CHALLENGES IN AUTONOMOUS VEHICLE TESTING AND VALIDATION

Philip Koopman, Carnegie Mellon University

Michael Wagner, Edge Case Research LLC

Paper at: https://users.ece.cmu.edu/~koopman/pubs.html

SAE INTERNATIONAL

Carnegie Mellon University

Overview:
Fully Autonomous Vehicles Are Cool!

https://en.wikipedia.org/wiki/Autonomous_car

But what about fleet deployment?

- Need V&V beyond just road tests
  - High ASIL assurance requires a *whole lot* of testing & some optimism
  - Machine-learning based autonomy is brittle and lacks “legibility”
- What breaks when mapping full autonomy to safety V model?
  - Autonomy requirements/high level design are implicit in training data
  - What “controllability” do you assign for full autonomy?
  - Nondeterministic algorithms yield non-repeatable tests
- Potential strategies for safer autonomous vehicle designs
  - Safing missions to minimize fail-operational cost
  - Run-time safety monitors using traditional high-ASIL software
  - Accelerated stress testing via fault injection

Validating High-ASIL Systems via Testing Is Challenging

Need to test for at least ~3x crash rate to validate safety

- Hypothetical fleet deployment: New York Medallion Taxi Fleet
  - – 13,437 vehicles, average 70,000 miles/yr = 941M miles/year
    - 7 critical crashes in 2015

[2014 NYC Taxi Fact Book]

[Fatal and Critical Injury data / Local Law 31 of 2014]

→ 134M miles/critical crash (death or serious injury)

- Assume testing representative; faults are random independent
  - – $R(t) = e^{-lamba*t}$ is the probability of not seeing a crash during testing

- Illustrative: How much testing to ensure critical crash rate is at least as good as human drivers? → (At least 3x crash rate)
  - – These are optimistic test lengths…
    - Assumes random independent arrivals
    - Is simulated driving accurate enough?

Testing Miles	Confidence if NO critical crash seen
122.8M	60%
308.5M	90%
401.4M	**95%**
617.1M	99%

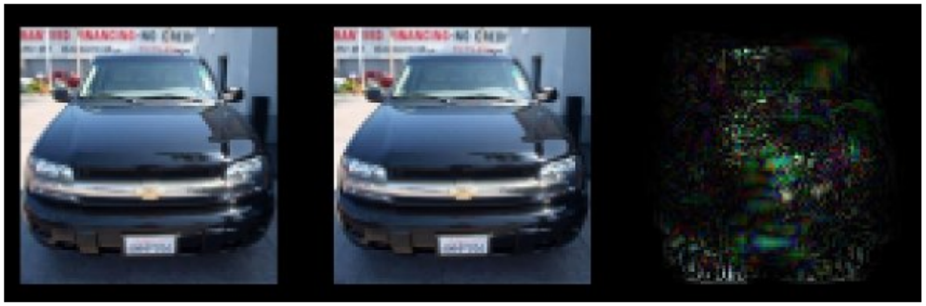
Using chi-square test from: http://reliabilityanalyticstoolkit.appspot.com/mtbf_test_calculator

Machine Learning Might Be Brittle & Inscrutable

Legibility: can humans understand how ML works?

- Machine Learning “learns” from training data
  - Result is a weighted combination of “features”
- Commonly the weighting is inscrutable, or at least not intuitive
  - There is an unknown (significant?) chance results are brittle
    - E.g., accidental correlations in training data, sensitivity to noise

QuocNet:



Car

Not a Car

Magnified Difference

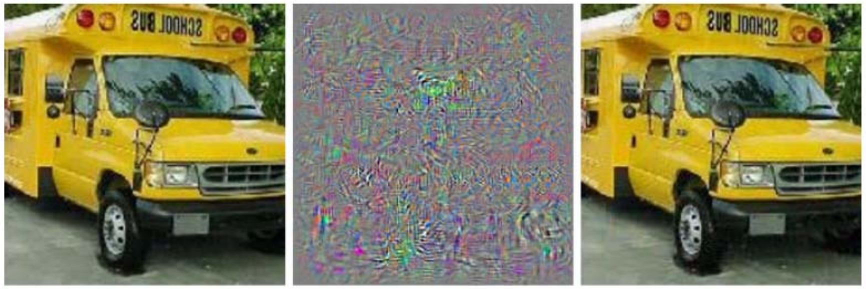
Szegedy, Christian, et al. "Intriguing properties of neural networks." *arXiv preprint arXiv:1312.6199* (2013).

