Monitoring Cyber-Physical Systems

Alexandre Donzé, Decyphir

Winter Training School on Runtime Verification

March 19th, 2018

Cyber-Physical Systems

Autonomy

Control

Computation

Smart Systems

Can we use **formal methods** to improve **safety, reliability and performance** of these systems?

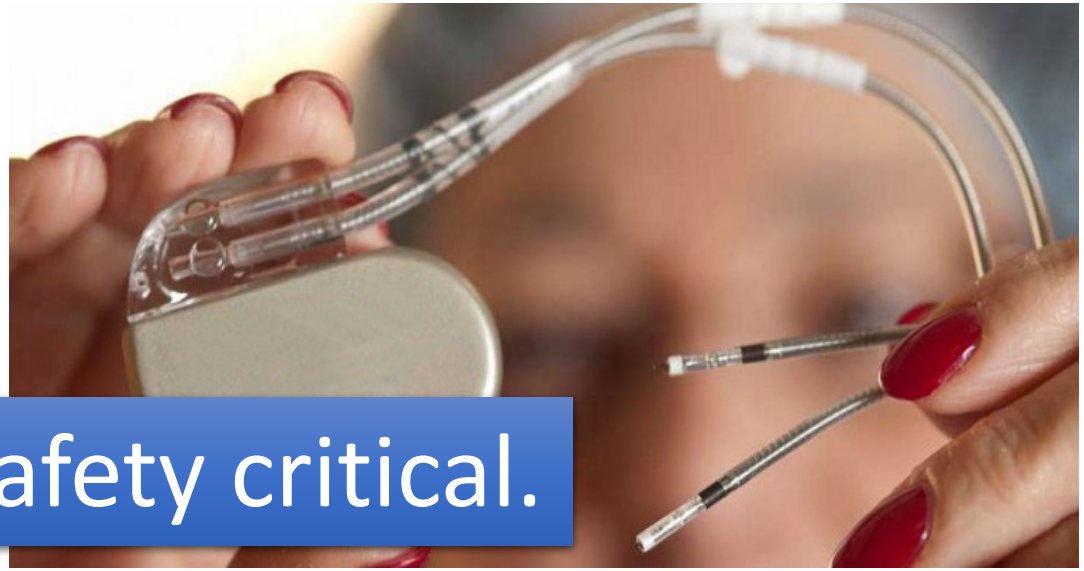
Communication

Human-in-the-loop

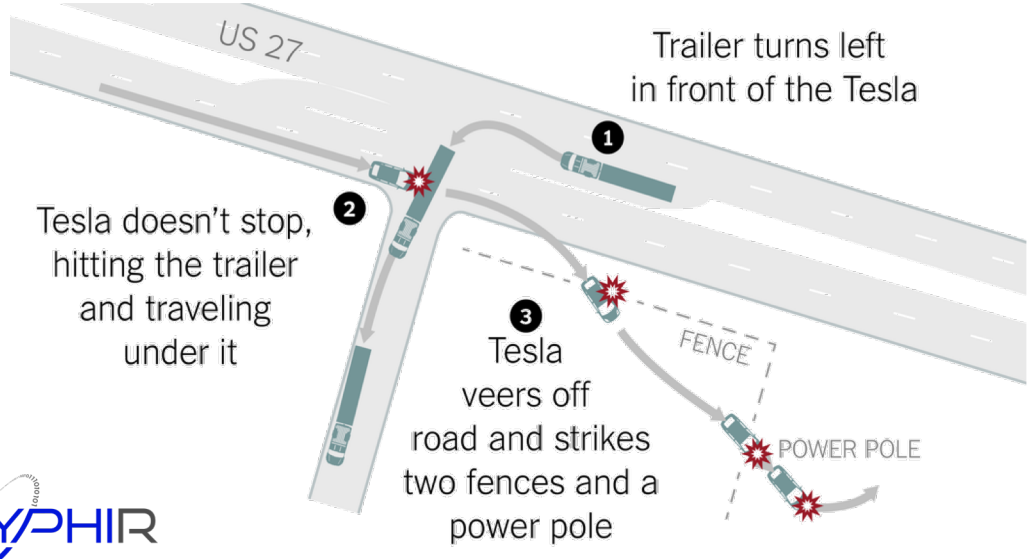
DECYPHIR

[All images from Google image search]

Formal methods are needed because …



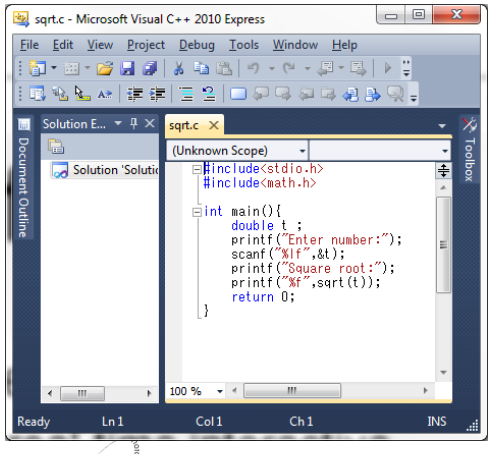
… kind of safety critical.

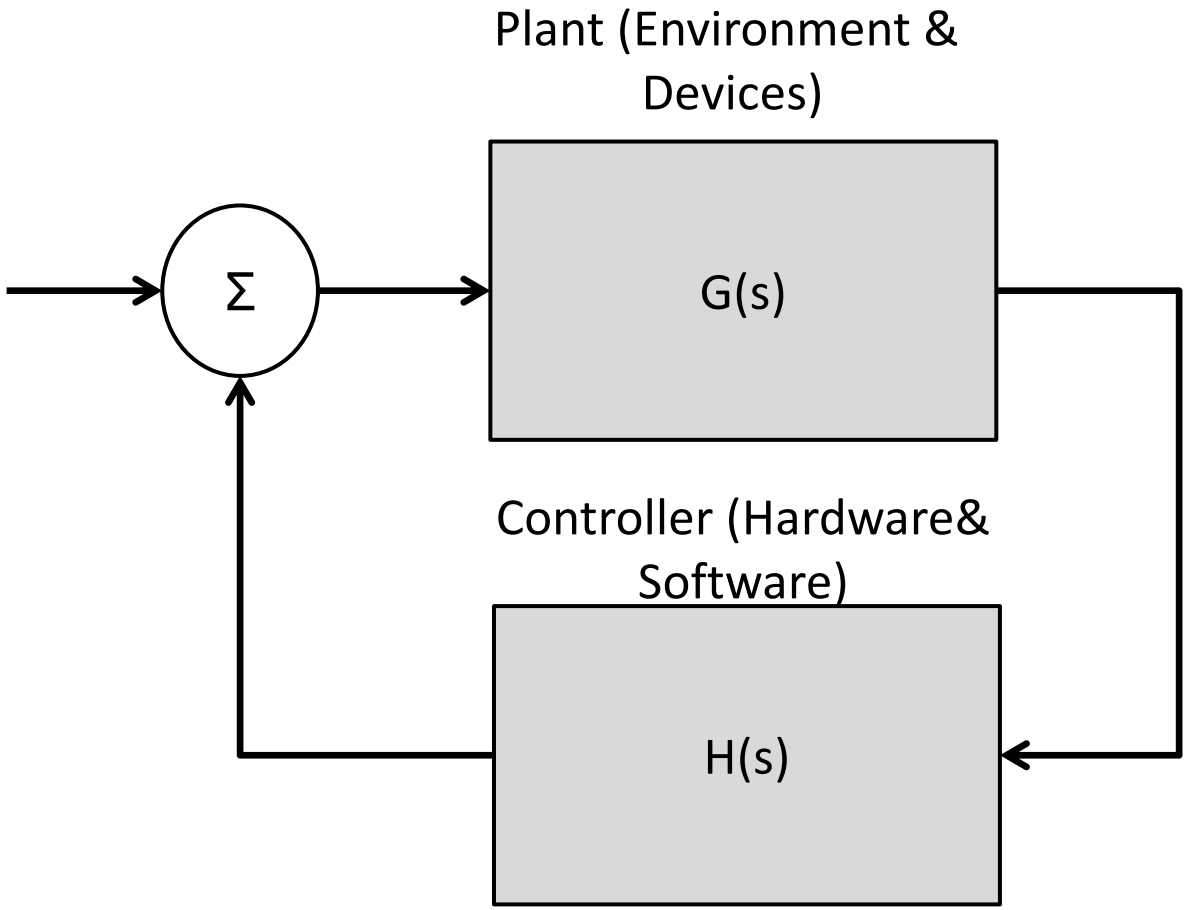


The **FDA** has issued **23 recalls of defective devices** during the first half of 2010, all of which are categorized as "Class I," meaning there is "**reasonable probability** that use of these products will cause serious **adverse health consequences** or **death**."

