Robustness Testing of Autonomy Software

Milda Zizyte, PhD Candidate advised by Dr. Philip Koopman, ECE dept.

Casidhe Hutchison, **Milda Zizyte**, Patrick E. Lanigan, David Guttendorf, Michael Wagner, Claire Le Goues, and Philip Koopman. 2018. Robustness Testing of Autonomy Software. In ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track, May 27-June 3 2018, Gothenburg, Sweden. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3183519.3183534

Overview

- **Autonomy system safety is important**
  - Robots interact with people and environment
  - Failures can cause life, property, monetary loss
- **Robustness testing can help evaluate safety**
  - Previous work in traditional SW domains
  - How do autonomy systems differ?
- **ASTAA tested 17 robotics systems over five years**
  - Unique access to robotics systems at NREC

Defining autonomy systems

- **Software systems that interact with the physical world**
- **Assist or automate some human task**
- **Comprise components that communicate via bus**
- **Usually safety-critical**

https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/

Traditional Systems vs. Autonomy Systems

Traditional SW Systems are typically…	Autonomy Systems are typically…
Procedural	Stateful
Transformational	Temporal
Monolithic	Distributed
Devoid of feedback	Cyber-physical

→ How do these differences inform robustness testing of autonomy systems?

Traditional SW Robustness Test

Send invalid inputs to SW and observe result

Past work: Fuzzing (Bart Miller), Ballista (Philip Koopman)

Input selection → Test case execution → Test evaluation

Ballista: exceptional value dictionary

`int:`
`MAXINT`

`void*:`
`NULL`

`float:`
NaN
0.0
-0.0
…

API
```
write(int filedes, const void* buffer, size_t nbytes)
```

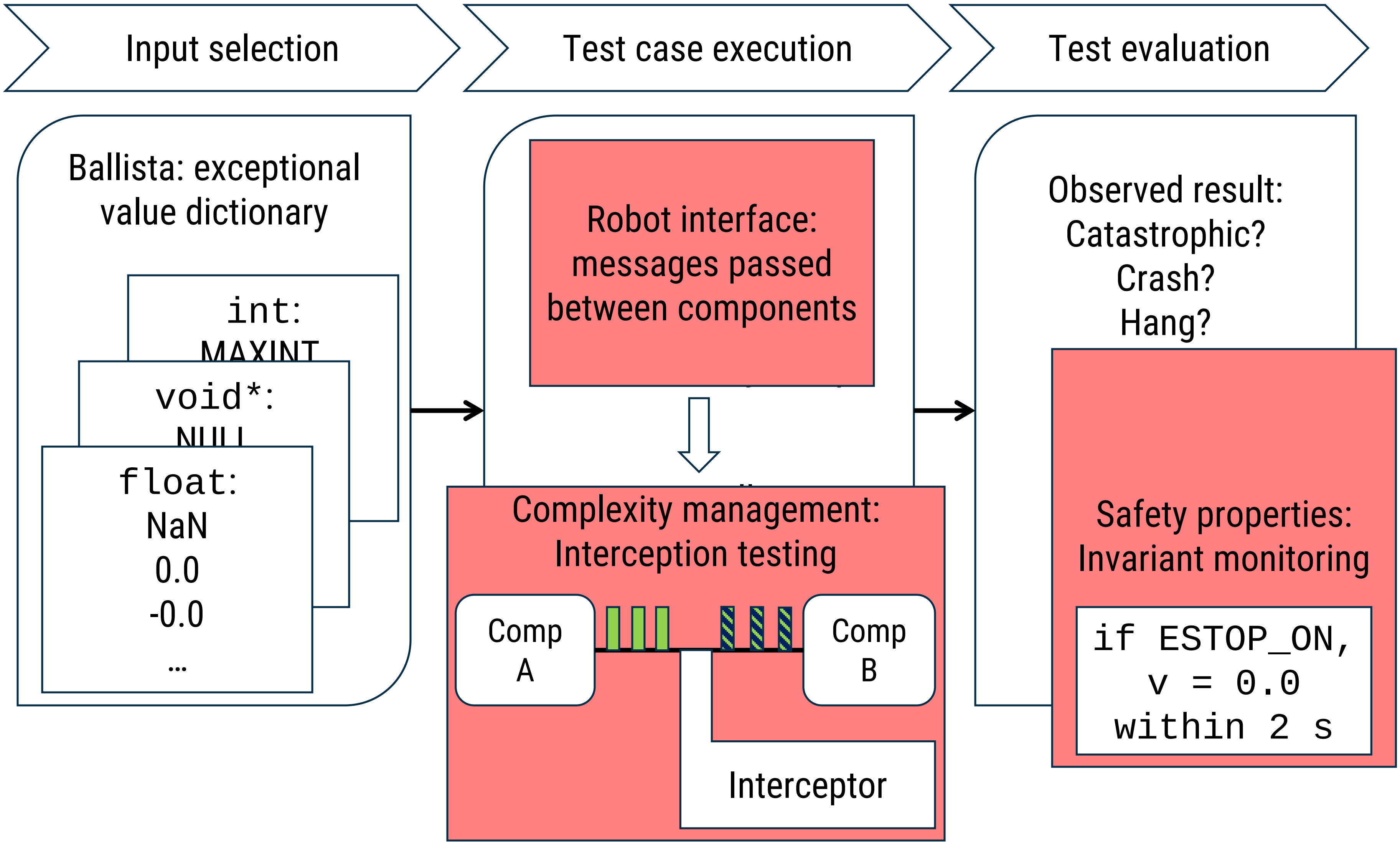
API call
```
write(FD_CLOSED, BUFF_NULL, SIZE_16)
```

