Systematic and Realistic Testing in Simulation of Control Code for Robots in Collaborative Human-Robot Interactions

Dejanira Araiza-Illan[1], David Western[1], Anthony G. Pipe[2] and Kerstin Eder[1]

[1] Department of Computer Science and Bristol Robotics Laboratory
University of Bristol, Bristol, UK
{dejanira.araizaillan,david.western,kerstin.eder}@bristol.ac.uk
[2] Faculty of Engineering Technology and
Bristol Robotics Laboratory, University of the West of England, Bristol, UK
tony.pipe@brl.ac.uk

Abstract. Industries such as flexible manufacturing and home care will be transformed by the presence of robotic assistants. Assurance of safety and functional soundness for these robotic systems will require rigorous verification and validation. We propose testing in simulation using Coverage-Driven Verification (CDV) to guide the testing process in an automatic and systematic way. We use a two-tiered test generation approach, where abstract test sequences are computed first and then concretized (e.g., data and variables are instantiated), to reduce the complexity of the test generation problem. To demonstrate the effectiveness of our approach, we developed a testbench for robotic code, running in ROS-Gazebo, that implements an object handover as part of a human-robot interaction (HRI) task. Tests are generated to stimulate the robot's code in a realistic manner, through stimulating the human, environment, sensors, and actuators in simulation. We compare the merits of unconstrained, constrained and model-based test generation in achieving thorough exploration of the code under test, and interesting combinations of human-robot interactions. Our results show that CDV combined with systematic test generation achieves a very high degree of automation in simulation-based verification of control code for robots in HRI.

1 Introduction

Robotic assistants for industrial and domestic applications are designed to interact and collaborate directly with humans. These close interactions have ethical and legal implications. Consequently, the safety and functional soundness of such technologies needs to be demonstrated for them to become viable commercial products [6]. Currently, a physical separation between robots and humans is enforced for safety, besides restrictions of speed and force.[3] These restrictions

[3] Standards ISO 13482:2014 for robotic assistants and ISO 10218 (parts I and II) for industrial robotics.

limit the scope of the applications for collaborative robots. To demonstrate that speed and force restrictions are being met, and thus to assure safety even without physical separation, the software that controls these robotic platforms must be subjected to rigorous verification and validation (V&V) processes. Software V&V needs to consider the robotic system as a whole entity, i.e. the software coupled with its hardware and electronics, as well as the reality and uncertainties of the target environments.

V&V of human-robot interactions (HRI) is challenging. The robot's environment is dynamic and uncertain (e.g., it includes people). Current V&V methods and tools are limited by computational resource bounds, restricting the degree of realism, detail, and exhaustiveness of exploration. Formal methods, e.g. model checking and theorem proving, are exhaustive and provide proof of requirement satisfaction, at the cost of employing highly abstracted models of the robotic systems and HRIs due to computational constraints, as in [23, 26]. Testing in simulations allows realism and detail [19, 20], at the cost of not being exhaustive with respect to the possibilities in the system under test (SUT), nor providing guarantees of requirement satisfaction.

Available verification methodologies from other domains, such as the microelectronics design industry, provide systematic and targeted approaches to maximize "coverage" (i.e., the extent to which a system's design has been explored) in testing. One of these methodologies is Coverage-Driven Verification (CDV), where various coverage models are used to assess exploration of the SUT and V&V completion [21]. Tests that maximize coverage –i.e., effective tests– are generated (mostly) automatically, coupled with feedback loops (automatic or manual) from automated coverage metrics collection, and automatic checks of (mostly) the SUT's response.

In test generation, constraints are commonly employed to bias testing towards rare events for coverage closure, after applying pseudo-random approaches to achieve exploration of the SUT [21, 7]. Model-based test generation uses formal methods (e.g., model checking) or other techniques to explore models in order to bias or constrain tests [25]. Nonetheless, computing tests that stimulate robotic code in a realistic or human-like manner, as it would happen in a real-life HRI scenario, makes the test generation problem quite complex.

