Gaining Confidence in the Correctness of Robotic and Autonomous Systems

Kerstin Eder

Trustworthy Systems Laboratory

Verification and Validation for Safety in Robots, Bristol Robotics Laboratory
Designing Trustworthy Systems

- Create *flawless* systems.

  **AND**

- Design these systems in such a way that the *flawlessness* can be demonstrated.

"Waterfall" by M.C. Escher.
Confidence in a system’s trustworthiness can be gained in many different ways, including:

- by design, systems that are simple are also understandable;
Confidence in a system’s trustworthiness can be gained in many different ways, including:

- **by design**, systems that are simple are also understandable;
- **through transparency**, systems that allow us an insight into how they make decisions, why they act in a certain way or how they use resources become understandable;
Confidence in a system’s trustworthiness can be gained in many different ways, including

- **by design**, systems that are simple are also understandable;

- **through transparency**, systems that allow us an insight into how they make decisions, why they act in a certain way or how they use resources become understandable; and

- **through verification and validation**, rigorous proof complemented by simulation-based testing using intelligent test generation methods can provide convincing evidence of a system’s trustworthiness.
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)
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:
- Null pointer dereferencing
- Accesses to uninitialized data
- Array and vector out-of-bounds accesses
- Dynamic memory allocation and blocking inter-thread communication (non real-time)

Domain-specific bugs:
- Integer and floating-point arithmetic errors
- Mathematic functions domain errors
Design for Verification

- **SPARK**, a verifiable subset of Ada
  - Originally developed for high integrity software

- 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

- Required code modifications:
  - Pre- and post-conditions, loop (in)variants
  - Numeric subtypes (e.g. Positive)
  - Formal data containers
Results

- 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 would require the use of complementary techniques.

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

- SPARK code is on average 30% longer than C/C++

- No significant difference wrt the performance of the SPARK and C/C++ code
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

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
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)
Correctness from specification to implementation

1. User Requirements
   - High-level Specification
   - Translate

2. Optimizer
   - Design and Analysis (Simulink)
   - Implement

3. 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](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.*  
Simulink in Control System Design

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

\[ x(k+1) = Ax(k) + Bu(k) \]
\[ u(k) = -Kx(k) \]
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

\[ x(k + 1) = Ax(k) + Bu(k) \]

\[ u(k) = -Kx(k) \]
Assertion-Based Verification
Combining Verification Techniques

Stability

Matrix $P \succ 0$ (Lyapunov function)

Equivalence

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

Matrix

$P - (A - BK)^T P (A - BK) \succ 0$
(Lyapunov function's difference)

Formalize logic theory to capture the Simulink design

Axiom: $Bu = B \cdot u$
...

Goal: $v_{diff} = v_{diff\_an}$

Test in simulation

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

D. Araiza Illan, K. Eder, A. Richards. *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](http://dx.doi.org/10.1109/CONTROL.2014.6915147)

What can be done to increase the productivity of simulation-based testing?


Challenges

- Complexity
  - HW
  - SW

- High levels of Concurrency

- Experiments
  - Expensive
  - Unsafe
Challenges

- Complexity
  - HW
  - SW
  - People
  - Environment

- High levels of Concurrency

- Experiments
  - Expensive
  - Unsafe

Simulator
UE4/Carla
Testing in simulation
Techniques well established in microelectronics design verification
– Coverage-Driven Verification

… to verify code that controls robots …
We are investigating…

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

… to verify code that controls robots and AVs
CDV to automate simulation-based testing

Dejanira Araiza-Illan, David Western, Anthony Pipe and Kerstin Eder.  
*Coverage-Driven Verification — An Approach to Verify Code for Robots that Directly Interact with Humans*. In Hardware and Software: Verification and Testing, pp. 69-84. Lecture Notes in Computer Science 9434. Springer, November 2015. (DOI [10.1007/978-3-319-26287-1_5](https://doi.org/10.1007/978-3-319-26287-1_5))

Dejanira Araiza-Illan, David Western, Anthony Pipe and Kerstin Eder.  
CDV for Human-Robot Interaction

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

Tests must be effective and efficient

Strategies:
- Pseudorandom (repeatability)
Test Generator

- Tests must be effective and efficient
- Strategies:
  - Pseudorandom (repeatable)
  - Constrained pseudorandom
  - Model-based to target specific scenarios

Robot to human object handover scenario
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
Dejanira Araiza-Illan, David Western, Anthony Pipe and Kerstin Eder.

Collision detected > 8 seconds with agent_id = 37 (cyclist)
Dejanira Araiza-Illan, David Western, Anthony Pipe and Kerstin Eder.