Observed result:
Catastrophic?
Crash?
Hang?

Autonomy Robustness Testing - ASTAA

- **Ballista-like exceptional value dictionary approach**
- **Robots are stateful, temporal, distributed, cyber-physical:**
  - What is the interface to a robot?
  - How to deal with complexity of a robot system?
  - How to enforce safety properties?

Traditional SW Test vs. ASTAA



Testing Experiences

Researchers evaluated 150 bugs from 11 distinct projects over 4 years

From "RIOT Expanded Technical Brief, NAVAIR Public Release- 2016-842 'Approved for Public Release; distribution is unlimited'.

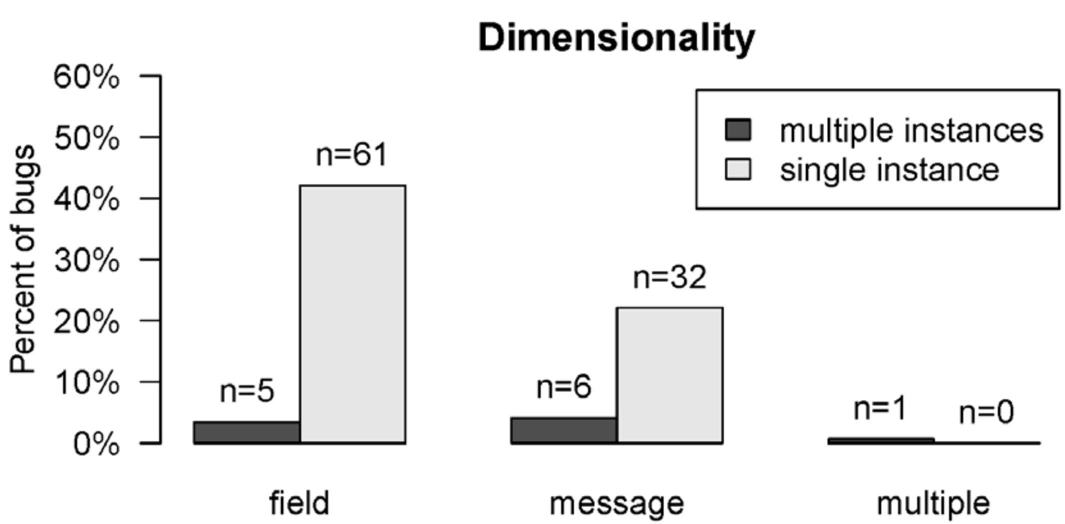
Bug classification

- **ASTAA logged 150 bugs in 11 projects**
- **Three authors analyzed each bug report independently**
  - Scaffolding messages
  - Invariants
  - Dimensionality
  - Wrappers
- **Resolved disagreements through deliberation**
- **Allows for broad qualitative discussion of autonomy systems**