We manage complexity via a two-tiered test generation approach. Abstract test sequences are generated first, and then instantiated to obtain concrete tests that stimulate the robotic code indirectly –i.e., the tests stimulate the human, environment, sensors and actuators in simulation, these then stimulate the robot. For example, a test requires a human to send voice commands to activate the robot in a particular order, expressed as 'send voice command' actions in the abstract layer. Code that executes these 'human' actions is assembled according to the test action sequences. The concretization of these action sequences is the production of timed sequences from the human voice model in simulation, that will stimulate simulated voice sensors, and then will send their readings to the robot's code to stimulate it. This two-tiered process is employed in model-based testing [25]. In this paper we apply unconstrained, constrained, and model-based

abstract test generation, coupled with test concretization via uniform sampling from classified ranges for variables and parameters. We demonstrate the complementary strengths of exploratory and targeted tests, particularly through model-based test generation, in achieving high levels of coverage for different coverage models, including code, cross-product, and assertions (requirements).

We tested the code for an object handover interaction between a humanoid torso and a person, envisaged for cooperative manufacture tasks, in a simulator developed in Robot Operating System[4] (ROS) and Gazebo[5], a 3D physics simulator. We employed a CDV testbench prototype developed for our simulator, fully compatible with ROS-Gazebo[6]. This paper extends our previous work in [2], with more requirements, coverage models, generated tests and simulation runs. Our testbench prototype is transferable and extendible to other robotic simulators based on ROS, and other collaborative and assistive applications.

The paper is structured as follows. We present the handover scenario in Section 2. The testbench components are presented in Section 3. A discussion of V&V and coverage results is presented in Section 4. Related work is presented in Section 5, and Section 6 concludes with an outlook on future work.

2 Case Study: Robot to Human Object Handover Task

The object handover case study was chosen because it is critical in many HRI tasks, such as cooperative manufacture, or home care. The robot platform, BERT2, is a humanoid torso with two arms [15]. A handover starts with voice activation from the person to the robot. The robot proceeds to pick up an object, holds it out to the human, and signals for the human to take it. The human indicates readiness to take the object through another voice command. Then, the robot will collect three sensor readings: "pressure," indicating whether the human is holding the object (applying force against the robot's hold of the object); "location," visually tracking that the person's hand is close to the object; and "gaze," visually tracking that the person's head is directed towards the object. Each sensor reading is classified into $G = P = L = \{\bar{1}, 1\}$, where 1 indicates the sensing was positive that the human is ready to receive the object, and $\bar{1}$ is any other sensing outcome, including null. After the sensing, the robot should decide to release the object if the human is ready, i.e. $GPL = (1, 1, 1)$ from the Cartesian product of the sensor readings (GPL for short), or it should decide not to release the object otherwise, i.e. $GPL \in \{(\bar{1}, *, *), (*, \bar{1}, *), (*, *, \bar{1})\}$, where $* \in \{1, \bar{1}\}$, within a time threshold. The person may disengage from the task before the robot makes a decision. The robot can time out whilst sensing, or while waiting for a signal.

A ROS 'node' contains the robot's action control code, comprising 212 statements in Python. The code was structured as a FSM using the SMACH modules [4], to facilitate computing a model of it for model-based test generation.

[4] http://www.ros.org/
[5] http://gazebosim.org/
[6] Available at: https://github.com/robosafe/testbench-v3

2.1 Requirements List

The following safety and functional requirements need to be verified, derived from the standard ISO 13482:2014 and previous work on handover interaction protocols and their testing in [8, 2]:

1. If the gaze, pressure and location are sensed as correct, then the object shall be released.
2. If the gaze, pressure or location are sensed as incorrect, then the object shall not be released.
3. The robot shall make a decision before a threshold of time.
4. The robot shall always either time out, decide to release the object, or decide not to release the object.
5. The robot shall not close the gripper when the human is too close.
6. The robot shall start in restricted speed.
7. The robot shall not collide with itself at high speeds.
8. The robot shall operate within allowable maximum values to avoid dangerous unintentional collisions with humans and other safety-related objects.