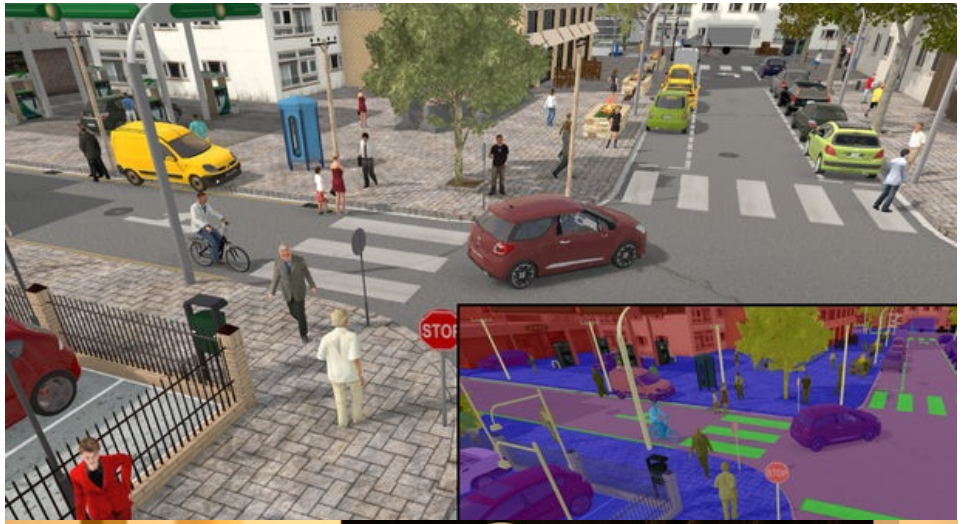
DECYPHIR

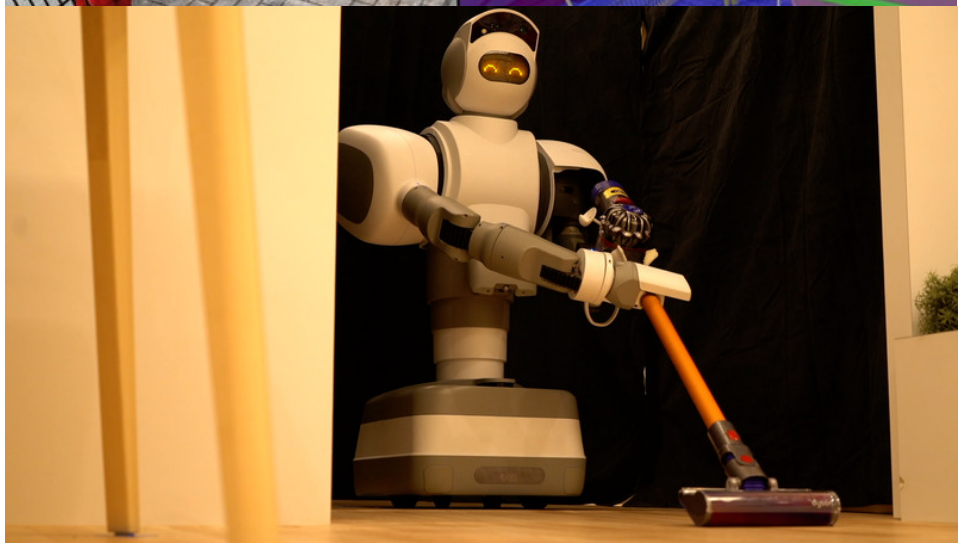
Designing a CPS

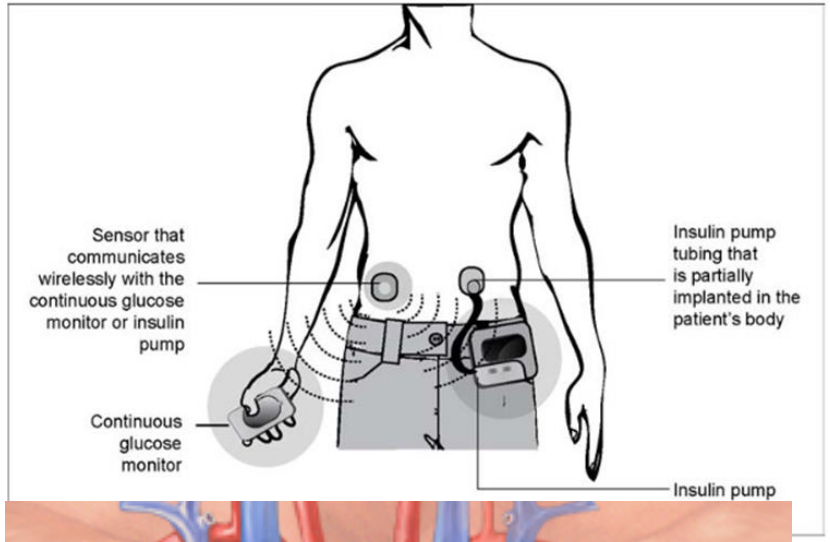


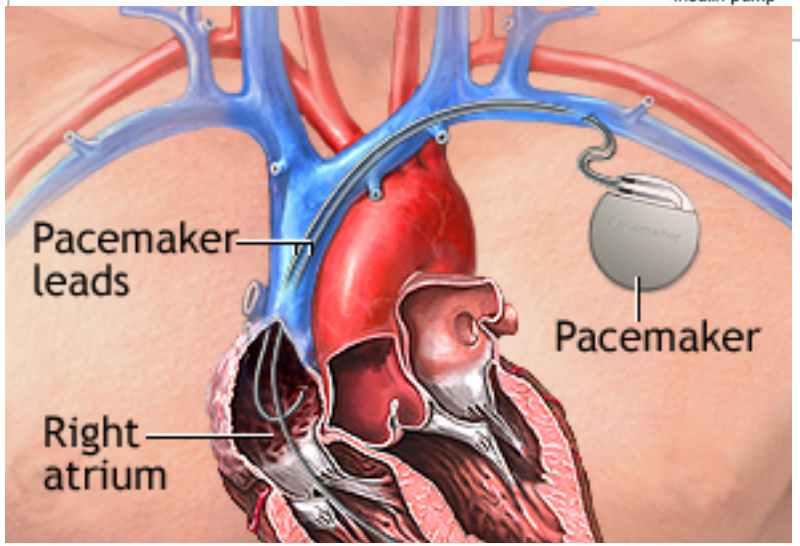


Challenges of Designing CPSs (Plant)



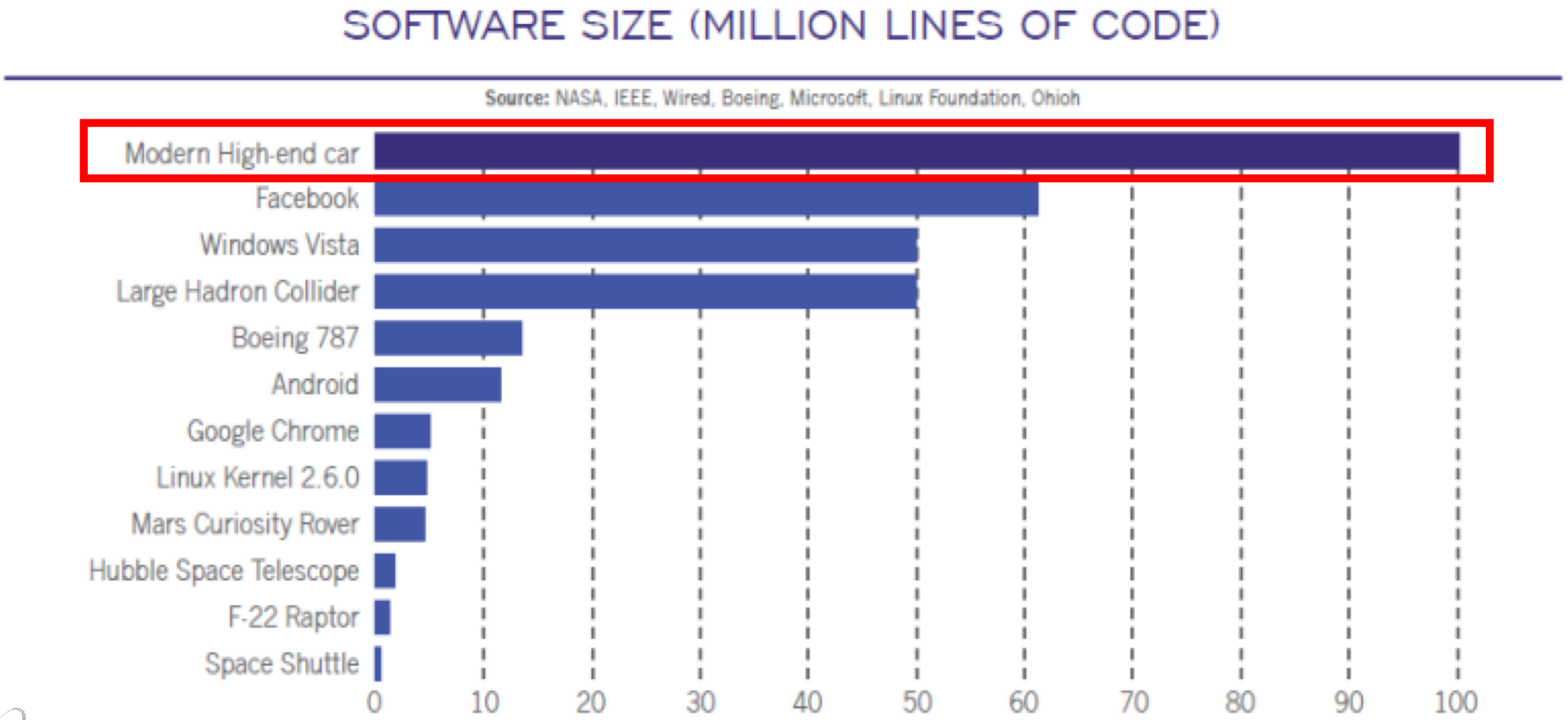






[All images from Google image search]