AlexNet:

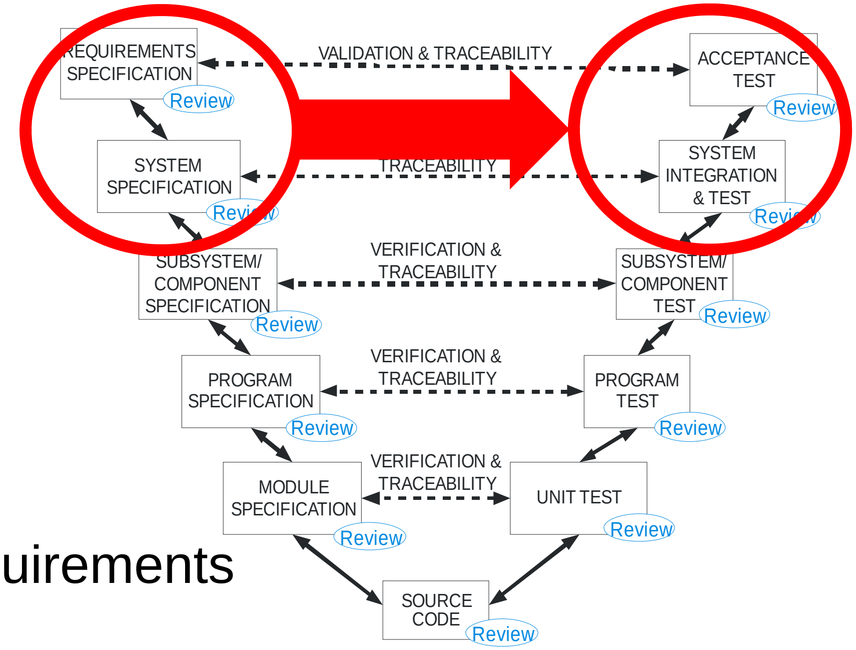
Bus

Magnified Difference

Not a Bus

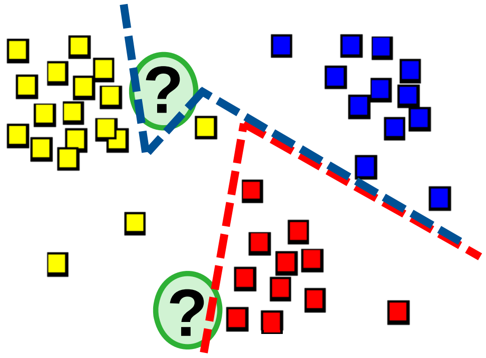


Where Are the Requirements for Machine Learning?


REQUIREMENTS SPECIFICATION
Review
VALIDATION & TRACEABILITY
ACCEPTANCE TEST
Review
SYSTEM SPECIFICATION
Review
TRACEABILITY
SYSTEM INTEGRATION & TEST
Review
SUBSYSTEM/ COMPONENT SPECIFICATION
Review
VERIFICATION & TRACEABILITY
SUBSYSTEM/ COMPONENT TEST
Review
PROGRAM SPECIFICATION
Review
VERIFICATION & TRACEABILITY
PROGRAM TEST
Review
MODULE SPECIFICATION
Review
VERIFICATION & TRACEABILITY
UNIT TEST
Review
SOURCE CODE
Review

Machine Learning requirements are the training data

- V model traces reqts to V&V
- Where are the requirements in a machine learning based system?
  - ML system is just a framework
  - The **training data** forms de facto requirements
- How do you know the training data is “complete”?
  - Training data is safety critical
  - What if a moderately rare case isn’t trained?
    - It might not behave as you expect
    - People’s perception of “almost the same” does not necessarily predict ML responses!

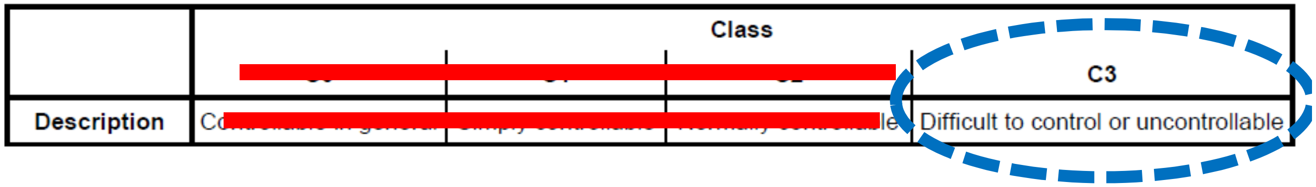


Cluster Analysis

How Do We Assess Controllability?