The last requirement was interpreted in four different quantifiable manners, considering a speed threshold of 250 mm/s based on standard ISO 10218-1:2011:

8a. The robot hand speed is always less than 250 mm/s.
8b. If the robot is within 10 cm of the human, the robot's hand speed is less than 250 mm/s.
8c. If the robot collides with anything, the robot's hand speed is less than 250 mm/s.
8d. If the robot collides with the human, the robot's hand speed is less than 250 mm/s.

2.2 Handover Simulator

A simulator of the handover scenario was developed in ROS-Gazebo. ROS is an open-source platform for the development and deployment of robotics code, using C++ and/or Python. Gazebo is a 3D physics simulator, compatible with ROS. BERT2, a cylindrical object, and the person's head and hand were modelled in Gazebo, as shown in Fig. 1. Models were developed in code for the sensors and the human action enactment.

3 A CDV Testbench for a ROS-GAZEBO Simulator

In the CDV methodology, a verification plan indicates the requirements to test, and the coverage models and metrics to use over the SUT [21, 2]. A CDV testbench has four components: the **Test Generator**, the **Driver**, the **Checker** and the **Coverage Collector**. Figure 2 shows our testbench, considering the ROS-Gazebo simulator's components. The simulator's design ensures the access to internal parameters in the robot's code and data about the physical models from Gazebo, to facilitate checking and coverage collection. The dotted line indicates feedback to the test generation for coverage closure and verification completion that may require human input.

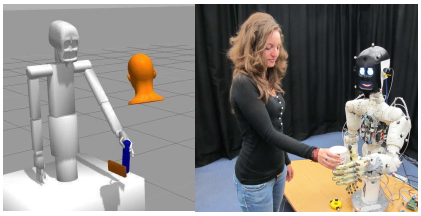

Fig. 1. The ROS-Gazebo simulation (LHS) and a real interaction from [8] (RHS).

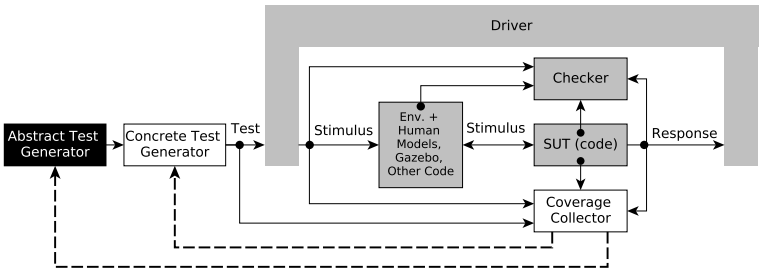

Fig. 2. Testbench and simulator elements in ROS-Gazebo. ROS nodes in gray; modules imported into ROS in white, and modules externally executed in black.

3.1 Test Generator

The aim of the test generation process is to trigger faults in the SUT (the robot's code), while exploring a wide range of scenarios. Guidance to produce effective tests comes from coverage and verification progress feedback. The generated tests must be valid and realistic, which makes the case for non-conventional software test generation approaches due to the complexity of "stimulating the robot's code in a human-like manner".

A test for the handover simulator is formed by an abstract test sequence for the human, environment, sensors and actuators (the environment surrounding the robotic code under test), which assembles code fragments to be executed concurrently by these simulator components. A concrete test is then computed, after parametrization, constraint solving and/or instantiation for all the individual parameters involved in the code fragments. We propose a two-tiered test generation approach to divide and simplify what would be a complex constraint solving, search or optimization problem. An abstract-to-concrete test construction is shown in Fig. 3.