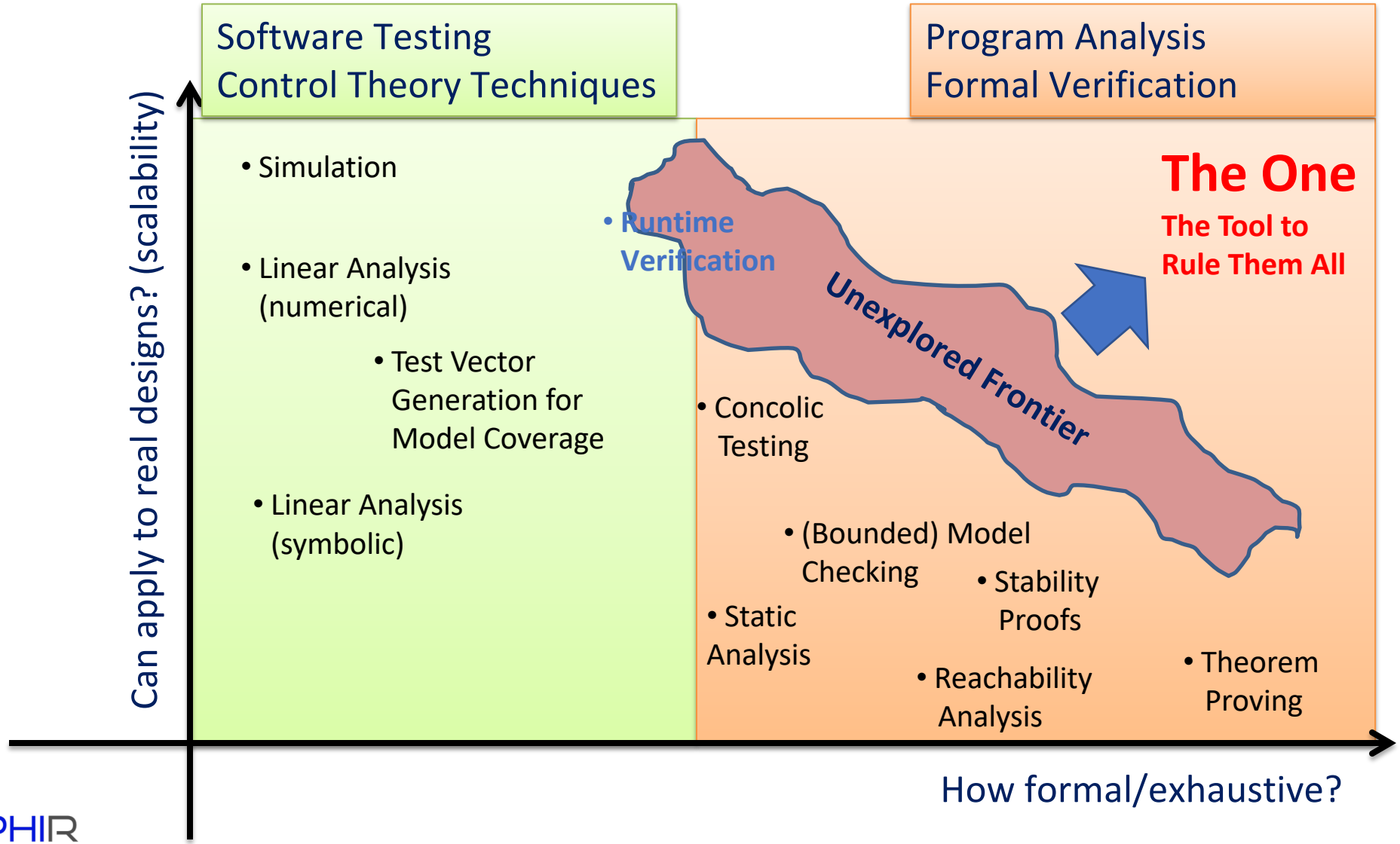
Challenges of Designing a CPS (Software)



Formal Methods in Practice

- Industrial techniques mostly focused on “verifying” the controller
  - Actually means “verifying” embedded C code
  - Actually means **testing** the embedded C code
- Some formal methods being marketed by tool vendors
  - Static analysis: Dead code detection, Divide by zero
  - Property Proving
- Mostly don’t work well for closed-loop systems

Spectrum of Analysis Techniques

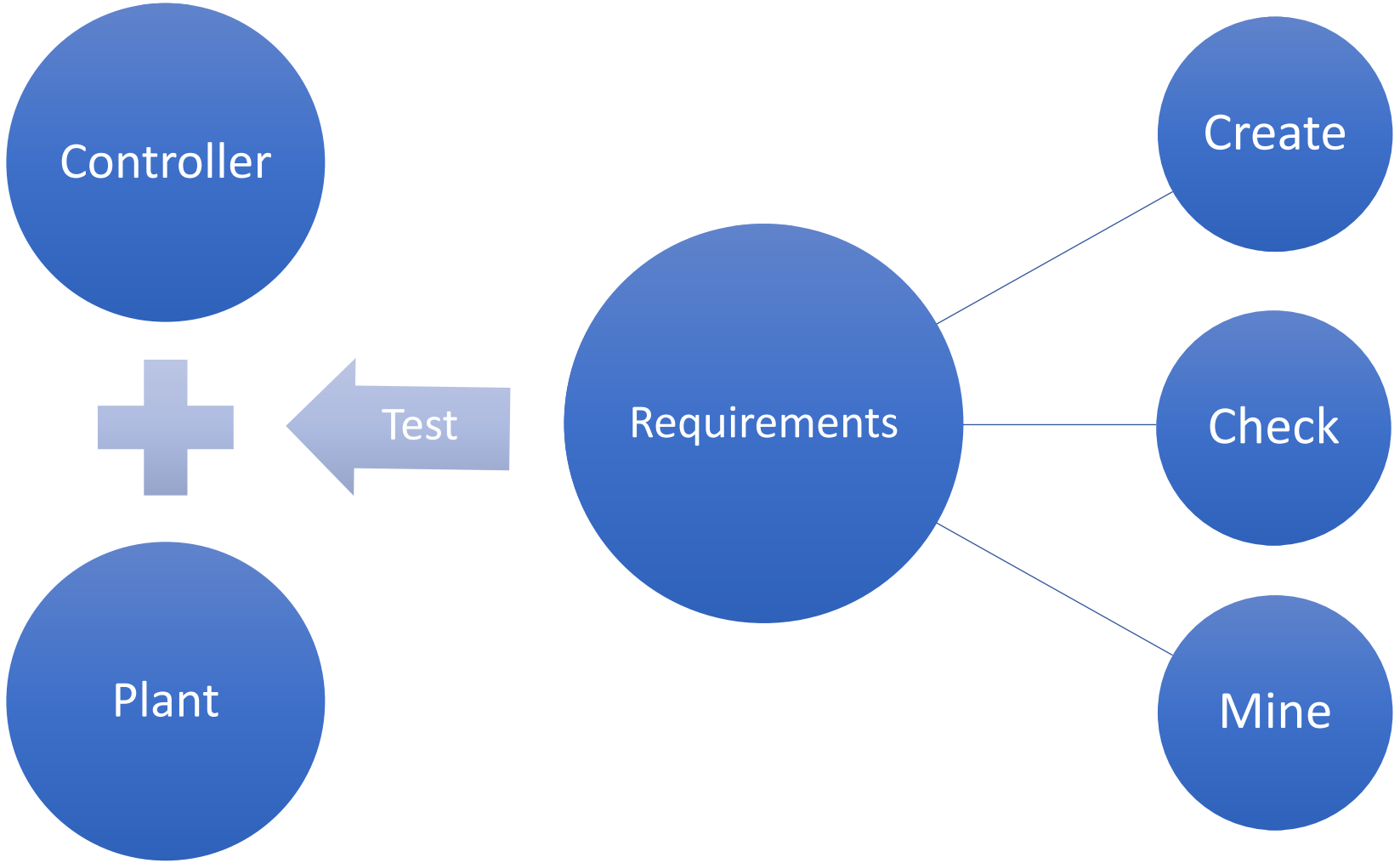


Runtime Verification

- Light-weight verification technique
  - ➢Improves on basic testing by automating the assessment part
- “Semi”-formal methods
  - Formal comes from rigorously defined requirement
  - Not complete: not exhaustive in general
- **Falsification** is a variant that tries to find one trace that don’t satisfy a requirement

Requirement-Driven Design

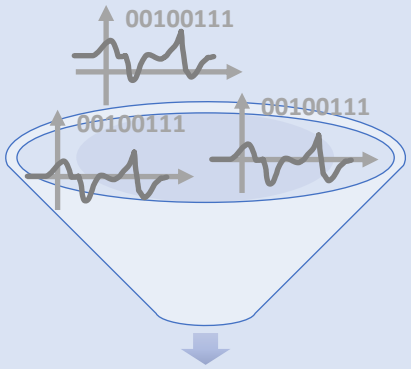
Requirements **formally** capture what it means for a system to operate correctly in its operating environment



