Practical Techniques for Verification and Validation of Robots

Kerstin Eder and with a demo by Dejanira Araiza Illan

University of Bristol and Bristol Robotics Laboratory
Simulation-based testing
Why and how?

D. Araiza Illan, D. Western, A. Pipe, K. Eder.

D. Araiza Illan, D. Western, A. Pipe, K. Eder.
System Complexity
“Model checking works best for well defined models that are not too huge. Most of the world is thus not covered.”

Yaron Kashai,
Fellow at the Systems and Verification R&D Division of Cadence
impossible
Traditional Approach: Directed Testing

Verification engineer sets goals and writes directed test for each item in the Verification Plan:

- **Automation**: Significant manual effort to write all the tests
- **Automation**: Work required to verify each goal was reached
- **Completeness**: Poor coverage of non-goal scenarios
  … especially the cases that you didn’t “think of”

Redo if design changes

Slide kindly provided by Cadence
Directed Test Environment

- Composition of a directed test
  - Directed tests contain more than just stimulus.
  - Checks are embedded into the tests to verify correct behavior.
  - The passing of each test is the indicator that a functionality has been exercised.

- Reusability and maintenance
  - Tests can become quite complex and difficult to understand the intent of what functionality is being verified.
  - Since the checking is distributed throughout the test suite, it is a lot of maintenance to keep checks updated.
  - It is usually difficult or impossible to reuse the tests across projects or from module to system level.

- The more tests you have the more effort is required to develop and maintain them.

Directed test approach

slide kindly provided by Cadence
Focuses on reaching **goal areas** (*versus execution of test lists)*:

- Add constraints to target a specific corner case
- Simply changing seeds generates new stimulus

**Defining Coverage “Goals” Enables Automation**

Constrained-random stimulus **generation** explores goal areas (& beyond).

**Coverage** shows which **goals** have been exercised and which need attention.

*(Self-Checking ensures proper DUT response.)*

**Automation** – Constrained-random stimulus accelerates hitting coverage goals and exposing bugs. Coverage and checking results indicate effectiveness of each simulation, which enables scaling many parallel runs.

*Even for non-goal states!*
Coverage-Driven Verification

Components of a coverage-driven verification environment
- Reusable stimulus sequences developed with "constrained random" generation.
- Running unique seeds allows the environment to exercise different functionality.
- Monitors independently watch the environment.
- Independent checks ensure correct behavior.
- Independent coverage points indicate which functionality has been exercised.
Coverage-Driven Verification
Coverage-Driven Verification
Coverage-Driven Verification

Test Generator → SUT → Response
Test Generator

- **Effective tests:**
  - meaningful events
  - interesting events
  - while exploring the system

- **Efficient tests:**
  - minimal set of tests (regression)

- **Strategies:**
  - Pseudorandom (repeatability)
  - Constrained pseudorandom
  - Model-based to target coverage directly
Model-based test generation

Formal model → Traces from model checking → Test template → Test components:
- High-level actions
- Parameter instantiation

System + environment

Environment to drive system
Model-based Test Generation

Example trace

State: robot.start, human.start
Transitions:
human to human.activateRobot
robot to robot.activateRobot

State: robot.activateRobot, human.activateRobot, time+=40
Transitions:
robot to robot.getPiece

State: robot.getPiece, human.activateRobot
Transitions:
human to human.waitSignal
robot to robot.informHuman...

State: robot.informHuman..., human.waitSignal...

High-level stimulus

send_signal activateRobot
set_param time = 40
receive_signal informHumanOfHandoverStart
send_signal humanIsReady
set_param time = 10
set_param h_onTask = true
set_param h_gazeOk = true
set_param h_pressureOk = true
set_param h_locationOk = true
Model-based Test Generation

High-level stimulus

```
send_signal activateRobot
set_param time = 40
receive_signal informHumanOfHandoverStart
send_signal humanIsReady
set_param time = 10
set_param h_onTask = true
set_param h_gazeOk = true
set_param h_pressureOk = true
set_param h_locationOk = true
```

“Human” actions in ROS

```
Parameter instantiation:
2 s

0.5 s

Gaze: (0.1 m, 0.5 m, 40°)
Location: (0.45 m, 0.05 m, 0.73 m)

Gaze: (0.1 m, 0.5 m, 30°)
Pressure: (15, 120, 140) to (7, 90, 100)
Location: (0.45 m, 0.05 m, 0.73 m)
```

Interaction done
Model-based test generation

Formal model → Traces from model checking → Test template → Test components:
- High-level actions
- Parameter instantiation

System + environment

Environment to drive system

State: robot.start, human.start
Transitions:
- human to human.activateRobot
- robot to robot.activateRobot
- robot to robot.getPiece
- robot to robot.getPiece, human.activateRobot
- robot to robot.informHuman...