We explored three options for the abstract test generation: pseudorandom, constrained and model-based. Pseudorandom (for repeatability purposes) is, in principle, unconstrained with respect to any assumptions about the HRI protocol. Thus, abstract test sequences are concatenated randomly, e.g., representing a person that disregards the handover protocol. To generate interesting tests, e.g, to verify a particular requirement, pseudorandom test generation can be biased

1	`sendsignal`	`activateRobot`	Send human voice A1 for 5 sec.
2	`setparam`	`time = 40`	Human waits 40×0.05 sec.
3	`receivesignal`	`informHumanOfHandoverStart`	Human waits for max. 60 sec.
4	`sendsignal`	`humanIsReady`	Send human voice A2 for 2 sec.
5	`setparam`	`time = 10`	Human waits 40×0.05 sec.
6	`setparam`	`hgazeOk = true`	Move human head in Gazebo to pose within ranges: offset $[0.1, 0.2]$, distance $[0.5, 0.6]$ and angle $[15, 40)$

Fig. 3. An abstract test sequence for the human to stimulate the robot's code (LHS), and its concretization from sampling from defined ranges (RHS).

using constraints. The implementation of these constraints requires significant manual input to be effective. Model-based test generation techniques [9, 14, 25] can target specific scenarios or requirements more effectively. In model-based test generation, a model of the system is explored or traversed in a systematic manner, e.g., through model checking for a requirement expressed as a temporal logic property [5]. A path through the model can be considered as a set of constraints [13, 17] for test generation.

For model-based test generation, the model captures both the ideal robot's code functionality and the human/environment's actions, assuming both follow the handover protocol. We chose probabilistic-timed automata (PTA) [10] models constructed manually in UPPAAL[7], to capture uncertain actions such as disengaging from the task, and the important aspect of human-like response timing in HRI. Requirements 1 to 4 (Section 2.1) were expressed as temporal logic properties, and model checked in UPPAAL. A witness trace (or path over the automata) is produced as a result of model checking, from which an abstract test sequence is extracted, disregarding the robot's actions in the trace.

3.2 Driver

The Driver distributes the resulting concrete tests into the simulator components, to be enacted to stimulate the robot indirectly. The Driver reacts to the responses of the SUT if necessary (a "reactive Driver").

3.3 Checker

The Checker monitors the response of the SUT during simulation, to detect requirement violations. Automata-based assertion monitors were implemented manually for all the requirements in Section 2.1, as in [2]. Events can be monitored at different abstraction levels, from "the robot received the correct command" (abstract), to "speed is less than the safe thresholds" (semi-continuous signals or variables). For example, the assertion monitor for Req. 5 is triggered every time the code executes the `hand(close)` function. The pose of the human hand is queried from the physical models in Gazebo. If the mass centre of the human hand is within a 0.05 m distance of the robot's hand, the monitor indicates `Failed` (requirement violation), or otherwise `Passed` (requirement satisfaction).

[7] http://www.uppaal.org/

3.4 Coverage Collector

The Coverage Collector records the progress achieved by each test in exploring the SUT. We implemented three coverage models: requirements, cross-product and code. For requirements coverage, we assessed which assertion monitors were triggered by each test.

Cross-product coverage accounts for a complete set of conceivable scenarios. We computed the Cartesian product, $Human \times Robot$, focusing on tuples where the robot times out, and different GPL selections by the human element. The set of events to cover for the human comprised: failure to activate the robot at all, sending the first activation signal but not the second, setting any combination of GPL amongst the possible 8, and disengaging whilst the robot is sensing; i.e. $Human = \{NotActive, ActivSignal, GPL = (*, *, *), Disengaged\}$. The set of events to cover for the robot comprised: timing out whilst receiving any of the two signals (voice command) from the human or whilst sensing, releasing the object, and not releasing the object; i.e. $Robot = \{TimedOut, Released, NotReleased\}$. The total size of this cross-product is of 33 tuples, but 13 of them should not be reached if the code is functionally correct. Most of the tuples that should be reachable are meaningful for the handover, since to be covered in a test, at least part of the protocol was followed correctly by the human and the robot. The cross-product coverage was computed offline from the simulation reports. Cross-product coverage (*situation* coverage) has been proposed (independently) for the verification of autonomous robots [1], including combinations of environment events only.

For code coverage, we accumulate the number of executed code statements per test, through the 'coverage'[8] Python module.