Decyphir Breach Platform

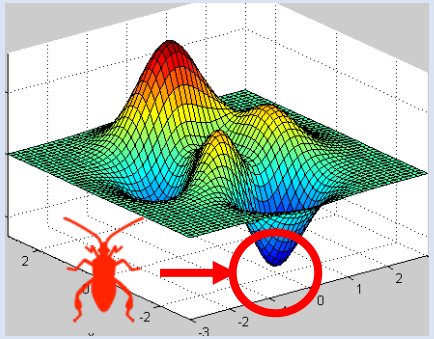
Test Generation

Requirement

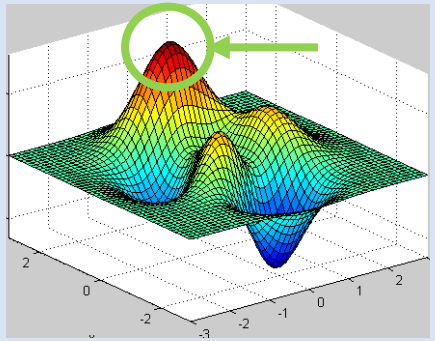


Always rpm[t] < 6000

Falsification



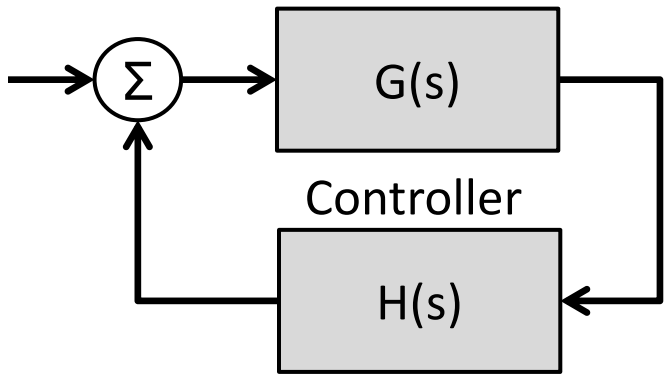
Calibration



https://github.com/decyphir/breach

Model-Based Design

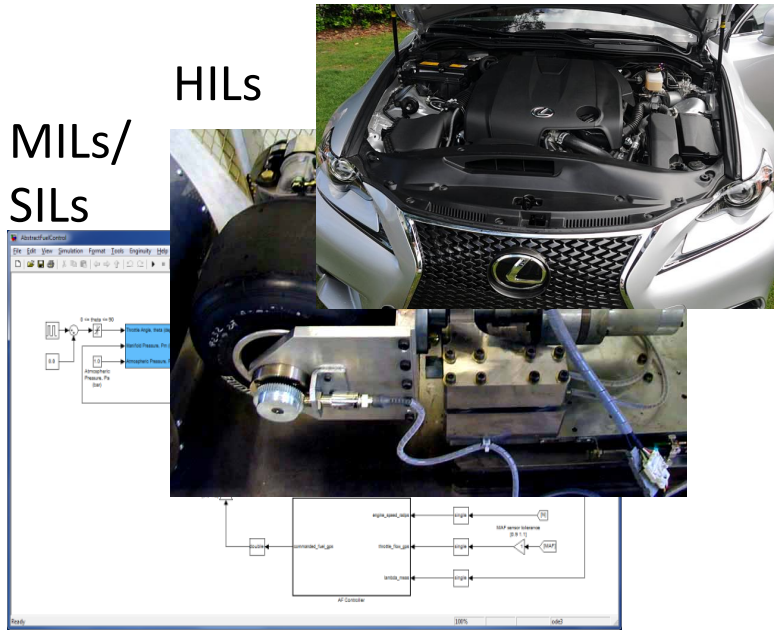
Environment & Devices (Plant)



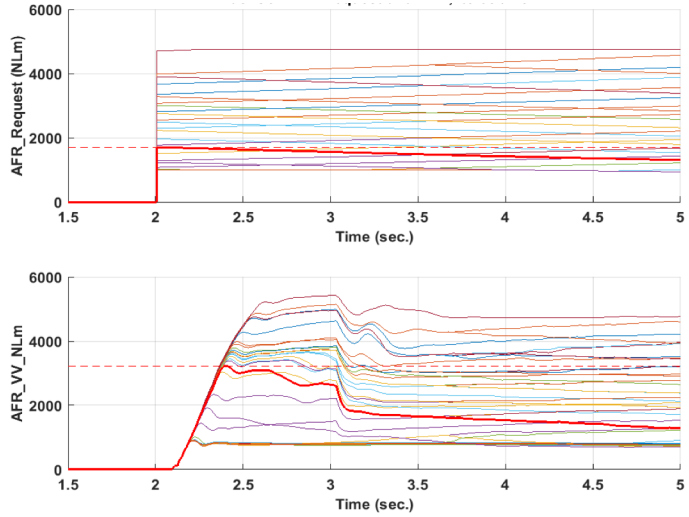
MILs/ SILs

HILs

Prototype Driving Test



Produce data to monitor



MIL/SIL/HIL: **M**odel/**S**oftwate/**H**ardware **I**n the **L**oop

This talk

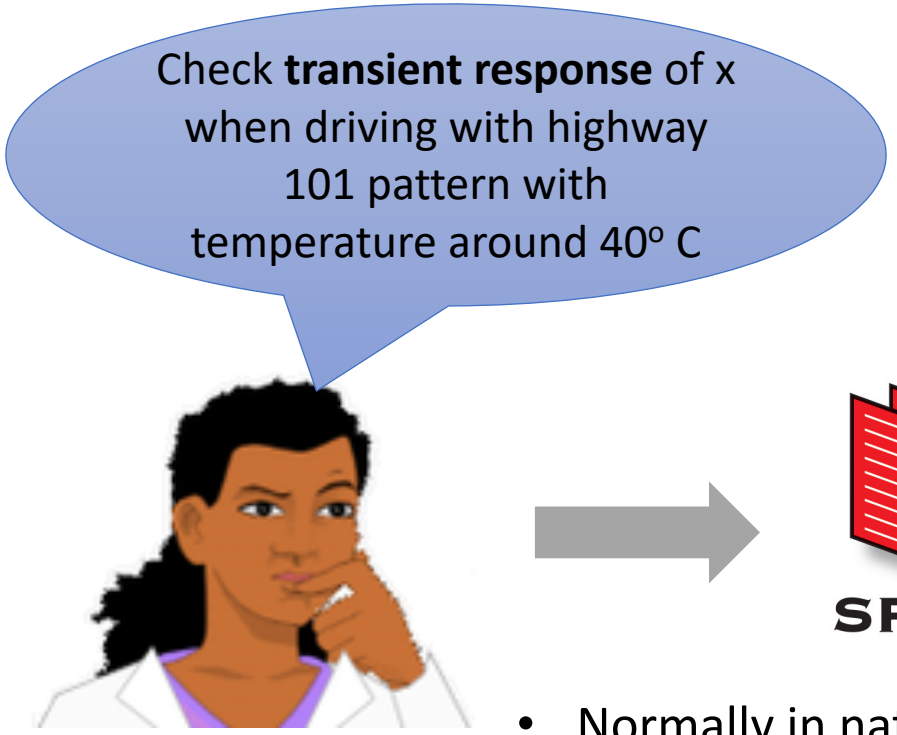
“Traditional” CPS

- E.g., powertrain control, electric drive, battery management system
- Tools: Simulink + Code Generation
- Monitoring **Signals**,i.e Time Series

“Modern” CPS

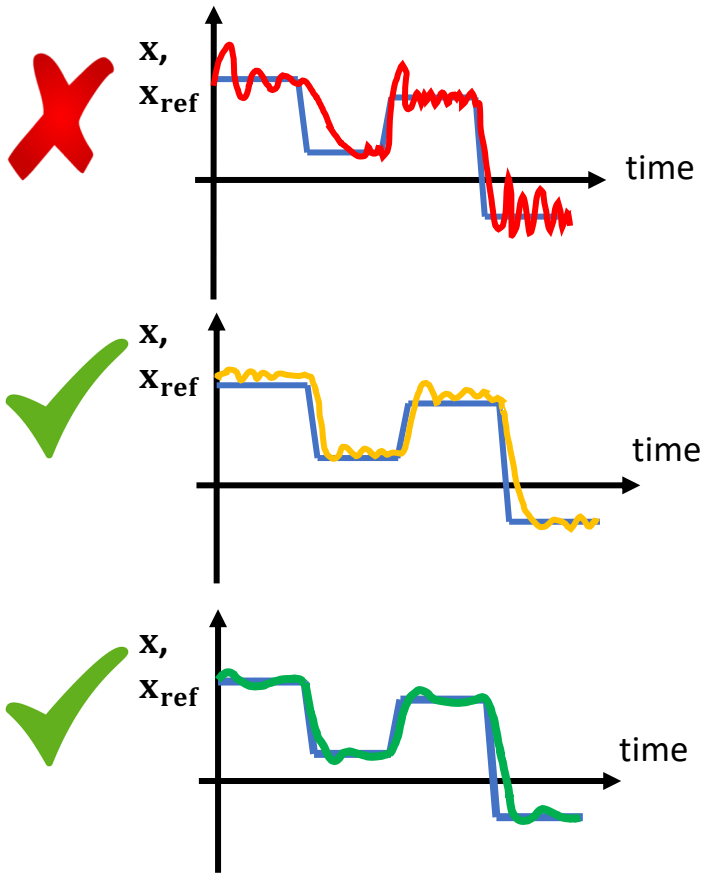
- Multi-Sensors+Machine Learning based perception (e.g., for ADAS/autonomous driving)
- Onboard PC + high-end GPU for DNN, no established toolchain
- Monitoring **the World**

