Verification and Validation for Safety in Robots

Kerstin Eder
Design Automation and Verification
Trustworthy Systems Laboratory
Verification and Validation for Safety in Robots, Bristol Robotics Laboratory
Verification and Validation for Safety in Robots

To develop techniques and methodologies that can be used to design autonomous intelligent systems that are demonstrably trustworthy.
Correctness from specification to implementation

User Requirements
High-level Specification

Translate

Optimizer
Design and Analysis
(Simulink)

Implement

Controller (SW/HW)
e.g. C, C++,
RTL (VHDL/Verilog)
What can be done at the code level?

P. Trojanek and K. Eder. 
*Verification and testing of mobile robot navigation algorithms: A case study in SPARK.*
http://dx.doi.org/10.1109/IROS.2014.6942753
Navigation algorithms are fundamental for mobile robots. While the correctness of the algorithms is important, it is equally important that they do not fail because of bugs in their implementation.

What can be done at the code level?

P. Trojanek and K. Eder.  
*Verification and testing of mobile robot navigation algorithms: A case study in SPARK.*  
http://dx.doi.org/10.1109/IROS.2014.6942753
What can go wrong in robot navigation software?

Generic bugs:
- Array and vector out-of.bounds accesses
- Null pointer dereferencing
- Accesses to uninitialized data

Domain-specific bugs:
- Integer and floating-point arithmetic errors
- Mathematic functions domain errors
- Dynamic memory allocation and blocking inter-thread communication (non real-time)
Verification Approach

State of the art verification approaches:

- Model checking: infeasible
- Static analysis of C++: not possible
- Static analysis of C: requires verbose and difficult to maintain annotations

Our “Design for Verification” approach:

- SPARK, a verifiable subset of Ada
  - No Memory allocation, pointers, concurrency
- Required code modifications:
  - Pre- and post-conditions, loop (in)variants
  - Numeric subtypes (e.g. Positive)
  - Formal data containers
Results

- Three open-source implementations of navigation algorithms translated from C/C++ (2.7 kSLOC) to SPARK (3.5 kSLOC)
  - VFH+ (Vector Field Histogram)
  - ND (Nearness Diagram)
  - SND (Smooth Nearness-Diagram) navigation
  - Explicit annotations are less than 5% of the code
  - SPARK code is on average 30% longer than C/C++

- Several bugs discovered by run-time checks injected by the Ada compiler
  - Fixed code proved to be run-time safe
    - except floating-point over- and underflows
    - These require the use of complementary techniques, e.g. abstract interpretation.

- Up to 97% of the verification conditions discharged automatically by SMT solvers in less than 10 minutes

- Performance of the SPARK and C/C++ code similar
Moral

If you want to make runtime errors an issue of the past, then you must select your tools (programming language and development environment) wisely!

http://github.com/riveras/spark-navigation

Correctness from specification to implementation

- **User Requirements**
  - High-level Specification

- **Optimizer**
  - Design and Analysis
    - (Simulink)

- **Implement**

- **Controller (SW/HW)**
  - e.g. C, C++,
  - RTL (VHDL/Verilog)
Correctness from specification to implementation

User Requirements
High-level Specification

Translate

Optimizer
Design and Analysis
(Simulink)

Implement

Controller (SW/HW)
e.g. C, C++,
RTL (VHDL/Verilog)
What can be done at the design level?

*Formal Verification of Control Systems’ Properties with Theorem Proving.*
International Conference on Control (CONTROL), pp. 244 - 249. IEEE, Jul 2014.
http://dx.doi.org/10.1109/CONTROL.2014.6915147

*Verification of Control Systems Implemented in Simulink with Assertion Checks and Theorem Proving: A Case Study.*
http://arxiv.org/abs/1505.05699
Important to distinguish design flaws from coding bugs

- Analysis techniques from control systems theory (e.g., stability)
- Serve as requirements/specification
- For (automatic) code generation
Verifying Stability

Matrix $P > 0$ (Lyapunov function)

Matrix $P - (A - BK)^T P (A - BK) > 0$ (Lyapunov function's difference)