4 Experiments and Results

We verified the robot's code for the handover, with respect to the requirements in Section 2.1. The simulator ran in ROS Hydro and Gazebo 1.9, on a PC with Intel i5-3230M 2.60 GHz CPU, 8 GB of RAM, and Ubuntu 13.03. We used UPPAAL 4.0.14 for model-based test generation.

4.1 Requirements Coverage

We first generated 100 unconstrained abstract tests from uniformly sampling the set of all possible abstract human actions and producing sequences of these. We concretized each abstract test by uniformly sampling from defined ranges of variables and parameters, as dictated by the abstract actions. The tests did not cover Reqs. 1 and 8d, and other assertions were triggered less frequently (e.g. Req. 5).

Subsequently, we generated 100 constrained abstract tests that enforced the activation of the robot, in an attempt to increase the coverage, concretized in

[8] http://nedbatchelder.com/code/coverage/

Table 1. Requirements (assertion) coverage results

Req.	Unconstrained			Constrained			Model-Based		
	C	P	F	C	P	F	C	P	F
1	0/100	0/100	0/100	0/100	0/100	0/100	2/4	2/4	0/4
2	30/100	30/100	0/100	94/100	94/100	0/100	2/4	2/4	0/4
3	30/100	30/100	0/100	94/100	94/100	0/100	4/4	4/4	0/4
4	100/100	100/100	0/100	100/100	100/100	0/100	4/4	4/4	0/4
5	46/100	44/100	2/100	100/100	100/100	0/100	4/4	4/4	0/4
6	100/100	0/100	100/100	100/100	0/100	100/100	4/4	0/4	4/4
7	14/100	14/100	0/100	22/100	22/100	0/100	2/4	2/4	0/4
8a	100/100	0/100	100/100	100/100	0/100	100/100	4/4	0/4	4/4
8b	98/100	0/100	98/100	100/100	0/100	100/100	4/4	0/4	4/4
8c	96/100	5/100	91/100	99/100	0/100	99/100	4/4	0/4	4/4
8d	0/100	0/100	0/100	0/100	0/100	0/100	1/4	0/4	1/4

the same manner as the unconstrained. We based our pseudorandom generators on the procedure described in [3] for software testing. Finally, we generated four model-based abstract tests targeting Reqs. 1 to 4, to target specifically Req. 1 (also concretized like the others). A test triggered the assertion for Req. 8d, as the robot collided with the human, an important safety violation. Overall, no assertion violations were found for Reqs. 1 to 4. These results are shown in Table 1. If the assertion monitors were Covered (C), either they Passed (P) or Failed (F). The colour code in the table helps to highlight the coverage level of each assertion monitor (green for high coverage, red for no coverage).

For requirements coverage, model-based test generation is most efficient, triggering all the monitors with just four tests. The checks for Reqs. 6 and 8a-d exposed some design flaws, as the robot violates the safety speed threshold of 250 mm/s at the start of the handover, and when picking the object. This could be improved by imposing speed constraints explicitly in the motion of the robot.

4.2 Cross-Product Coverage

To target the 20 reachable tuples in the cross-product coverage (Section 3.4), we began with a different set of 100 unconstrained abstract tests, concretized as for requirements coverage. Subsequently, we employed model-based test generation to target the uncovered tuples, formulating the reachability of each tuple as a temporal logic property and model checking it in UPPAAL. Each abstract test sequence was concretized with 20 different sampling instances (column "MB 1"). Finally, we added constraints in the concretization of these abstract tests, reducing the maximum length of timeout thresholds, to trigger the *TimedOut* event in the robot's code, and produced another set of 20 concrete tests for each abstract sequence (column "MB 2").

