Techniques

- Formal methods, co-simulation → using different methods together
- Good theory is needed to bring together good techniques
- Automation is very necessary, particularly in practice in industry
- Unifying theories of programming → theorem proving (Isabelle HOL) to generate tests for simulation, and as oracles for the results of simulation.
- Statistical methods for exploration of the state space
- Co-modelling → multi-modelling continuous/discrete, structure for what is going to be explored and what is to be verified
- We need to verify the translation processes between requirements – design – implementation
- Use implementation languages for code that facilitate correctness and verification
- Writing assumptions and requirements when people design
- Borrowing from other disciplines into autonomous systems (e.g. hardware methods)
- Specification that allows adaptation → flexible specifications
- Test generation → big challenge: valid and interesting
- Assurances for adaptive systems → shifting towards runtime assurances
- Using multi-core and parallelism for model checking
- Synthesis of behaviours that are correct vs. verifying that a behaviour is correct → model checking planning approaches. Challenges → scalability due to complexity, imprecisions of environment, different requirements, multi-robot systems, expressibility of the properties one can use for synthesis.
- Testing for systems where assumptions about the world are violated, the environment is open and hard to model and predict
- Avoiding emerging circumstances from sensors and decision making → no silver bullet. Although formal methods are useful combined with testing in simulation and finally in the real world. Gaps → specifications for autonomous systems, architectures that we can understand and describe
- Realistic testing in simulation → realistic human models, synthetic agents → the autonomous system's sensors need to think it is real, and we need to observe what the system is doing internally with no latency.
- How to generate data for real sensors?
- Monitoring the decision making to delegate control to the person.

Discussion

Generating tests effectively for systems that have agency and learn. Agency for the real environment to stimulate systems.

Confused about need for practical verification… real-world is not specifiable, so how can it be verified in simulation? We cannot test over all possible situations → curse of dimensionality.

Verification is confused in a variety of ways… in formal methods verification is a provable. Workshop context would be classified as high assurance, not formal verification. Verification → high assurance: the system works as it is supposed to.

In test environments, there may be human intervention… how to verify this would work? Hierarchy of requirements, and the specification can be wrong. There exist formal representations of humans in the system (e.g. training of operators according to manuals). Interesting → when the system is in a state where it cannot be interrupted → bound latency in interactive control a primary issue. Can you make guarantees with human in the loop? Not guarantees per se, but assumptive guarantees (identifying where, the case studies, the limits).

Exact moments where you switch between the control of the autonomous systems and the operator → latency to decide when this happens→ how do you model this for verification? Using hybrid systems and other timed formalisms, or specified by design.

What are limits of formal specification for practical verification? Test and verification groups should not try to constrain what the development folks do. Recommend verification watch-dogs methods to assist, (hopefully) verify the watch dog.

Verification of aircraft systems are expensive - $1000s of dollars per lines of code, updates require complete re-verification. Autonomous systems are going to be much more complex; are there practical verification methods for this? Is re-verification possible? What about opaque pieces like FPGAs under the operating systems? What about GPU and SIMD? These systems are being black-boxed. We need to consider all the levels in the verification from micro-architecture. Make the verification problem as small as possible so you can use the available tools. Divide-and-conquer into subsystems.

Design goal should include verification. Architecture and verification folks should collaborate from the beginning, verification feedback help guide architectural development to make easier validation. Not a single technique will work, but chance combination of methods including design for assurance perhaps will. Formal verification is not always practical.

How can all tools work together? Informing each other? There is no understanding and little research to combine tools.