Equivalence

$V(k) - V(k-1) = x(k-1)^T [(A - BK)^T P (A - BK) - P] x(k-1)$
(Lyapunov's equation application)

Capture control systems requirements

Add as assertions

Retain in code implementation
Assertion-Based Verification
Stability

Matrix $P > 0$ (Lyapunov function)

Matrix $P - (A - BK)^T P (A - BK) > 0$ (Lyapunov function's difference)

Equivalence

$V(k) - V(k-1) = x(k-1)^T [(A - BK)^T P (A - BK) - P] x(k-1)$ (Lyapunov's equation application)

Formalize logic theory of the Simulink diagram

Axiom: $Bu = B * u$

...$

...$

Goal: vdiff == vdiff_an

Test in simulation

Automatic theorem proving

Combining Verification Techniques
Stability

Matrix \( P > 0 \) (Lyapunov function)

Matrix
\[
\begin{align*}
    \begin{bmatrix} (A-BK) \end{bmatrix}^T \begin{bmatrix} P(A-BK) \end{bmatrix} > 0 \\
    \text{(Lyapunov function's difference)}
    
\end{align*}
\]

Equivalence
\[
V(k) - V(k-1) = x(k-1)^T [(A-BK)^T P(A-BK)-P]x(k-1)
\]
(Lyapunov's equation application)

Test in simulation

First order logic theory of the Simulink diagram

Axiom: \( Bu = B * u \)

... 

... 

Goal: \( vdiff == vdiff_an \)

Automatic theorem proving
Moral

No single technique is adequate to cover a whole design in practice.

Combine techniques and learn from areas where verification is more mature.
http://github.com/riveras/simulink

**Formal Verification of Control Systems’ Properties with Theorem Proving.** International Conference on Control (CONTROL), pp. 244 - 249. IEEE, Jul 2014.  
http://dx.doi.org/10.1109/CONTROL.2014.6915147

**Verification of Control Systems Implemented in Simulink with Assertion Checks and Theorem Proving: A Case Study.**  
http://arxiv.org/abs/1505.05699
What can be done to increase the productivity of simulation-based testing?


HRI Verification Challenges

- System complexity
  - HW
  - SW
  - People
- Concurrency
- Experiments in labs
  - Expensive
  - Unsafe
We are investigating...

- Testing in simulation
- Techniques well established in microelectronics design verification
  - Coverage-Driven Verification

... to **verify** code that controls robots in HRI.
Agency for Intelligent Testing

- Robotic assistants need to be both powerful and *smart*.
  - AI and learning are increasingly used in robotics
- We need *intelligent* testing.
  - No matter how clever your robot, the testing environment needs to reflect the *agency* your robot will meet in its target environment.
CDV to automate simulation-based testing


Coverage-Driven Verification
Coverage-Driven Verification

- Test Generator
- Test
- SUT
- Response
Test Generator

- Tests must be effective and efficient
- Strategies:
  - Pseudorandom (repeatability)
Tests must be effective and efficient

Strategies:
- Pseudorandom (repeatability)
- Constrained pseudorandom
- Model-based to target specific scenarios
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

```
Send signal
Delay
Receive signal
Send signal
Delay
Set gaze, pressure and location
Set gaze, pressure and location
Interaction done
```

Parameter instantiation:

- 2s
- 0.5s

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)
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
Coverage-Driven Verification
Checker

- Requirements as assertion monitors:
  - Implemented as automata
  - if [precondition], check [postcondition]