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 (Situation 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>
</tbody>
</table>
Coverage-Directed Verification

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

- **systematic, goal directed** verification method
  - offers a **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
Dejanira Araiza-Illan, David Western, Anthony Pipe and Kerstin Eder. 
**Coverage-Driven Verification — An Approach to Verify Code for Robots that Directly Interact with Humans.** In Hardware and Software: Verification and Testing, pp. 69-84. Lecture Notes in Computer Science 9434. Springer, November 2015. (DOI: 10.1007/978-3-319-26287-1_5)

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

What about agency?
Robots and autonomous systems in general need to be both powerful and *smart*.

- AI and learning are increasingly used

**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.
// INITIAL BELIEFS
3 preparing for flight.
4 initialising systems.
5 ~hardware_system_passed_test.
6 ~has_read_flight_environment_model.
7 ~has_read_new_flight_path.
8 ~pilot_comms_work.
9 ~all Beacon_comms_work.
10 ~created_flight_path_execution_plan.
11 ~plan_is_unsafe_for_energy_level_available(Flight).
12 ~announced_text_object.
13 ~ready_for_mission.
14 ~on_ground before flight.
15 ~responded_to_take_off_permission.
16 ~permission_given_for_take_off.
17 ~flying.
18 ~take_off_testing.
19 ~there_is_flight_system_Weakness_to_report.
20 ~responded_to_start_mission.
21 ~on_mission.
22 ~people_pause.
23 ~vehicle_pause.
24 ~flying_pause.
25 ~avoiding_behaviour.
26 ~power_return.
27 ~emergency_landing.
28 ~in_manual_control.
29 ~landed.
30 // Environment Events and States
31 ~people_appearing.
32 ~vehicles_appearing.
33 ~flying_object_appearing.
34 ~weather_too_bad.
35 ~visibility_too_bad.
36 ~onboard_faults.
37 ~command_received.
38 ~manual_control_request.
39
40 // EXECUTABLE PLANS
41 // executable plan:
42 ~ease_permission_to_take_off:
43 already_for_mission <-
44 invoke(comms), runOnce, asking_for_permission("take off"),[]).1
45 // executable plan:
46 ~there_is_flight_system_Weakness_to_report:
47 ready_for_mission & ~received_take_off_permission <-
48 invoke(comms), runOnce, announcing_text_object,["R"],[]).1
49 // executable plan:
50 ~on_ground_before_flight:
51 ~received_take_off_permission:
52 invoke(comms), runOnce, announcing_text_object,["W"],[].
53 // executable plan:
54 ~take_off_testing:
55 ~there_is_flight_system_Weakness_to_report:
56 invoke(comms), runOnce, announcing_text_object,["R"],[]).
57 // executable plan:
58 ~on_mission:
59 ~people_avoiding:
60 ~vehicle_avoiding:
61 ~flying_avoiding:
62 ~avoiding_behaviour.
63 ~power_return:
64 ~emergency_landing:
65 ~in_manual_control.
66 ~landed.
67 // INTERPRETED COMMANDS
68 /* new_commands_has_arrived(Com).
69 ready_for_mission | on_mission <-
70 invoke(comms), runOnce, interpreting_commands,"Com"],["Tx"],[])
71 /* interpreted_command, commands_ unclear, did not yet acknowledge all commands */
72 /* a new_commands_has_arrived(Com).
73 /* interpreted_commands:
74 /* ready_for_mission & not_yet_acknowledge_all_commands <-+
75 /* announced_text_object, did not hear my text object */
76 /* approval timed out, pilot approved take off, pilot disapproved */
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

- 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

![Diagram showing interactions between agents](attachment:image.png)
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 agent interactions with beliefs and a question about which beliefs.](image-url)
Verification Agents

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

(Meta Agent) Verification Agent

belief subsets

Manual belief selection

beliefs

Agent for Simulated Human

beliefs

Agents for Simulated Sensors

beliefs

Robot’s Code Agent
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
Which beliefs are effective?

(Meta Agent) Verification Agent

Agent for Simulated Human

Agents for Simulated Sensors

Robot’s Code Agent

beliefs

beliefs

belief subsets

Optimal belief sets determined through RL

plan coverage

beliefs

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.

Convergence in <300 iterations, < 3 hours
Code Coverage Results
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

- **Traces of interactions** between BDI agents 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.

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

DOI: 10.1145/3022099.3022101 (arXiv:1604.05508)

D. Araiza-Illan, A.G. Pipe, K. Eder
Challenges for RAS V&V

- Specification
  - *Specification* is essential to perform V&V
    - flexible, vague and probabilistic*

- Automation, automation, automation

- Innovation
  - Creative combinations of techniques
  - Be more clever, *use the power of AI for V&V*

* J. Morse, D. Araiza-Illan, J. Lawry, A. Richards, K. Eder
Kerstin.Eder@bristol.ac.uk

Special thanks to Greg Chance, Abanoub Ghobrial, Séverin Lemaignan, 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.