ISO 26262 bases ASIL in part on Controllability

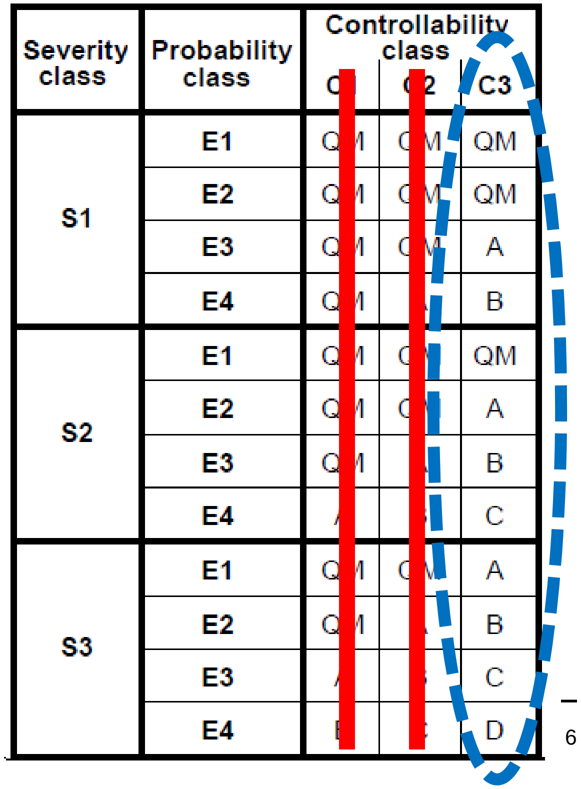
Table 3 — Classes of controllability



	Class			
	~~C0~~	~~C1~~	~~C2~~	C3
Description	~~Controllable in general~~	~~Simply controllable~~	~~Normally controllable~~	Difficult to control or uncontrollable

- If vehicle is fully autonomous, perhaps this means *zero* controllability
  - Are full emergency controls available?
  - Will passenger be awake to use them?
  - How much credit can you take for the proverbial "big red button"?
- Can you take credit for controllability of an independent emergency shutdown system?
  - Or, do we need "C4" for autonomy?

Table 4 — ASIL determination



Severity class	Probability class	Controllability class		
		~~C1~~	~~C2~~	C3
S1	E1	~~QM~~	~~QM~~	QM
	E2	~~QM~~	~~QM~~	QM
	E3	~~QM~~	~~QM~~	A
	E4	~~QM~~	~~A~~	B
S2	E1	~~QM~~	~~QM~~	QM
	E2	~~QM~~	~~QM~~	A
	E3	~~QM~~	~~A~~	B
	E4	~~A~~	~~B~~	C
S3	E1	~~QM~~	~~QM~~	A
	E2	~~QM~~	~~A~~	B
	E3	~~A~~	~~B~~	C
	E4	~~B~~	~~C~~	D

Testing Non-Deterministic Algorithms

How Do You Test a Randomized Algorithm?

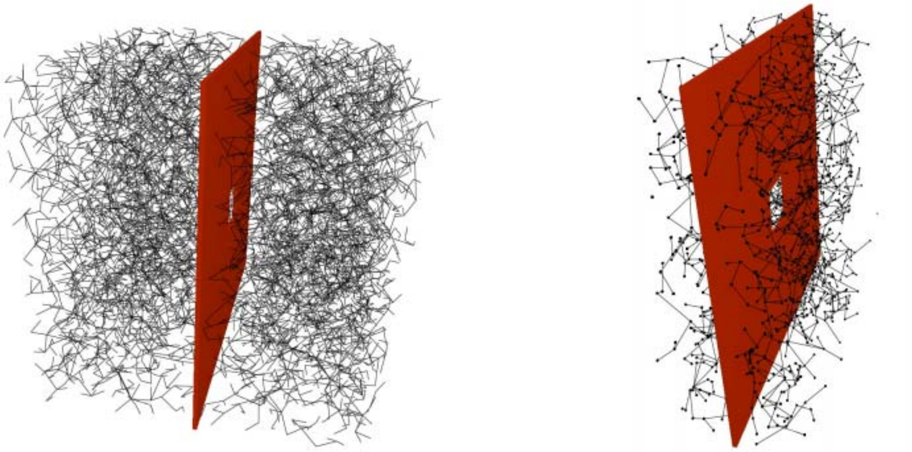


Fig. 1. The roadmap graph we get for the difficult hole test scene used in this paper. The left image shows the graph using halton sampling and the right image uses gaussian sampling.

