (2006) Directive 2006/42/EC on machinery
[27] Flener P, Schmid U (2017) Inductive Programming, Springer US, Boston, MA, pp 658–666. DOI 10.1007/978-1-4899-7687-1_137
[28] Ford L, Fulkerson D (1962) Flows in Networks. Princeton University Press
[29] Garey MR, Johnson DS (1990) Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman and Co., NY
[30] Gligoric M, Eloussi L, Marinov D (2015) Ekstazi: Lightweight Test Selection. In: Proceedings of the 37th International Conference on Software Engineering, vol 2, pp 713–716, DOI 10.1109/ICSE.2015.230
[31] Goodfellow IJ, Shlens J, Szegedy C (2014) Explaining and harnessing adversarial examples. 1412.6572
[32] Gotlieb A (2015) Constraint-based testing: An emerging trend in software testing. In: Advances in Computers, vol 99, Elsevier, pp 67–101
[33] Gotlieb A, Marijan D (2014) FLOWER: Optimal Test Suite Reduction as a Network Maximum Flow. In: Proc. of Int. Symp. on Soft. Testing and Analysis (ISSTA'14), pp 171–180
[34] Gotlieb A, Liaaen M, Alexandre P (2016) Automated Regression Testing Using Constraint Programming. In: Innovative Applications of Artificial Intelligence (IAAI), pp 4010–4015
[35] Granvilliers L, Benhamou F (2006) Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques. ACM TOMS 32(1):138–156
[36] Gross H, Mueller S, Schroeter C, Volkhardt M, Scheidig A, Debes K, Richter K, Doering N (2015) Robot companion for domestic health assistance: Implementation, test and case study under everyday conditions in private apartments. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 5992–5999
[37] Guiochet J, Machin M, Waeselynck H (2017) Safety-critical advanced robots: A survey. Robotics and Autonomous Systems 94:43 – 52, DOI https://doi.org/10.1016/j.robot.2017.04.004

[38] Hartmann S, Briskorn D (2010) A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research 207(1):1–14, DOI 10.1016/j.ejor.2009.11.005

[39] Jabbarvand R, Sadeghi A, Bagheri H, Malek S (2016) Energy-aware test-suite minimization for Android apps. In: Proceedings of the 25th International Symposium on Software Testing and Analysis - ISSTA 2016, pp 425–436, DOI 10.1145/2931037.2931067

[40] Jahangirova G, Tonella P (2020) An Empirical Evaluation of Mutation Operators for Deep Learning Systems. In: IEEE International Conference on Software Testing, Verification and Validation (ICST), IEEE

[41] Jia Y, Harman M (2011) An Analysis and Survey of the Development of Mutation Testing. IEEE Transactions on Software Engineering 37(5):649–678, DOI 10.1109/TSE.2010.62

[42] Jones A, Castellano G (2018) Adaptive robotic tutors that support self-regulated learning: A longer-term investigation with primary school children. International Journal of Social Robotics 10:357–370

[43] Kim J, Feldt R, Yoo S (2019) Guiding deep learning system testing using surprise adequacy. In: Proceedings of the 41st International Conference on Software Engineering, IEEE Press, ICSE '19, p 1039–1049, DOI 10.1109/ICSE.2019.00108

[44] Kober J, Bagnell JA, Peters J (2013) Reinforcement learning in robotics: A survey. The International Journal of Robotics Research 32(11):1238–1274

[45] Lasota PA, Fong T, Shah JA (2017) A Survey of Methods for Safe Human-Robot Interaction. Foundations and Trends in Robotics 5(3):261–349, DOI 10.1561/2300000052

[46] Leong C, Singh A, Papadakis M, Traon YL, Micco J (2019) Assessing Transition-based Test Selection Algorithms at Google. In: Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, IEEE Press, Piscataway, NJ, USA, ICSE-SEIP '10, pp 101–110, DOI 10.1109/ICSE-SEIP.2019.00019

[47] Lin C, Tang K, Chen C, Kapfhammer GM (2012) Reducing the cost of regression testing by identifying irreplaceable test cases. In: 2012 Sixth International Conference on Genetic and Evolutionary Computing, pp 257–260

[48] Lindvall M, Porter A, Magnusson G, Schulze C (2017) Metamorphic Model-based Testing of Autonomous Systems. Proceedings - 2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing, MET 2017 pp 35–41, DOI 10.1109/MET.2017.6