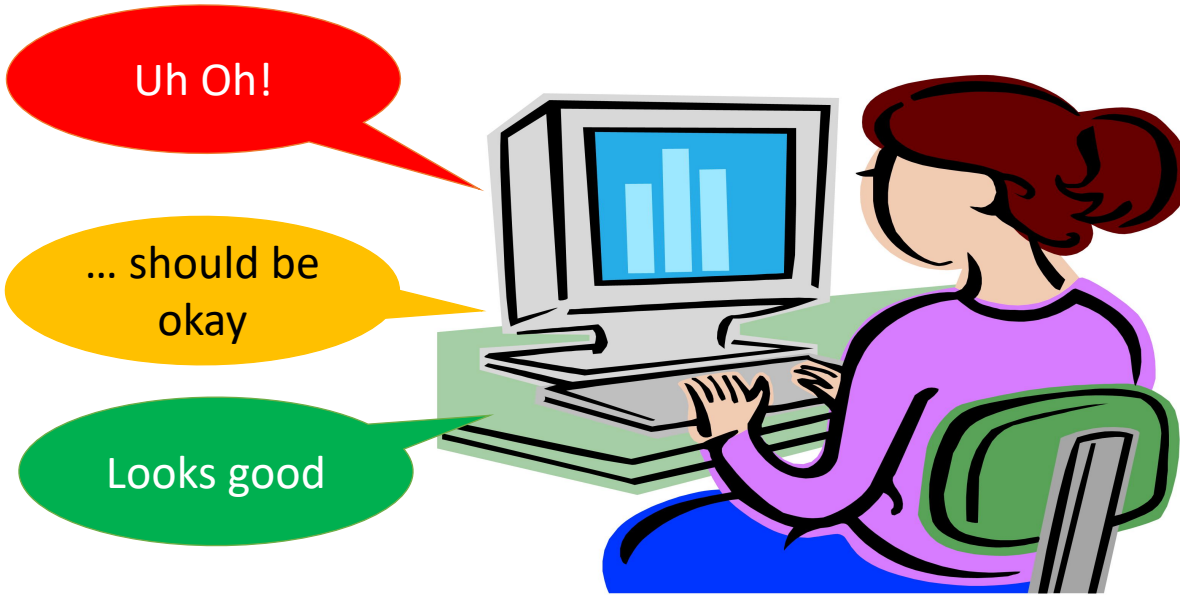
In the Field, How Are High-level Design Specs Checked



- Normally in natural language (ambiguous, error-prone)
- Sometime absent
- If you are LUCKY, they are written in English

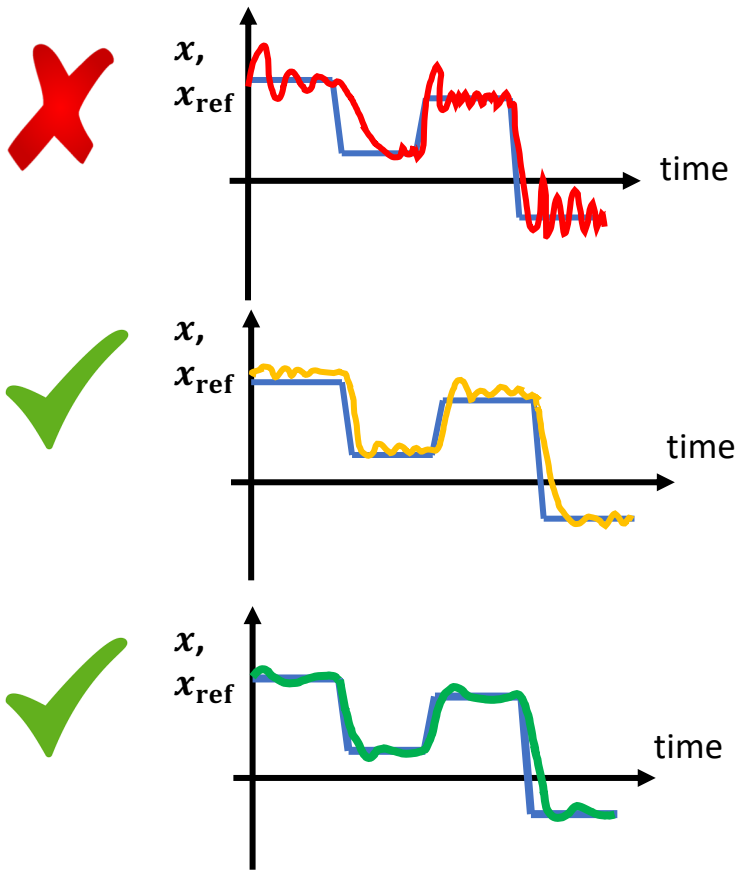
Design Correctness: Relies on Designer Experience

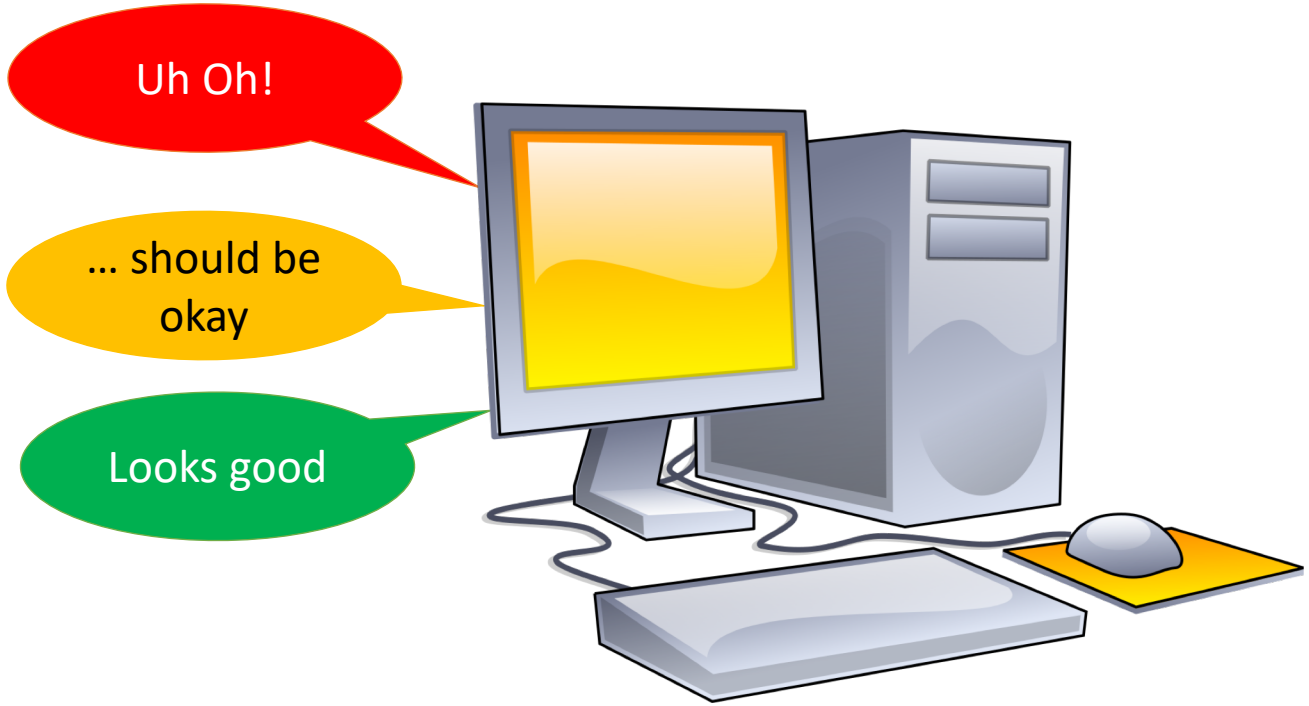




Can we improve this testing process?

Design Correctness: Relies on Machine Checkable Requirements





Can we formalize "*Uh-oh, should be okay, looks good, weird, clearly wrong, fuzzy*?"

Temporal Logic (In A Nutshell)

- Temporal logics specify properties over time

  E.g. For the next 3 days, the highest temperature will be below 75 degree and the lowest temperature will be above 60 degree

  Key ingredients

  1. Propositions

  "the highest temperature will be below 75 degree" $T < 75$

  "the lowest temperature will be above 60 degree " $T > 60$

  2. Propositional logic: propositions connected by Boolean operators, such as Not (¬), And(∧), Or(∨), and Imply (→)

$$(T < 75) \wedge (T > 60)$$

Temporal Logic (In A Nutshell)

- Temporal logics specify properties over time

  E.g. For the next 3 days, the highest temperature will be below 75 degree and the lowest temperature will be above 60 degree

  Key ingredients

  1. Propositions

  "the highest temperature will be below 75 degree" $T < 75$

  "the lowest temperature will be above 60 degree " $T > 60$

  2. Propositional logic: propositions connected by Boolean operators, such as Not (¬), And(∧), Or(∨), and Imply (→)

$$(T < 75) \wedge (T > 60)$$

LTL - Linear Temporal Logic*

Using a to represent the state $(T < 75) \wedge (T > 60)$

LTL adds temporal operators

a	“a is true now”
X a	“a is true in the neXt time step”
Fa	“a will be true in the Future (eventually) ”
Ga	“a will be Globally (always) true in the future”
a U b	“a will hold true Until b becomes true”

* Pnueli, Amir. "The temporal logic of programs." *Foundations of Computer Science, 1977., 18th Annual Symposium on*. IEEE, 1977.