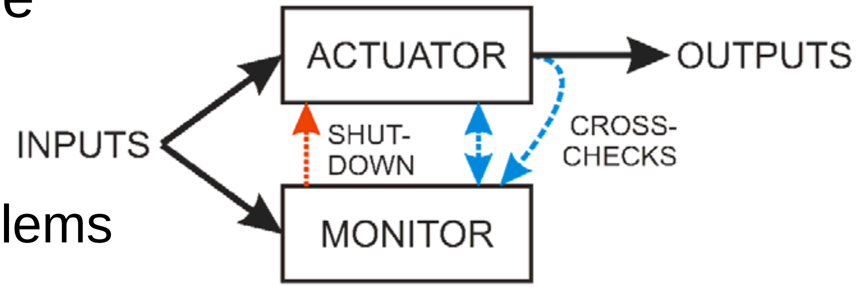
[Geraerts & Overmars, 2002]

- Example: Randomized path planner
  - Randomly generate solutions
  - Pick best solution based on fitness or goodness score
- Implications for testing:
  - If you can carefully control random number generator, *maybe* you can reproduce behavior in unit test
  - At system level, generally sensitive to initial conditions
    - Can be essentially impossible to get test reproducibility in real systems
    - In practice, significant effort to force or “trick” robot into displaying behavior

Run-Time Safety Monitors

Approach: Enforce Safety with Monitor/Actuator Pair

- "Actuator" is the ML-based software
  - – Usually works
  - – But, might sometimes be unsafe
  - – Actuator failures are drivability problems



- All safety requirements are allocated to Monitor
  - – Monitor performs safety shutdown if unsafe outputs/state detected
  - – Monitor is non-ML software that enforces a safety "envelope"

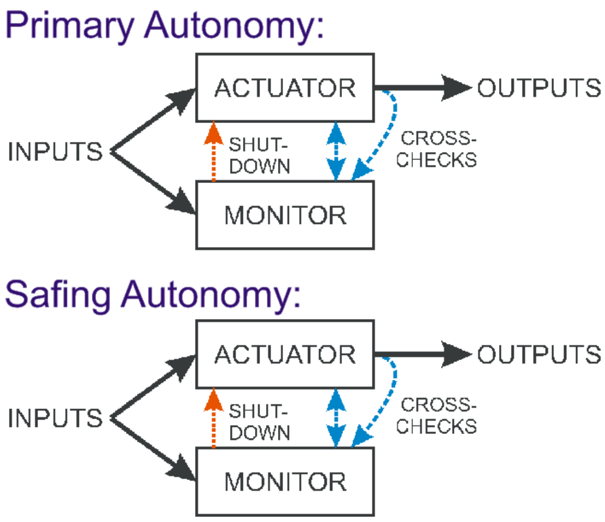
- In practice, we've had significant success with this approach
  - – E.g., over-speed shutdown on APD
  - – Important point: need to be clever in defining what "safe" means to create monitors
  - – Helps define testing pass/fail criteria too

APD is the first unmanned vehicle to use the Safety Monitor. (Unclassified: Distribution A. Approved for Public Release. TACOM Case # 19281 Date: 20 OCT 2009)

Safing Missions To Reduce Redundancy Requirements

What Happens When Primary Autonomy Has a Fault?

- Can't trust a sick system to act properly
  - With safety monitor approach, the monitor/actuator pair shuts down
  - But, you need to get car to safe state
- Bad news: need automated recovery
  - If driver drops out of loop, can't just say "it's your problem!"

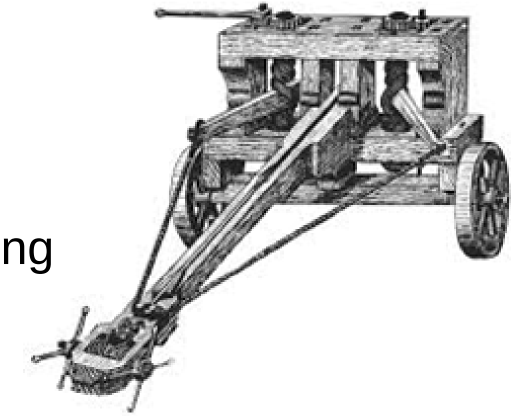


- Good news: short duration recovery mission makes things easier
  - Cars only need a few seconds to get to side of road or stop in lane
  - Think of this as a "safing mission" like diverting an aircraft
    - Easier reliability because only a few seconds for something else to fail
    - Easier requirements because it is a simple "stop vehicle" mission
    - In general, can get much simpler, inexpensive safing autonomy