Table 2 shows the coverage results, with a column, "TOTAL", accumulating the coverage after all the tests. These results highlight the effectiveness of model-based test generation to target the possible functionalities of the robot's code

Table 2. Reachable Cross-Product Coverage

$Human \times Robot$	Unconstr.	MB 1	MB 2	TOTAL
$\langle NotActive, TimedOut \rangle$	55/100	0/160	0/180	55/440
$\langle ActivSignal, TimedOut \rangle$	11/100	0/160	0/180	11/440
$\langle GPL = (1,1,1), TimedOut \rangle$	0/100	3/160	18/180	21/440
$\langle GPL = (1,1,1), Released \rangle$	0/100	17/160	2/180	19/440
$\langle GPL = (\bar{1},\bar{1},\bar{1}), TimedOut \rangle$	1/100	0/160	19/180	20/440
$\langle GPL = (\bar{1},\bar{1},\bar{1}), NotReleased \rangle$	25/100	0/160	1/180	26/440
$\langle GPL = (\bar{1},\bar{1},1), TimedOut \rangle$	0/100	2/160	18/180	20/440
$\langle GPL = (\bar{1},\bar{1},1), NotReleased \rangle$	2/100	18/160	2/180	22/440
$\langle GPL = (\bar{1},1,\bar{1}), TimedOut \rangle$	0/100	0/160	16/180	16/440
$\langle GPL = (\bar{1},1,\bar{1}), NotReleased \rangle$	2/100	20/160	4/180	24/440
$\langle GPL = (\bar{1},1,1), TimedOut \rangle$	0/100	0/160	17/180	17/440
$\langle GPL = (\bar{1},1,1), NotReleased \rangle$	0/100	20/160	3/180	23/440
$\langle GPL = (1,\bar{1},\bar{1}), TimedOut \rangle$	0/100	2/160	18/180	20/440
$\langle GPL = (1,\bar{1},\bar{1}), NotReleased \rangle$	4/100	18/160	2/180	24/440
$\langle GPL = (1,\bar{1},1), TimedOut \rangle$	0/100	0/160	18/180	18/440
$\langle GPL = (1,\bar{1},1), NotReleased \rangle$	0/100	20/160	2/180	22/440
$\langle GPL = (1,1,\bar{1}), TimedOut \rangle$	0/100	0/160	19/180	19/440
$\langle GPL = (1,1,\bar{1}), NotReleased \rangle$	0/100	20/160	1/180	21/440
$\langle Disengaged, NotReleased \rangle$	0/100	20/160	3/180	23/440
$\langle Disengaged, TimedOut \rangle$	0/100	0/160	17/180	17/440

and the expected critical human behaviours. For brevity, we omitted the cross-product tuples that were not reached (13/33 as mentioned in Section 3.4).

4.3 Code Coverage

The coverage of the code's 212 statements, shown in Fig. 4, was collected while running the tests for cross-product coverage. The code has been grouped using the SMACH FSM structure, and the percentages vary $\pm 2\%$ in inner decision branches. The block of code corresponding to the object's "release" was not covered by the unconstrained tests, but it was reached by the model-based tests.

In summary, while model-based test generation ensures that the requirements and the cross-product model are covered, unconstrained test generation can construct scenarios that the verification engineer has not foreseen, particularly from the environment stimulating a robot in the HRI domain.

5 Related Work

Although robotic code can be directly model checked, the focus of verification is on runtime errors, such as arrays out of bounds or unbounded loop executions, rather than functional requirements about the whole system interacting with its environment. Furthermore, formal tools are available only for selected sets of languages such as FRAMA-C or Ada-SPARK [24]. None of these tools are transferable to our robotic code in Python in a straight forward manner.

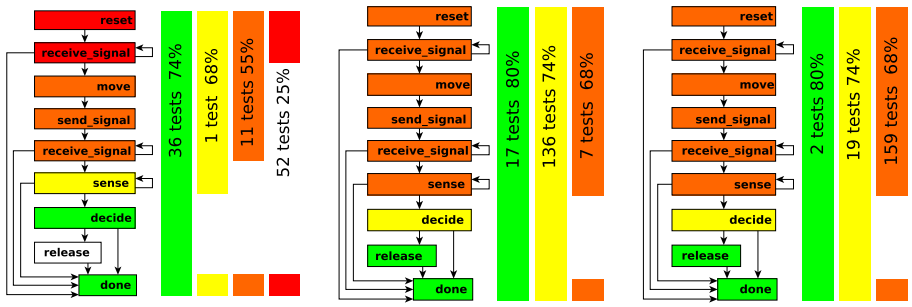

Fig. 4. Code coverage (percent values) from 100 unconstrained tests (LHS), 160 model-based (MB 1) tests (center), and 180 model-based (MB 2) tests (RHS).