Signal Temporal Logic (STL)

- Recall the Example:

  For the next 3 days, the highest temperature will be below 75 degree and the lowest temperature will be above 60 degree

- LTL: using a represents the state $(T < 75) \wedge (T > 60)$

  $\mathrm{X}\ a \wedge \mathrm{X}\ \mathrm{X}\ a \wedge \mathrm{X}\ \mathrm{X}\ \mathrm{X}\ a$

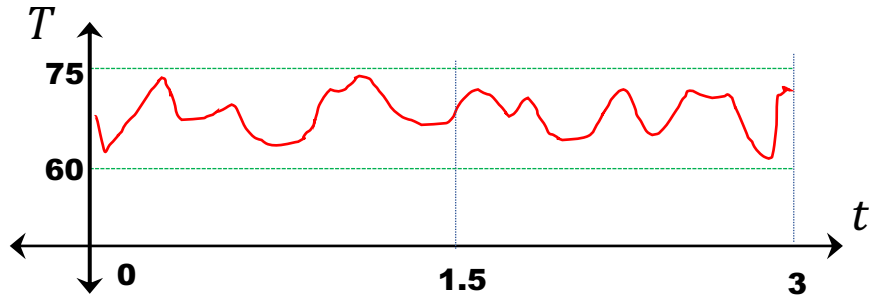
- STL: extended LTL by adding dense time and having predicates over real values

  $\mathrm{G}_{[0,3]}(T(t) < 75 \wedge T(t) > 60)$

STL Examples

$$\mathbf{G}_{[0,3]}\,(T(t) < 75 \wedge T(t) > 60)$$

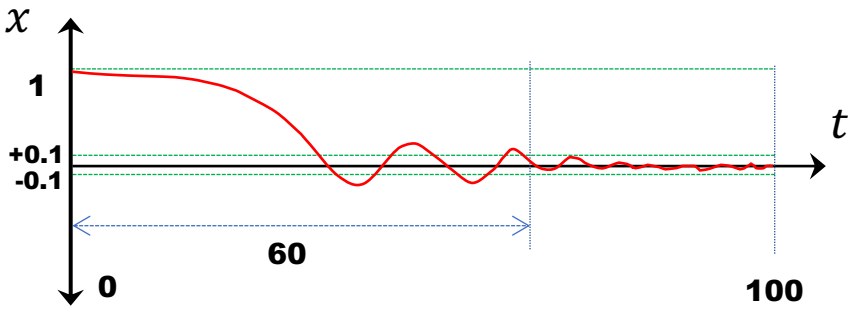
Always between time 0 and 3 days



$$\mathbf{F}_{[20,60]}(\mathbf{G}\,(|x(t)| < 0.1)\,)$$

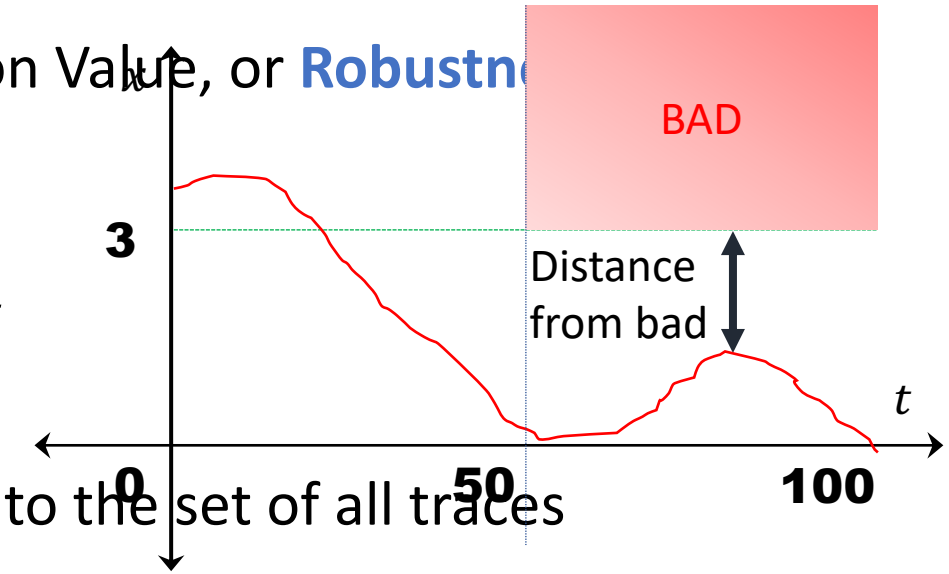
Eventually at **some time** *t* between time 20 and 60

From that time *t,* always till the end of the signal trace



Beyond Boolean Satisfaction

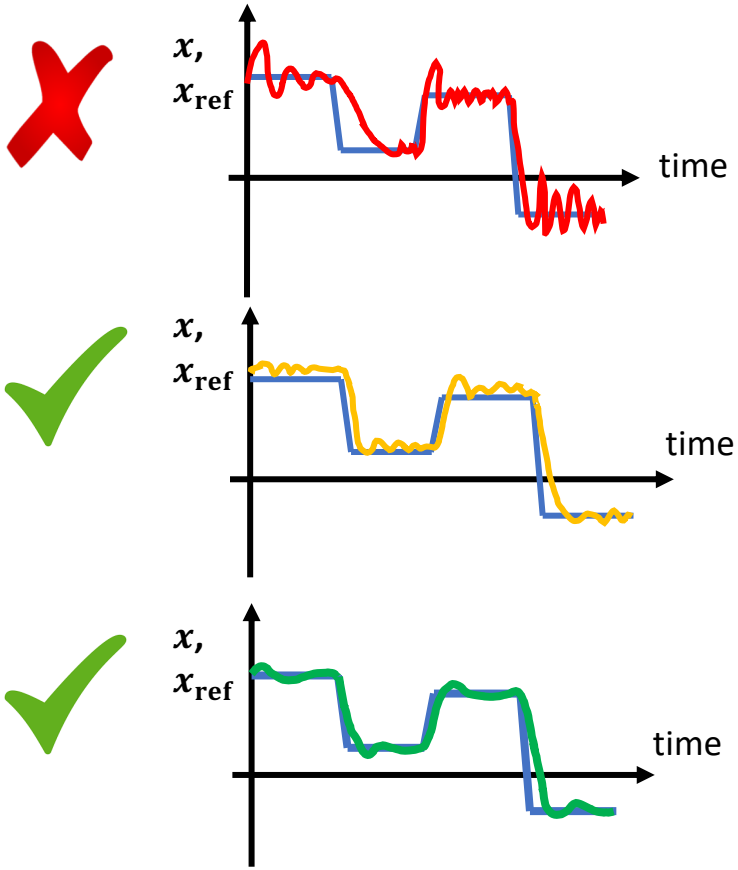
- STL has quantitative[1,2] semantics (Robust Satisfaction Value, or **Robustn**
  - for a given trace $x(t)$,
  - and STL formula φ,
  - maps φ, $x(t)$ to some real value for each time t
- Intuition:
  - Compute "signed distance" of the given trace x to the set of all traces satisfying φ
  - Distance ≥ 0 : $x \in$ set of traces satisfying φ
  - Distance < 0 : $x \notin$ set of traces satisfying φ
  - Going from positive to negative = going towards violation of φ

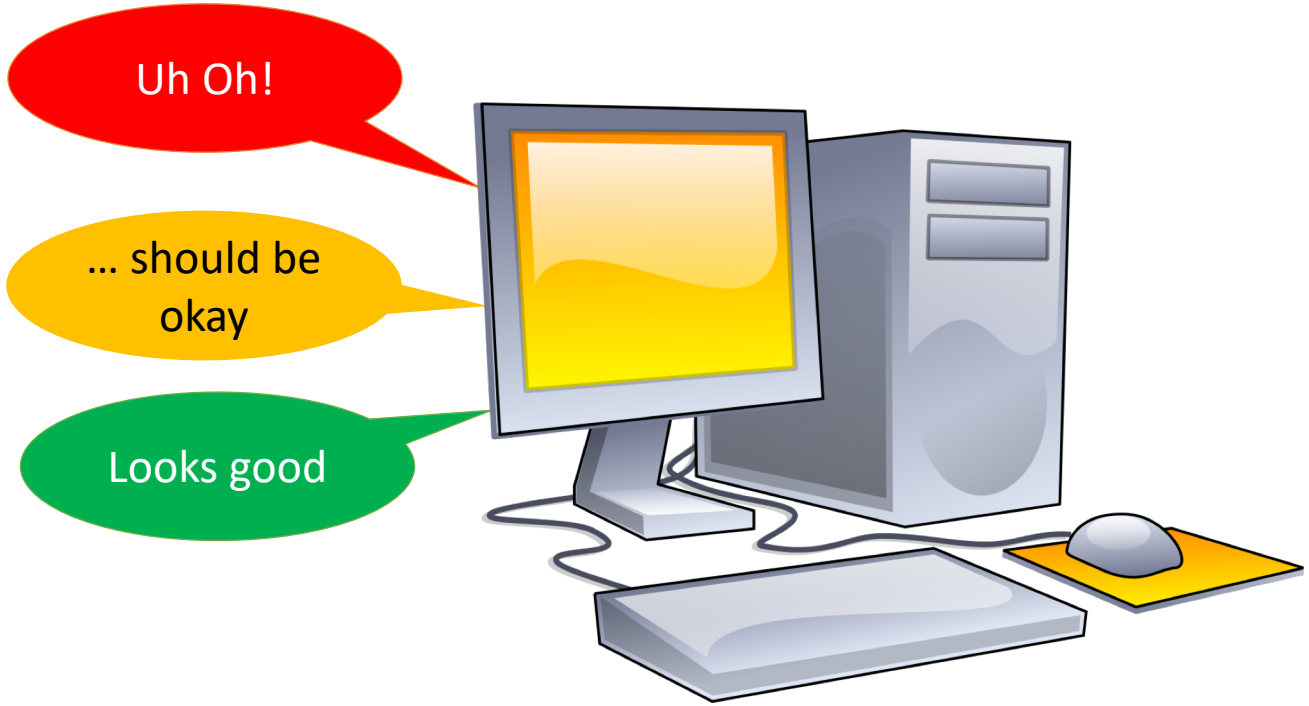


$\mathrm{G}_{[50,100]}(x(t) < 3)$

[1] A. Donzé, and O. Maler. *Robust satisfaction of temporal logic over real-valued signals*. FORMATS 2010
[2] G. E. Fainekos, G. J. Pappas, *Robustness of Temporal Logic Specifications*, FATES/RV 2006