Autonomy bugs are low-dimensionality

Many bugs are triggered by a very small number of inputs

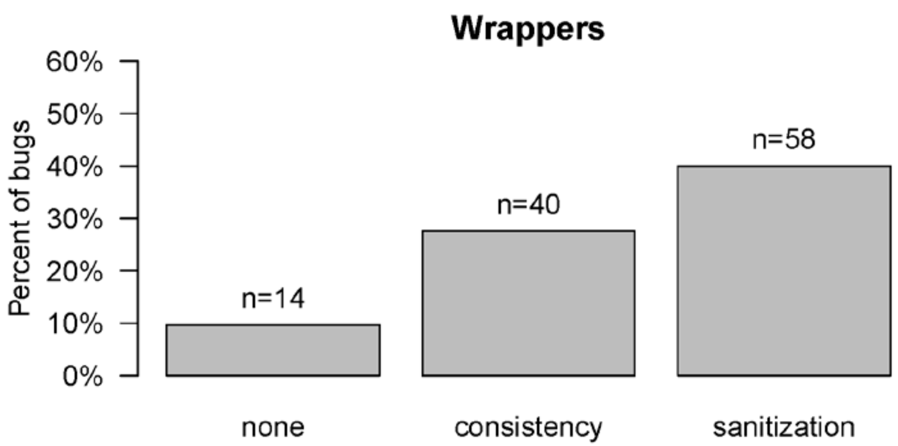
- **Dimensionality is more difficult to define than for desktop systems**
  - Interfaces: field, message, multiple
  - Instances: single, multiple
- **Most bugs (93) were activated by a single instance of one message or a single field**



Wrappers are effective

Many bugs in autonomy systems would have been avoided by using wrappers

- **Sanitization: exceptional value checks**
- **Consistency: enforcement between values**
- **Only 14 bugs not preventable by using wrappers**

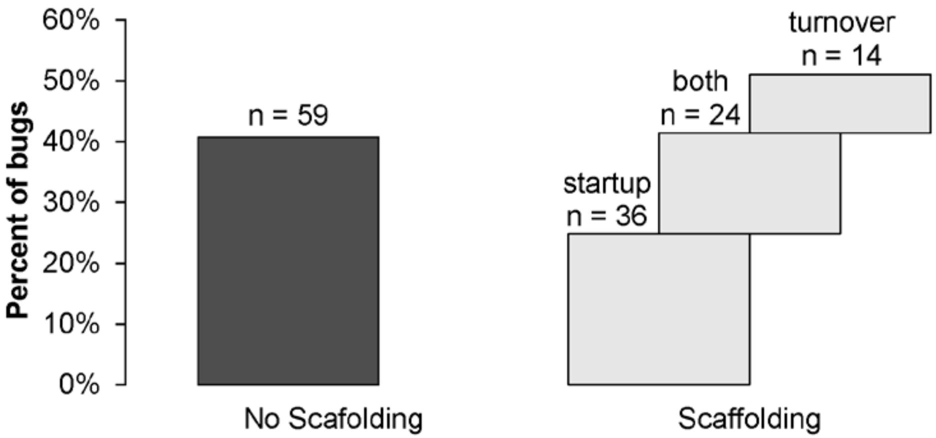


NAVAIR Public Release- 2017-35 'Approved for Public Release; distribution is unlimited'.

Scaffolding messages are neccessary

Many bugs in robotics systems can only be activated with sufficient scaffolding messages

- **Startup messages for initialization**
- **Turnover messages to keep the system running**
- **74 out of 133 classified bugs required scaffolding**

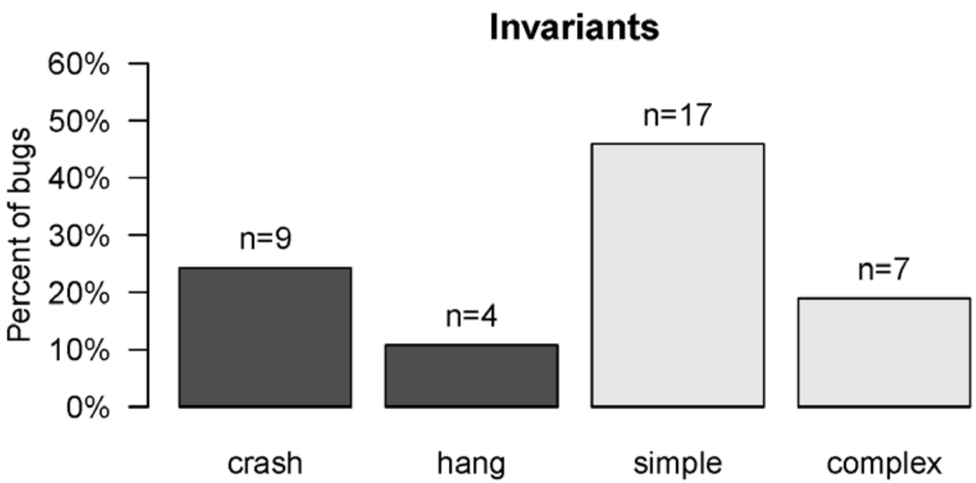


NAVAIR Public Release- 2017-35 'Approved for Public Release; distribution is unlimited'.

Invariant monitoring is valuable

Important autonomy bugs would remain uncaught if ASTAA only identified crashes or hangs

- **Some systems had no safety spec and therefore no invariants**
- **For systems with a safety spec: majority of crashes were invariant violations (image shows results for one such system)**



NAVAIR Public Release- 2017-35 'Approved for Public Release; distribution is unlimited'.