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

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

  ```assert {! (robot_3D_position == human_3D_position)} ```
Coverage-Driven Verification

Test Generator -> Test -> SUT -> Checker -> Response

Coverage Collector
Coverage Models

- Code coverage
- Structural coverage
- Functional coverage
  - Requirements coverage
    - 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.
Coverage Models

- **Code coverage**
- **Structural coverage**
- **Functional coverage**
  - Requirements coverage
    - Functional and safety (ISO 13482:2014, ISO 10218-1)
  - Cross-product functional coverage

<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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 1, 1)</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>
Functional Coverage Results

- 100 pseudo-randomly generated tests
- 160 model-based tests
- 180 model-based constrained tests
- 440 tests in total
CDV for Human-Robot Interaction

Dejanira Araiza-Illan, David Western, Anthony Pipe and Kerstin Eder.

Coverage-Directed Verification

- systematic, goal directed verification method
  - high level of automation
  - capable of exploring systems of realistic detail under a broad range of environment conditions

- focus on **test generation** and **coverage**
  - constraining test generation requires significant engineering skill and SUT knowledge
  - **model-based test generation** allows targeting requirements and cross-product coverage more effectively than constrained pseudorandom test generation
http://github.com/robosafe/testbench


CDV provides automation

What about agency?
// INITIAL BELIEFS
preparing_for_flight:
  initialising_systems,
  ~hardware_system_passed_test,
  ~has_read_flight_environment_model,
  ~has_read_new_flight_path,
  ~pilot_comms_work,
  ~all_beacon_comms_work,
  ~created_flight_path_execution_plan,
  ~plan_isUnsafe_for_energy_level_available(Flight),
  ~announced_text_object,
  ~ready_for_flight,
  ~on_ground_before_flight,
  ~on_ground_testing,
  ~responded_to_take_off_permission,
  ~permission_given_for_take_off,
  ~flying,
  ~take_off_testing,
  ~there_is_flight_system_weakness_to_report,
  ~responded_to_start_mission,
  ~on_mission,
  ~people_pause,
  ~vehicle_pause,
  ~flying_pause,
  ~avoiding_behavior,
  ~power_return,
  ~emergency_landing,
  ~in_manual_control,
  ~landed.

// Environment Events and States

// people_appearing,
// vehicles_appearing,
// flying_object_appearing,
// weather_too_bad,
// visibility_too_bad,
// onboard_faults,
// command_received,
// manual_control_request.
Belief-Desire-Intention Agents

Desires: goals to fulfil
Beliefs: knowledge about the world
Intentions: chosen plans, according to current beliefs and goals

Guards for plans
New goals
New beliefs
From executing plans

//Initial beliefs
//Initial goals
!reset.
//Plans
+!reset : true <- add_time(20);.print("Robot is resetting");!waiting.
+!waiting : not leg <- .print("Waiting");!waiting.
+!waiting : leg <- add_time(40);.print("You asked for leg");-leg[source(human)];!grabLeg.
...
Intelligent testing is harnessing the power of BDI agent models to introduce agency into test environments.
Research Questions

- Are Belief-Desire-Intention agents suitable to model HRI?
- How can we exploit BDI agent models for test generation?
- Can machine learning be used to automate test generation in this setting?
- How do BDI agent models compare to automata-based techniques for model-based test generation?
Interacting Agents

- BDI can model agency in HRI
  - Interactions between agents create realistic action sequences that serve as test patterns
Interacting Agents

- BDI can model agency in HRI
  - Interactions between agents create realistic action sequences that serve as test patterns

![Diagram showing interactions between Agent for Simulated Human, Agents for Simulated Sensors, and Robot's Code Agent with beliefs arrows.]
BDI can model agency in HRI
- Interactions between agents create realistic action sequences that serve as test patterns
Verification Agents

- Meta agents can influence beliefs
- This allows biasing/directing the interactions
Which beliefs are effective?

- **Robot's Code Agent**
- **Agents for Simulated Sensors**
- **Agent for Simulated Human**

**Manual belief selection**

**belief subsets**

**beliefs**

**Verification Agent**

**beliefs**
Which beliefs are effective?

(Meta Agent) Verification Agent

Manual belief selection