In generic software testing, research has focused on generating correct and valid data inputs, while exploring their state space through intelligent sampling [7], search [12], or constraint solving [16]. In robots for HRIs, however, the test generation problem goes beyond correct and valid data. The challenge is to include realistic, human-like, environment-like, timed streams of orchestrated stimuli, which interact concurrently with the robotic code. Robotic control code has been tested systematically in real-life experiments [16], in hybrid combinations of real-life and simulations [12], and in simulation [2]. Although hybrid systems methods might seem applicable, reducing our entire test generation problem to decidable hybrid automata for model checking, or hybrid models for search or sampling [11, 22], is not straightforward.

Model-based test generation has been applied to software [25], either directly or modelled (e.g., timed automata in [18]). To be effective, such models must comprise enough details to be meaningful, yet must also be simple to traverse, modify and maintain [25]. Consequently, we propose to employ a two-tiered test generation approach, complementing model-based with unconstrained (pseudo-random) and constrained methods.

6 Conclusions

We presented an approach to verify and validate robotic code for HRI tasks in simulation-based testing, coupled with an automated CDV methodology to systematically explore the code under test, and reduce the likelihood that important scenarios will be overlooked. In simulation, a robot and its environment can be modelled with higher or lower levels of detail and realism, as necessary to guarantee safety and functional correctness, within the limits of testing regarding coverage exhaustiveness. Methodologies from other domains, such as microelectronics design verification and software testing, are transferable to the HRI domain, allowing more efficient and effective V&V for systems that are meant to work in uncertain and dynamic environments (e.g., robotic assistants).

Our automated CDV testbench, comprising of a test generator, a driver, a checker and a coverage collector, accelerates and guides the testing process, via feedback from coverage models and V&V results. We proposed the combination of different test generation methods such as unconstrained, constrained and model-based, towards coverage of the SUT from different angles, from respective coverage models. This reduces the need for hand-crafted directed tests. Additionally, a two-tiered test generation approach, from abstract to concrete, facilitates the efforts by dividing what otherwise would be a single complex constraint solving, search or optimization problem. Furthermore, we propose stimulating the robotic code through human, environment, sensor and actuator models –i.e., indirect stimulation–, to provide a greater level of realism in the V&V process.

Our approach is scalable not only in HRI, but for autonomous systems in general, as more complex systems can be verified using the same approach, for the actual system's code. The prototypes we have developed can be used for robot-in-the-loop and human-in-the-loop V&V, and can be adapted to work with other open-source or proprietary V&V software.

The handover example in this paper demonstrated the feasibility of implementing a systematic testing methodology, such as CDV, for a ROS-Gazebo based simulator. The experimental results demonstrate how feedback loops in the testbench can be exploited to seek coverage of the unexplored aspects of the code under test, or the environment's possibilities. Unconstrained test generation allows a degree of unpredictability in the human and/or environment, so that unexpected behaviours of the SUT may be exposed. Model-based test generation usefully complements the generation by systematically directing tests according to the requirements of the SUT, or towards combinations of simultaneous events in the environment and the robot.

In the future, we will apply systematic simulation-based testing to robots that learn, or that adapt to new situations. Additionally, we will explore different modelling formalisms for model-based test generation, seeking to include uncertainty, rationality and choice in different manners.

Acknowledgement: This work is part of the EPSRC-funded project "Trustworthy Robotic Assistants" (refs. EP/K006320/1 and EP/K006223/1).