[49] M Collet NL A Gotlieb, Mossige M (2019) Stress testing of single-arm robots through constraint-based generation of continuous trajectories. In: In IEEE Int. Conf. On Artificial Intelligence Testing (AITest'19), San Francisco, CA, USA

[50] Ma L, Liu Y, Zhao J, Wang Y, Juefei-Xu F, Zhang F, Sun J, Xue M, Li B, Chen C, et al (2018) Deepgauge: multi-granularity testing criteria for deep learning systems. Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018 DOI 10.1145/3238147.3238202

[51] Ma L, Zhang F, Sun J, Xue M, Li B, Juefei-Xu F, Xie C, Li L, Liu Y, Zhao J, Wang Y (2018) DeepMutation: Mutation Testing of Deep Learning Systems. In: 2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE), IEEE, pp 100–111, DOI 10.1109/ISSRE.2018.00021

[52] Malekzadeh M, Queißer J, Steil JJ (2017) Imitation learning for a continuum trunk robot. In: Proceedings of the 25. European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. ESANN 2017

[53] Manès VJM, Han H, Han C, Cha SK, Egele M, Schwartz EJ, Woo M (2018) The art, science, and engineering of fuzzing: A survey. arXiv: Cryptography and Security

[54] Marijan D, Gotlieb A, Sen S, Hervieu A (2013) Practical pairwise testing for software product lines. In: Proceedings of the 17th International Software Product Line Conference, Association for Computing Machinery, New York, NY, USA, SPLC '13, p 227–235, DOI 10.1145/2491627.2491646

[55] Merlet JP (2009) Interval analysis and reliability in robotics. Int J Reliability and Safety 3(1/2/3):104–130

[56] Mohammed A, Viola T, Tucker CS, Duarte J (2017) Towards co-robot navigation in manufacturing environments through machine learning of human movement patterns. Challenges for Technology Innovation 189(194):189–194
[57] Morioka M, Sakakibara S (2010) A new cell production assembly system with human–robot cooperation. CIRP Annals 59(1):9 – 12, DOI 10.1016/j.cirp.2010.03.044
[58] Mossige M, Gotlieb A, Meling H (2014) Testing Robotized Paint System Using Constraint Programming: An Industrial Case Study. In: Merayo MG, de Oca EM (eds) IFIP International Conference on Testing Software and Systems, Springer, Lecture Notes in Computer Science, pp 145–160, DOI 10.1007/978-3-662-44857-1_10
[59] Mossige M, Gotlieb A, Meling H (2014) Using CP in Automatic Test Generation for ABB Robotics' Paint Control System. In: Principles and Practice of Constraint Programming, Springer International Publishing, Cham, LNCS, vol 8656, pp 25–41, DOI 10.1007/978-3-319-10428-7_6
[60] Mossige M, Gotlieb A, Meling H (2015) Testing robot controllers using constraint programming and continuous integration. Information and Software Technology 57:169–185, DOI 10.1016/j.infsof.2014.09.009
[61] Mossige M, Gotlieb A, Meling H (2016) Generating tests for robotized painting using constraint programming. In: Kambhampati S (ed) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, IJCAI/AAAI Press, pp 4200–4204
[62] Mossige M, Gotlieb A, Spieker H, Meling H, Carlsson M (2017) Time-aware Test Case Execution Scheduling for Cyber-Physical Systems. In: Proceedings of the 23rd International Conference on Principles and Practice of Constraint Programming, LNCS, vol 10416, pp 387–404, DOI 10.1007/978-3-319-66158-2_25
[63] Nethercote N, Stuckey PJ, Becket R, Brand S, Duck GJ, Tack G (2007) MiniZinc: Towards a standard CP modelling language. In: Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings, pp 529–543, DOI 10.1007/978-3-540-74970-7_38
[64] Pei K, Cao Y, Yang J, Jana S (2017) Deepxplore. Proceedings of the 26th Symposium on Operating Systems Principles DOI 10.1145/3132747.3132785
[65] Pierson HA, Gashler MS (2017) Deep learning in robotics: a review of recent research. Advanced Robotics 31(16):821–835, DOI 10.1080/01691864.2017.1365009
[66] Qu X, Cohen MB, Rothermel G (2008) Configuration-aware regression testing: An empirical study of sampling and prioritization. In: Proceedings of the 2008 International Symposium on Software Testing and Analysis, Association for Computing Machinery, New York, NY, USA, ISSTA '08, p 75–86, DOI 10.1145/1390630.1390641
[67] Rossi F, Beek PV, Walsh T (2006) Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York, USA
[68] Rothermel G, Untch RH, Chu C, Harrold MJ (2001) Prioritizing Test Cases For Regression Testing. IEEE Transactions on Software Engineering 27(10):929–948, DOI 10.1145/347324.348910
[69] Schaal S, Peters J, Nakanishi J, Ijspeert A (2005) Learning movement primitives. In: Dario P, Chatila R (eds) Robotics Research. The Eleventh International Symposium, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 561–572
[70] Schulte C, Tack G, Lagerkvist MZ (2018) Modeling and Programming with Gecode
[71] Segura S, Fraser G, Sanchez AB, Ruiz-Cortes A (2016) A Survey on Metamorphic Testing. IEEE Transactions on Software Engineering 42(9):805–824, DOI 10.1109/TSE.2016.2532875
[72] Shen W, Wan J, Chen Z (2018) Munn: Mutation analysis of neural networks. In: 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp 108–115
[73] Shi A, Gyori A, Mahmood S, Zhao P, Marinov D (2018) Evaluating test-suite reduction in real software evolution. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2018, pp 84–94, DOI 10.1145/3213846.3213875