State: robot.activateRobot, human.activateRobot, time=40
Transitions:
- receive_signal informHumanOfHandoverStart

State: robot.getPiece, human.activateRobot
Transitions:
- set_param time = 10

State: robot.informHuman..., human.waitSignal...
Transitions:
- set_param h.onTask = true
- set_param h.gazeOk = true
- set_param h.pressureOk = true
- set_param h.locationOk = true

Send signal
 Delay
 Receive signal
 Send signal
 Delay
 Set gaze, pressure and location
 Set gaze, pressure and location
 Interaction done
Coverage-Driven Verification

Test Generator \[\text{Test} \rightarrow \text{SUT} \rightarrow \text{Checker} \rightarrow \text{Response}\]
Requirements as assertions monitors:
- if [precondition], check [postcondition]

“If the robot decides the human is not ready, then the robot never releases an object”.

- Implemented as automata

Continuous monitoring at runtime, self-checking
- High-level requirements
- Lower-level requirements depending on the simulation's detail (e.g., path planning, collision avoidance).

```c
assert (! (robot_3D_space == human_3D_space))
```
Coverage-Driven Verification

Test Generator

Test

SUT

Checker

Response
Coverage-Driven Verification

Test Generator → Test → SUT → Response → Coverage Collector → Checker → SUT → Test Generator
Coverage Collector

- **Coverage models:**
  - **Code coverage**
    - statement
    - branch
    - expression
    - MC/DC
  - **Structural coverage**
    - FSM
Code Coverage - Limitations

- Coverage questions not answered by code coverage tools
  - Did every operation take every exception?
  - Did two instructions access the register at the same time?
  - How many times did cache miss take more than 10 cycles?
  - Does the implementation cover the functionality specified?
  - ...(and many more)

- Code coverage indicates how thoroughly the test suite exercises the source code!
  - Can be used to identify outstanding corner cases

- Code coverage lets you know if you are not done!
  - It does not indicate anything about the functional correctness of the code!

- 100% code coverage does not mean very much. 😞

- Need another form of coverage!
It is important to cover the **functionality** of the DUV.  
- Most functional requirements can’t easily be mapped into lines of code!

**Functional coverage models** are designed to assure that various aspects of the functionality of the design are verified properly, they link the requirements/specification with the implementation.

Functional coverage models are specific to a given design or family of designs.

Models cover  
- The inputs and the outputs  
- Internal states or micro architectural features  
- Scenarios  
- Parallel properties  
- Bug Models
Coverage Collector

- Coverage models:
  - Code coverage
  - Structural coverage
  - Functional coverage
    - Requirements coverage
HRI Handover Scenario

Requirements:
- Functional and safety (ISO 13482:2014, ISO 10218-1)
Requirements based on ISO 13482 and ISO 10218

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 and force.
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.
Requirements based on ISO 13482 and ISO 10218

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 and force.
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.
Requirements based on 
ISO 13482 and ISO 10218

Considering a speed threshold of 250 mm/s (from ISO 120218-1), last requirement implemented as:

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.
Coverage Collector

- Coverage models:
  - Code coverage
  - Structural coverage
  - Functional coverage
    - Requirements coverage
    - Cross-product functional coverage
      - Cartesian product of environment actions, sensor states and robot actions
A cross-product coverage model is composed of the following parts:

1. A semantic **description** of the model (story)
2. A list of the **attributes** mentioned in the story
3. A set of all the **possible values** for each attribute (the attribute value **domains**)
4. A list of **restrictions** on the legal combinations in the cross-product of attribute values

A **functional coverage space** is defined as the Cartesian product over the attribute value domains.

Cross-Product Models in e

Verification Languages, such as e, support cross-product coverage models natively.

(struct instruction {
    opcode: [NOP, ADD, SUB, AND, XOR];
    operand1 : byte;
    event stimulus;
    cover stimulus is {
        item opcode;
        item operand1;
        cross opcode, operand1
            using ignore = (opcode == NOP);
    }
};

(ADD, 00000000)
(ADD, 00000001)
(ADD, 00000010)
(ADD, 00000011)
...
(XOR, 11111110)
(XOR, 11111111)
**Situation Coverage** [2015]

Situation coverage – a coverage criterion for testing autonomous robots

*Rob Alexander*, Heather Hawkins*, Drew Rae†

* University of York, York, United Kingdom
† Griffith University, Brisbane, Australia

rob.alexander@york.ac.uk

<table>
<thead>
<tr>
<th>Car</th>
<th>Bike</th>
<th>HGV</th>
<th>Ped</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Functional Coverage