Design Correctness: Relies on Machine Checkable Requirements



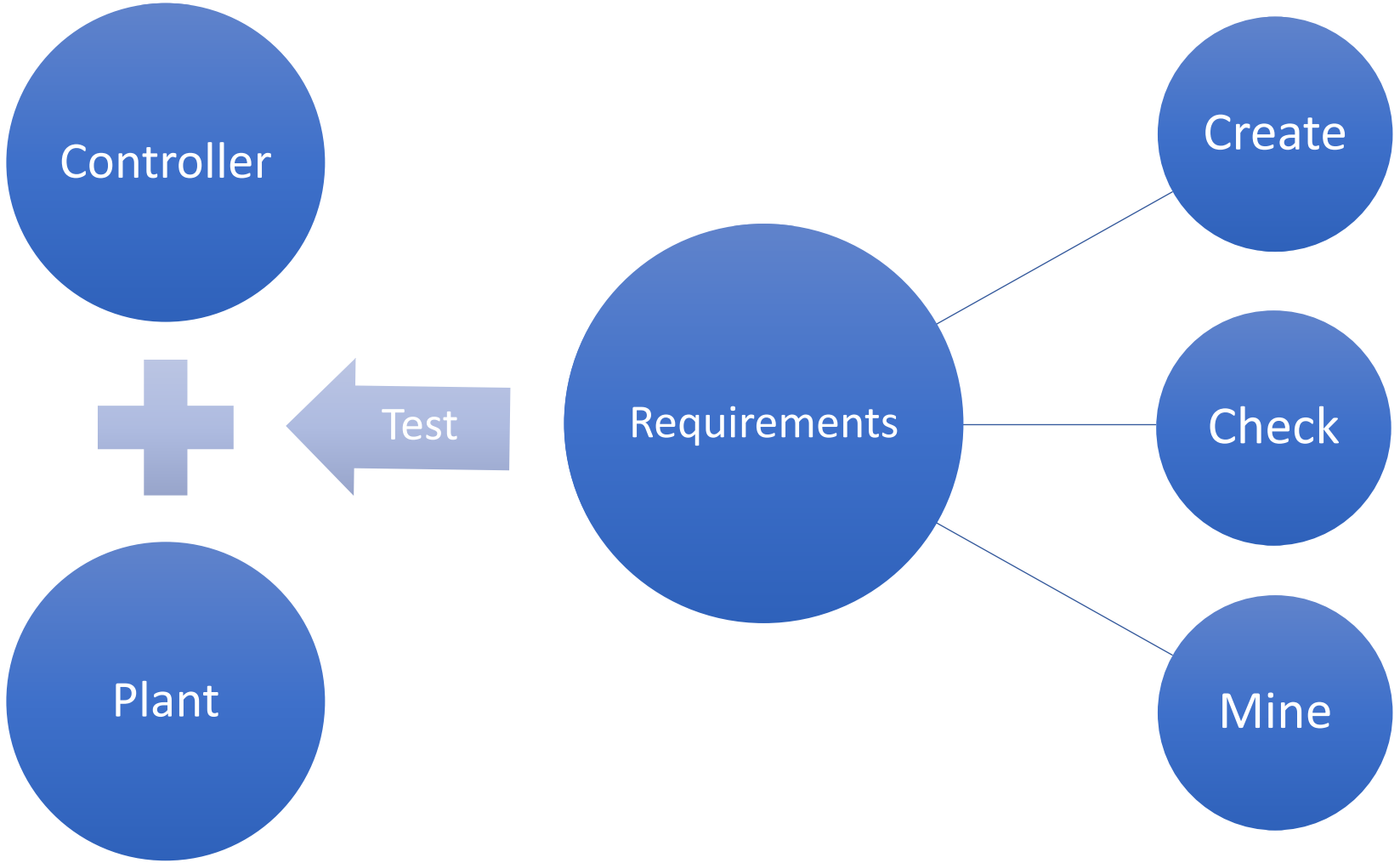


Can we formalize *"Uh-oh, should be okay, looks good, weird, clearly wrong, fuzzy?"*

$$\varphi_{\text{transientOK}} \equiv \mathrm{G}_{[0,T]}\big(\text{step} \Rightarrow \mathrm{G}_{[0,2]}(|x - x_{\text{ref}}| < 0.05x_{\text{ref}})\big)$$

Requirement-Driven Design

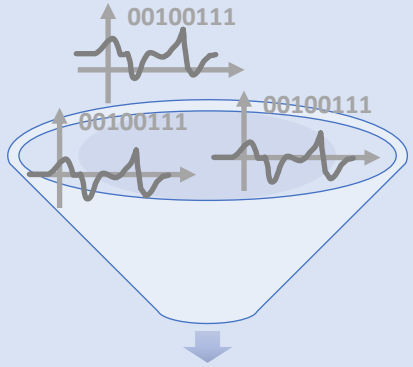
Requirements **formally** capture what it means for a system to operate correctly in its operating environment