Random belief selection

Agent for Simulated Human

Agent for Simulated Sensors

Robot’s Code Agent

belief subsets

beliefs
Which beliefs are effective?

Optimal belief sets determined through RL

belief subsets

(Meta Agent) Verification Agent

Agent for Simulated Human

Agents for Simulated Sensors

Robot’s Code Agent

plan coverage

Optimal belief sets determined through RL
Results

How effective are BDI agents for test generation?
How do they compare to model checking timed automata?


D. Araiza-Illan, A.G. Pipe, K. Eder
The cost of learning a good belief set needs to be considered when assessing the different BDI-based test generation approaches.
The cost of learning belief sets

The cost of learning a good belief set needs to be considered when assessing the different BDI-based test generation approaches.

Convergence in <300 iterations, < 3 hours

Could be sped up by adding constraints and knowledge to the learning
Code Coverage Results
Code Coverage Results

- All model-based BDI reached > 80%
- Code branches coverage
- Pseudorandom never reached > 66% in 100 tests
- Model-based + BDI vs. pseudorandom (abstract) test generation
- Per individual test, ascending order
BDI-agents vs timed automata

Effectiveness:
- high-coverage tests are generated quickly
BDI-agents vs timed automata
## BDI-agents vs Timed Automata

<table>
<thead>
<tr>
<th>Cooperative Manufacturing Assistant</th>
<th>Model checking timed automata</th>
<th>BDI agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model’s lines of code</td>
<td>725</td>
<td>348</td>
</tr>
<tr>
<td>Number of states (transitions) or plans</td>
<td>53 (72)</td>
<td>79</td>
</tr>
<tr>
<td>Modelling time</td>
<td>$\approx 10.5$ hrs</td>
<td>$\approx 6$ hrs</td>
</tr>
<tr>
<td>Model exploration time (min/test)</td>
<td>0.001 s</td>
<td>5 s</td>
</tr>
<tr>
<td>Model exploration time (max/test)</td>
<td>33.36 s</td>
<td>5 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Home Care Assistant</th>
<th>Model checking timed automata</th>
<th>BDI agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model’s lines of code</td>
<td>722</td>
<td>131</td>
</tr>
<tr>
<td>Number of states (transitions) or plans</td>
<td>42 (67)</td>
<td>35</td>
</tr>
<tr>
<td>Modelling time</td>
<td>$\approx 5.5$ hrs</td>
<td>$\approx 3$ hrs</td>
</tr>
<tr>
<td>Model exploration time (min/test)</td>
<td>0.001 s</td>
<td>1 s</td>
</tr>
<tr>
<td>Model exploration time (max/test)</td>
<td>2.775 s</td>
<td>1 s</td>
</tr>
</tbody>
</table>
Back to our Research Questions

- **Belief-Desire-Intention agents** are suitable to model HRI.
- **Traces of interactions** between BDI agent models provide **test templates**.
- **Machine learning** (RL) can be used to automate the selection of belief sets so that test generation can be biased towards maximizing coverage.
- Compared to traditional model-based test generation (model checking timed automata), BDI models are:
  - more intuitive to write, they naturally express agency,
  - smaller in terms of model size,
  - more predictable to explore and
  - equal if not better wrt coverage.
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, Springer, November 2015)

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

**Intelligent Agent-Based Stimulation for Testing Robotic Software in Human-Robot Interactions.** (Proceedings of MORSE 2016, ACM, July 2016) 
DOI: [10.1145/3022099.3022101](http://dx.doi.org/10.1145/3022099.3022101) (arXiv:1604.05508)

D. Araiza-Illan, A.G. Pipe, K. Eder 
In conclusion...

- Learn from more mature disciplines
- Select your tools and programming languages wisely
- Exploit combinations of techniques
- Automate
In conclusion...

- Learn from more mature disciplines
- Select your tools and programming languages wisely
- Exploit combinations of techniques
- Automate... *turn your solutions into “formal apps”*
- Be more clever
In conclusion...

- Learn from more mature disciplines
- Select your tools and programming languages wisely
- Exploit combinations of techniques
- Automate...*turn your solutions into “formal apps”*
- Be more clever...*use the power of AI for verification*
Thank you

Kerstin.Eder@bristol.ac.uk

Special thanks to Dejanira Araiza Illan, Jeremy Morse, David Western, Arthur Richards, Jonathan Lawry, Trevor Martin, Piotr Trojanek, Yoav Hollander, Yaron Kashai, Mike Bartley, Tony Pipe and Chris Melhuish for their collaboration, contributions, inspiration and the many productive discussions we have had.