<table>
<thead>
<tr>
<th>(Gaze, Pressure, Location)</th>
<th>Sense timeout</th>
<th>Release piece</th>
<th>No release</th>
<th>Signal 1 timeout</th>
<th>Signal 2 timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ī, ĭ, Ī)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ī, ĭ, 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ī, 1, Ī)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ī, 1, 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, Ī, Ī)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, Ī, 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, Ī)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Action</th>
<th>Sense timeout</th>
<th>Release piece</th>
<th>No release</th>
<th>Signal 1 timeout</th>
<th>Signal 2 timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>No activation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activation signal 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not on task</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HRI Handover Scenario

Coverage models:
- Code statement (robot high-level control)
- Requirements in the form of Assertions
- Cross-product functional coverage
DEMO 2:

Model-Based, Coverage-Driven Verification and Validation of Code for Robots in Human-Robot Interactions.
Coverage Results
Code Coverage Results

Pseudorandom

Constrained

Model-based

Coverage Hole
Assertion Coverage Results

- 100 pseudorandomly generated tests
- 100 *constrained* pseudorandomly generated tests
- 4 model-based tests
### Functional Coverage Results

- **100 pseudorandomly generated tests**
- **160 model-based tests**
- **180 model-based constrained tests**
- **440 tests in total**

<table>
<thead>
<tr>
<th>(Gaze, Pressure, Location)</th>
<th>Sense timeout</th>
<th>Release piece</th>
<th>No release</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Action</td>
<td>Sense timeout</td>
<td>Release piece</td>
<td>No release</td>
</tr>
<tr>
<td>No activation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not on task</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(Gaze, Pressure, Location)</th>
<th>Sense timeout</th>
<th>Release piece</th>
<th>No release</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Action</td>
<td>Sense timeout</td>
<td>Release piece</td>
<td>No release</td>
</tr>
<tr>
<td>No activation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not on task</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(Gaze, Pressure, Location)</th>
<th>Sense timeout</th>
<th>Release piece</th>
<th>No release</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>19</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>18</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td>Sense timeout</td>
<td>Release piece</td>
<td>No release</td>
</tr>
<tr>
<td>No activation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not on task</td>
<td>17</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(Gaze, Pressure, Location)</th>
<th>Sense timeout</th>
<th>Release piece</th>
<th>No release</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>20</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>20</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>17</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>20</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>18</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td>19</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td>Sense timeout</td>
<td>Release piece</td>
<td>No release</td>
</tr>
<tr>
<td>No activation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not on task</td>
<td>17</td>
<td>23</td>
<td></td>
</tr>
</tbody>
</table>
Coverage-Driven Verification

Coverage analysis enables feedback to test generation
Stimulating the SUT
D. Araiza Illan, D. Western, A. Pipe, K. Eder.
Coverage-Driven Verification - An approach to verify code for robots that
directly interact with humans. Proceedings of HVC 2015, Lecture Notes in
DOI: 10.1007/978-3-319-26287-1_5 http://arxiv.org/abs/1509.04852

D. Araiza Illan, D. Western, A. Pipe, K. Eder.
Model-Based, Coverage-Driven Verification and Validation
of Code for Robots in Human-Robot Interactions.
(under review) http://arxiv.org/abs/1511.01354
What can YOU do?

K. Eder, C. Harper and U. Leonards
Towards the safety of human-in-the-loop robotics: Challenges and opportunities for safety assurance of robotic co-workers

DOI: [10.1109/ROMAN.2014.6926328](https://doi.org/10.1109/ROMAN.2014.6926328)
What next?

- **Sophisticated Test Generation**
  - model-based TG
  - refinement and abstraction

- **Safety of human assistive robots**
  - decision making, foresight windows
  - physical interaction

- **Adaptive systems:**
  - flexible specifications
  - verification at runtime

- **Walking with robots**
Summary

- Use the right tools for the job.
- No single technique is adequate to verify an entire system in practice.
  - Combine verification techniques
- Learn from areas where verification techniques are (more) mature.
- Design *for verification*!
Thank you

Any questions?

Kerstin.Eder@bristol.ac.uk
Dejanira.AraizalIlan@bristol.ac.uk

Special thanks to Dejanira Araiza Illan, David Western, Arthur Richards, Jonathan Lawry, Trevor Martin, Piotr Trojanek, Yoav Hollander, Yaron Kashai, Mike Bartley, Tony Pipe and Chris Melhuish for their hard work, collaboration, inspiration and the many productive discussions we have had.
Why red wine is so important for Christmas

Merry Christmas and a Happy New Year