[74] Shi A, Zhao P, Marinov D (2019) Understanding and Improving Regression Test Selection in Continuous Integration. In: International Symposium on Software Reliability Engineering, pp 228–238

[75] Shin SY, Nejati S, Sabetzadeh M, Briand LC, Zimmer F (2018) Test case prioritization for acceptance testing of cyber physical systems: A multi-objective search-based approach. In: Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2018, pp 49–60, DOI 10.1145/3213846.3213852

[76] Simonis H, O'Sullivan B (2008) Search strategies for rectangle packing. In: Principles and Practice of Constraint Programming, Springer, LNCS, vol 5202, pp 52–66

[77] Spieker H, Gotlieb A (2020) Adaptive metamorphic testing with contextual bandits. Journal of Systems and Software 165:110574, DOI 10.1016/j.jss.2020.110574

[78] Spieker H, Gotlieb A, Marijan D, Mossige M (2017) Reinforcement Learning for Automatic Test Case Prioritization and Selection in Continuous Integration. In: Proceedings of 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017), pp 12–22, DOI 10.1145/3092703.3092709

[79] Spieker H, Gotlieb A, Mossige M (2019) Rotational Diversity in Multi-Cycle Assignment Problems. In: Thirty-Third AAAI Conference on Artificial Intelligence, pp 7724–7731, DOI 10.1609/aaai.v33i01.33017724

[80] Sunderhauf N, Brock O, Scheirer W, Hadsell R, Fox D, Leitner J, Upcroft B, Abbeel P, Burgard W, Milford M, Corke P (2018) The limits and potentials of deep learning for robotics. The International Journal of Robotics Research 37(4-5):405–420, DOI 10.1177/0278364918770733

[81] Tobin J, Biewald L, Duan R, Andrychowicz M, Handa A, Kumar V, McGrew B, Ray A, Schneider J, Welinder P, Zaremba W, Abbeel P (2018) Domain Randomization and Generative Models for Robotic Grasping. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp 3482–3489, DOI 10.1109/IROS.2018.8593933

[82] Tremblay J, Prakash A, Acuna D, Brophy M, Jampani V, Anil C, To T, Cameracci E, Boochoon S, Birchfield S (2018) Training Deep Networks With Synthetic Data: Bridging the Reality Gap by Domain Randomization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp 969–977

[83] Wang R, Lu Y, Qu B (2015) Empirical study of the effects of different profiles on regression test case reduction. IET Software 9(2):29–38, DOI 10.1049/iet-sen.2014.0008

[84] Yin H, Alves-Oliveira P, Melo FS, Billard A, Paiva A (2016) Synthesizing robotic handwriting motion by learning from human demonstrations. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence, CONF

[85] Yoo S, Harman M (2012) Regression Testing Minimization, Selection and Prioritization: A Survey. Software Testing, Verification and Reliability 22(2):67–120, DOI 10.1002/stvr.430

[86] Zacharaki A, Kostavelis I, Gasteratos A, Dokas I (2020) Safety bounds in human robot interaction: A survey. Safety Science 127:104667, DOI 10.1016/j.ssci.2020.104667