Takeaways

Having tested a large body of autonomy systems highlighted the differences and similarities vs. traditional software systems

- **Autonomy systems as software systems**
  - Low-dimensionality faults
  - Sanitization is effective
- **Unique aspects of autonomy systems**
  - Scaffolding messages
  - Invariants

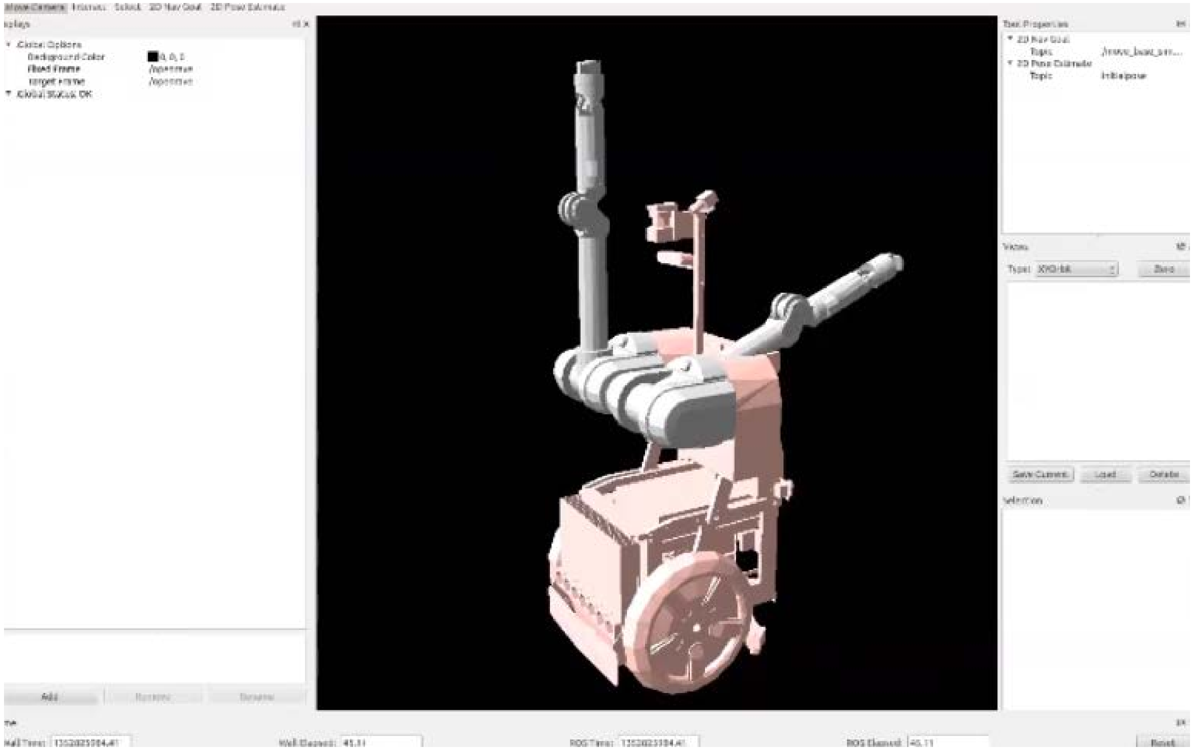
What recommendations came from ASTAA?

- **Recurring lessons observed by ASTAA team**
- **Protect your robots from data assumptions**
  - Don’t trust that your configuration is valid
  - Time is not always monotonic
  - Violations can happen between semantically redundant fields
- **Floats and NaNs are useful but dangerous**
  - Do not use floats as iterators
  - NaNs propagate
- **Plan for the system to fail**
  - Nodes should not fail silent
  - Good logging is invaluable
- **May be common sense, but keep coming up again and again in practice!**

Robot Arm Example

Mature robot built on ROS sent an exceptional but logical arm angle

https://www.youtube.com/watch?v=kK6iKwjKA54



Summary

- **ASTAA expands traditional SW testing techniques for autonomy systems**
  - Built for stateful, temporal, distributed, cyber-physical autonomy systems
  - Messages as interface, interception, invariants
- **Testing autonomy systems provides insight into their behavior**
  - Opportunity at NREC to test many industry robots in academic setting
  - Autonomy systems are similar to traditional SW systems:
    - Bugs are low-dimensionality
    - Sanitization is effective
  - Testing autonomy systems requires novel approaches:
    - Scaffolding messages are important
    - Invariant monitoring is important
- **Robustness testing can inform autonomy development practices**