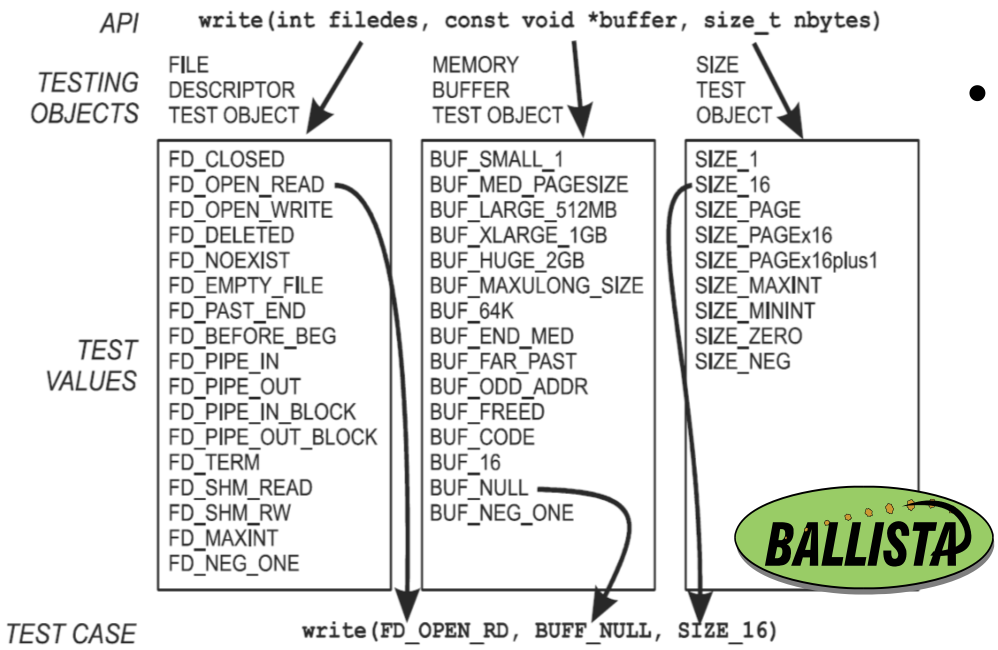
What About Unusual Situations and Unknown Unknowns?

Use Robustness Testing (SW Fault Injection) to Stress Test

- Apply combinations of valid & invalid parameters to interfaces
  - Subroutine calls (e.g., null pointer passed to subroutine)
  - Data flows (e.g., NaN passed as floating point input)
  - Subsystem interfaces (e.g., CAN messages corrupted on the fly)
  - System-level digital inputs (e.g., corrupted Lidar data sets)
- In our experience, robustness testing finds interesting bugs
  - You can think of it as a targeted, specialized form of fuzzing
- Results:
  - Finds functional defects in autonomous systems
    - Basic design faults, not just exception handling
    - Commonly finds defects missed in extensive field testing
  - Is capable of finding architectural defects
    - e.g., finds missing but necessary redundancy