References

1. Alexander, R., Hawkins, H., Rae, D.: Situation coverage – a coverage criterion for testing autonomous robots. Tech. rep., Department of Computer Science, University of York (2015)
2. Araiza-Illan, D., Western, D., Eder, K., Pipe, A.: Coverage-driven verification — an approach to verify code for robots that directly interact with humans. In: Proc. HVC. pp. 1–16 (2015)
3. Bird, D., Munoz, C.: Automatic generation of random self-checking test cases. IBM Systems Journal 22(3), 229–245 (1983)
4. Boren, J., Cousins, S.: The SMACH high-level executive. IEEE Robotics & Automation Magazine 17(4), 18–20 (2010)

5. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
6. Eder, K., Harper, C., Leonards, U.: Towards the safety of human-in-the-loop robotics: Challenges and opportunities for safety assurance of robotic co-workers. In: Proc. ROMAN. pp. 660–665 (2014)
7. Gaudel, M.: Counting for random testing. In: Proc. ICTSS. pp. 1–8 (2011)
8. Grigore, E., Eder, K., Lenz, A., Skachek, S., Pipe, A., Melhuish, C.: Towards safe human-robot interaction. In: Proc. TAROS. pp. 323–335 (2011)
9. Haedicke, F., Le, H., Grosse, D., Drechsler, R.: CRAVE: An advanced constrained random verification environment for SystemC. In: Proc. SoC. pp. 1–7 (2012)
10. Hartmanns, A., Hermanns, H.: A modest approach to checking probabilistic timed automata. In: Proc. QEST. pp. 187–196 (2009)
11. Julius, A.A., Fainekos, G.E., Anand, M., I. Lee, G.J.P.: Robust test generation and coverage for hybrid systems. In: Proc. HSCC. pp. 329–342 (2007)
12. Kim, J., Esposito, J.M., Kumar, R.: Sampling-based algorithm for testing and validating robot controllers. International Journal of Robotics Research 25(12), 1257–1272 (2006)
13. Lackner, H., Schlingloff, B.: Modeling for automated test generation a comparison. In: Proc. MBEES Workshop (2012)
14. Lakhotia, K., McMinn, P., Harman, M.: Automated Test Data Generation for Coverage: Havent We Solved This Problem Yet? In: Proc. TAIC (2009)
15. Lenz, A., Skachek, S., Hamann, K., Steinwender, J., Pipe, A., Melhuish, C.: The BERT2 infrastructure: An integrated system for the study of human-robot interaction. In: Proc. IEEE-RAS Humanoids. pp. 346–351 (2010)
16. Mossige, M., Gotlieb, A., Meling, H.: Testing robot controllers using constraint programming and continuous integration. Information and Software Technology 57, 169–185 (2014)
17. Nielsen, B., Skou, A.: Automated test generation from timed automata. Int. J. Softw. Tools Technol. Transfer. (5), 59–77 (2003)
18. Nielsen, B.: Towards a method for combined model-based testing and analysis. In: Proc. MODELSWARD. pp. 609–618 (2014)
19. Petters, S., Thomas, D., Friedmann, M., von Stryk, O.: Multilevel testing of control software for teams of autonomous mobile robots. In: Proc. SIMPAR (2008)
20. Pinho, T., Moreira, A.P., Boaventura-Cunha, J.: Framework using ROS and SimTwo simulator for realistic test of mobile robot controllers. In: Proc. CONTROLO. pp. 751–759 (2014)
21. Piziali, A.: Functional verification coverage measurement and analysis. Kluwer Academic (2004)
22. Sankaranarayanan, S., Fainekos, G.E.: Falsification of temporal properties of hybrid systems using the cross-entropy method. In: Proc. HSCC. pp. 125–134 (2012)
23. Stocker, R., Dennis, L.A., Dixon, C., Fisher, M.: Verification of Brahms human–robot teamwork models. In: Proc. JELIA. pp. 385–397 (2012)
24. Trojanek, P., Eder, K.: Verification and testing of mobile robot navigation algorithms: A case study in SPARK. In: Proc. IROS. pp. 1489–1494 (2014)
25. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability 22, 297–312 (2012)
26. Webster, M., Dixon, C., Fisher, M., Salem, M., Saunders, J., Koay, K.L., Dautenhahn, K.: Formal verification of an autonomous personal robotic assistant. In: Proc. AAAI FVHMS. pp. 74–79 (2014)