Basic Idea of Scalable Robustness Testing



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

TESTING OBJECTS

FILE DESCRIPTOR TEST OBJECT

MEMORY BUFFER TEST OBJECT

SIZE TEST OBJECT

TEST VALUES

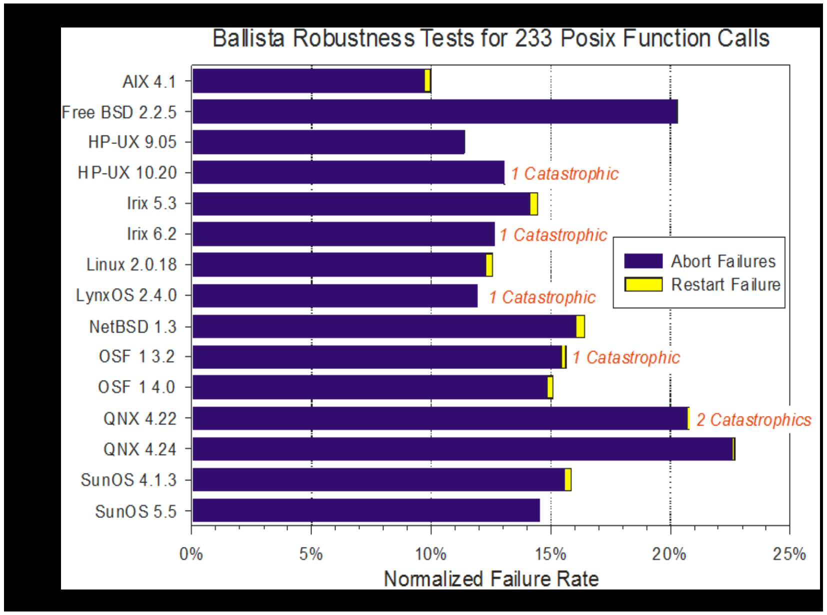
FD_CLOSED
FD_OPEN_READ
FD_OPEN_WRITE
FD_DELETED
FD_NOEXIST
FD_EMPTY_FILE
FD_PAST_END
FD_BEFORE_BEG
FD_PIPE_IN
FD_PIPE_OUT
FD_PIPE_IN_BLOCK
FD_PIPE_OUT_BLOCK
FD_TERM
FD_SHM_READ
FD_SHM_RW
FD_MAXINT
FD_NEG_ONE

BUF_SMALL_1
BUF_MED_PAGESIZE
BUF_LARGE_512MB
BUF_XLARGE_1GB
BUF_HUGE_2GB
BUF_MAXULONG_SIZE
BUF_64K
BUF_END_MED
BUF_FAR_PAST
BUF_ODD_ADDR
BUF_FREED
BUF_CODE
BUF_16
BUF_NULL
BUF_NEG_ONE

SIZE_1
SIZE_16
SIZE_PAGE
SIZE_PAGEx16
SIZE_PAGEx16plus1
SIZE_MAXINT
SIZE_MININT
SIZE_ZERO
SIZE_NEG

TEST CASE write(FD_OPEN_RD, BUFF_NULL, SIZE_16)

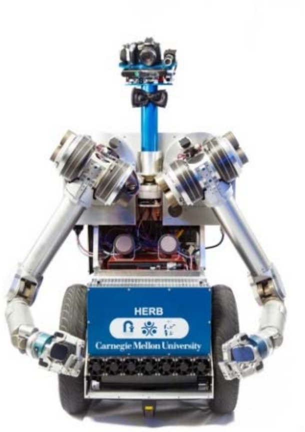
- **Use testing dictionary based on data types**
  - Random combinations of pre-selected dictionary values
  - Both valid and exceptional values



- **Caused task crashes and kernel panics on commercial desktop OS**
  - But what about on robots?
  - Use Robustness testing for stress + run-time monitoring for pass/fail detector

Example Autonomous Vehicle Defects Found via Robustness Testing

ASTAA Project at NREC found system failures due to:



Improper handling of floating-point numbers:

- Inf, NaN, limited precision

Array indexing and allocation:

- Images, point clouds, etc…
- Segmentation faults due to arrays that are too small
- Many forms of buffer overflow, especially dealing with complex data types
- Large arrays and memory exhaustion

ROS

Time:

- Time flowing backwards, jumps
- Not rejecting stale data

Problems handling dynamic state:

- For example, lists of perceived objects or command trajectories
- Race conditions permit improper insertion or removal of items
- Vulnerabilities in garbage collection allow memory to be exhausted or execution to be slowed down

The Black Swan Meets Autonomous Vehicles

Suggested Philosophy for Testing Autonomous Vehicles:

- Some testing should look for proper functionality
  - But, some testing should attempt to ***falsify a correctness hypothesis***
- Much of vehicle autonomy is based on Machine Learning
  - ML is inductive learning… which is vulnerable to black swan failures
  - We’ve found robustness testing to be useful in this role

Thousands of miles of “white swans”…

Make sure to fault inject some “black swans”

Conclusions

Fully Autonomous vehicles have fundamental differences

- Doing enough testing is challenging. Even worse…
  - Machine learning systems are inherently brittle and lack "legibility"
- Challenges trying to map to traditional V model for safety
  - Training data is the de facto requirement+design information
  - What are "controllability" implications for assigning an ASIL?
  - Non-determinism makes it difficult to do testing
- Potential solution elements:
  - Safing missions to minimize fail-operational costs
  - Run-time safety monitors worry about safety, not "correctness"
  - Accelerated stress testing via fault injection finds defects that were otherwise missed in vehicle-level testing
  - Testing philosophy should include black swan events