Decyphir Breach Platform

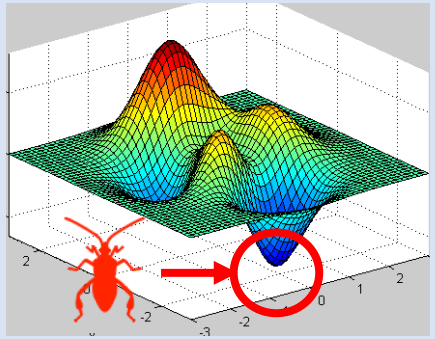
Test Generation

Requirement

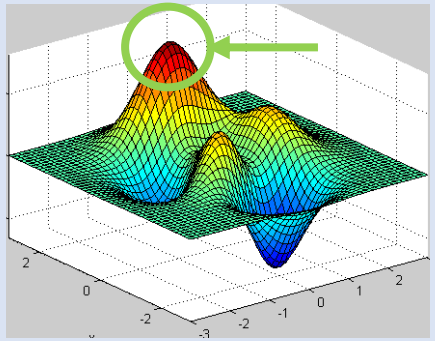


Always rpm[t] < 6000

Falsification



Calibration

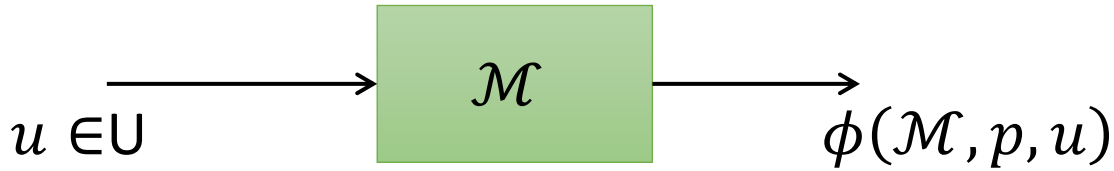


https://github.com/decyphir/breach

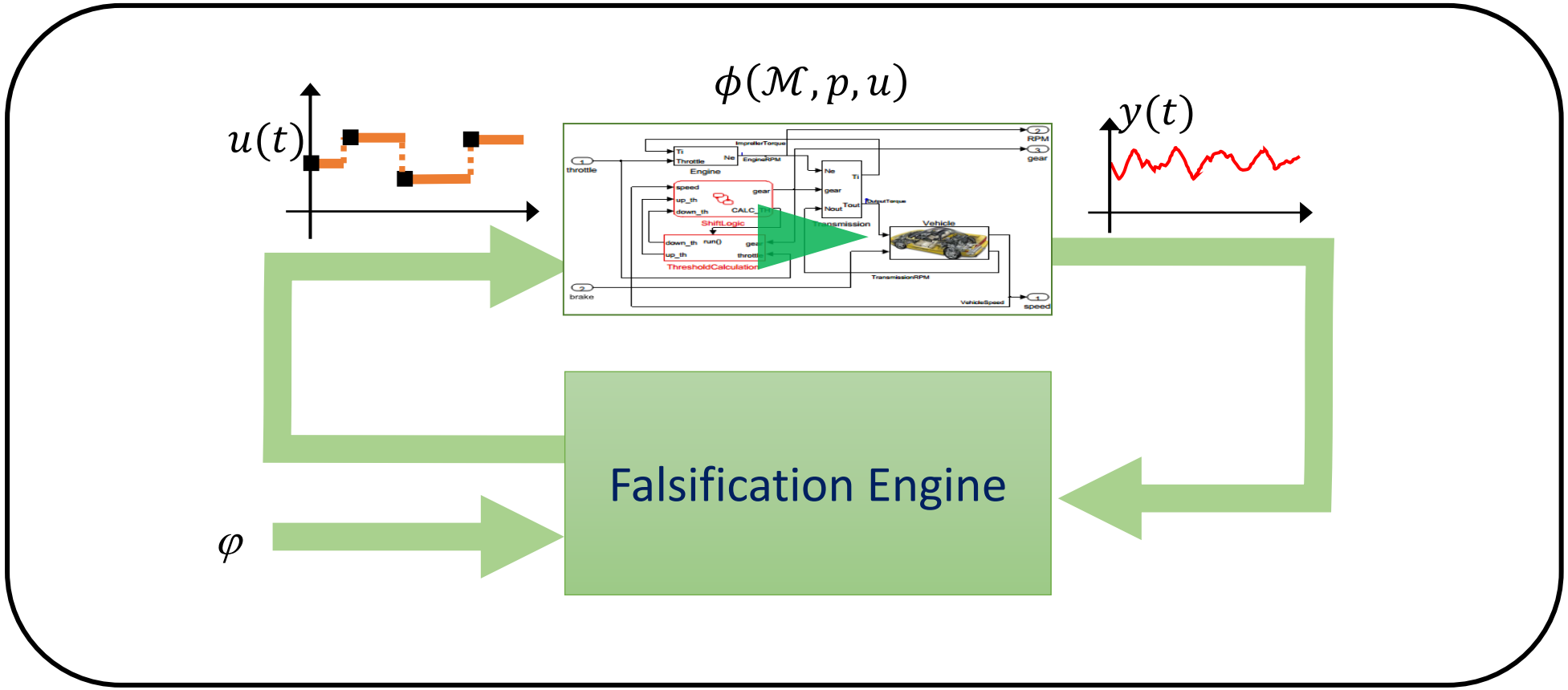
Requirement Falsification

Definition (Falsification):

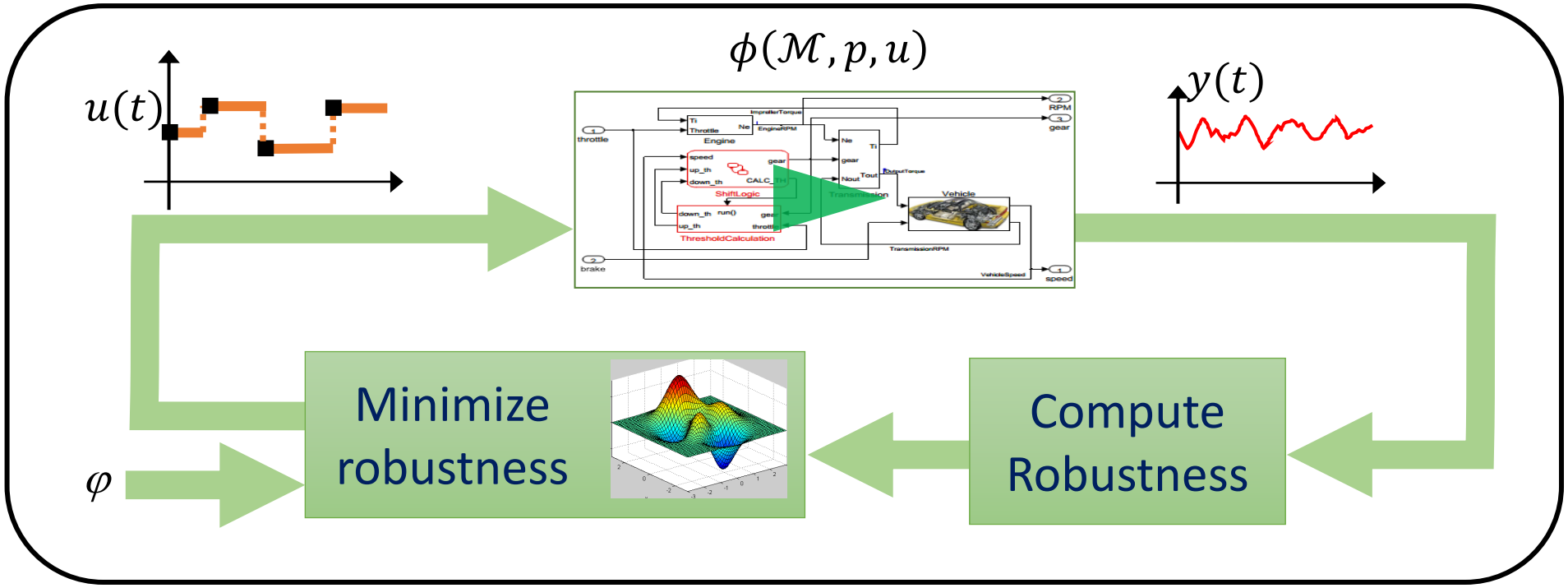
Find parameters $p \in P$ and input $u \in U$ such that behaviors $\phi(\mathcal{M}, p, u)$ do **NOT** satisfy requirements φ (i.e., $\phi(\mathcal{M}, p, u) \not\models \varphi$)



Falsification Procedure



Falsification by optimization

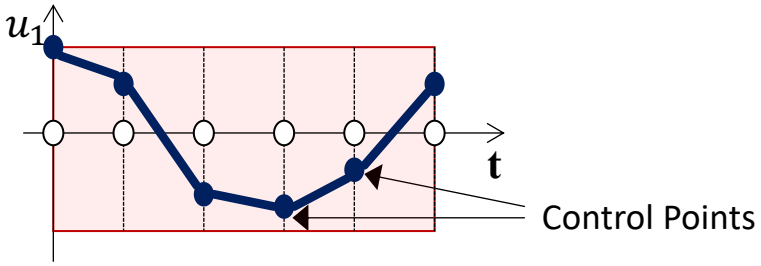


- Use robustness as a cost function to minimize with Black-box/Global Optimizers
- Breach supports flexible modularity: input generators, simulators, robustness evaluation (cost function), and optimization engines

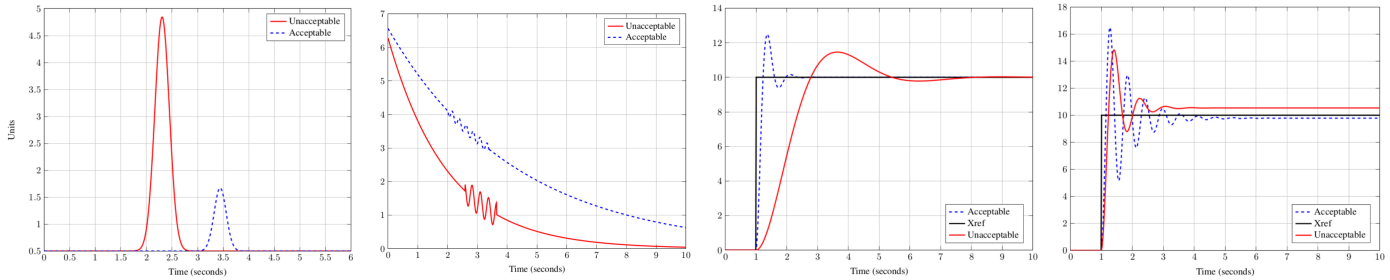
Falsification with Parameterized Inputs

- Breach supports flexible control point parameterization
- Breach also uses ST-Lib to alleviate designers burden to specify intended design behaviors using a signal template

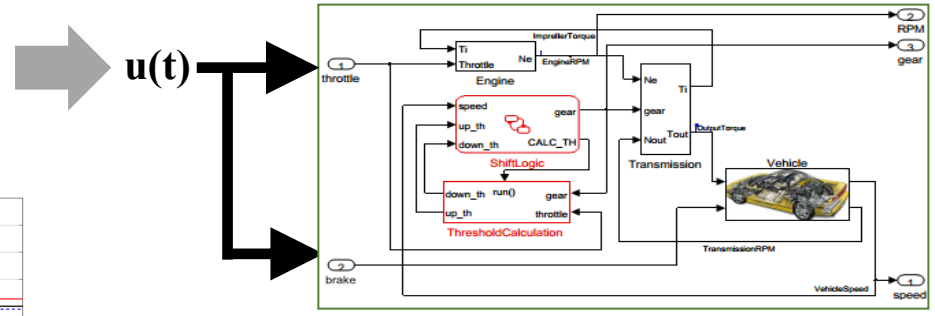
(Inputs also include tunable parameters in both plant and controller)



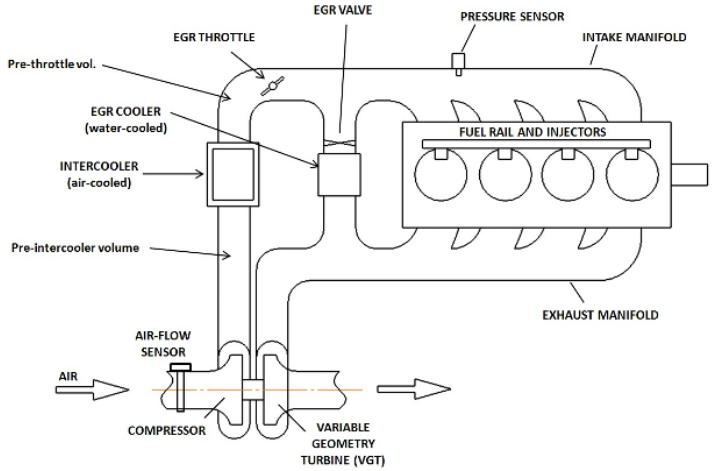
ST-Lib

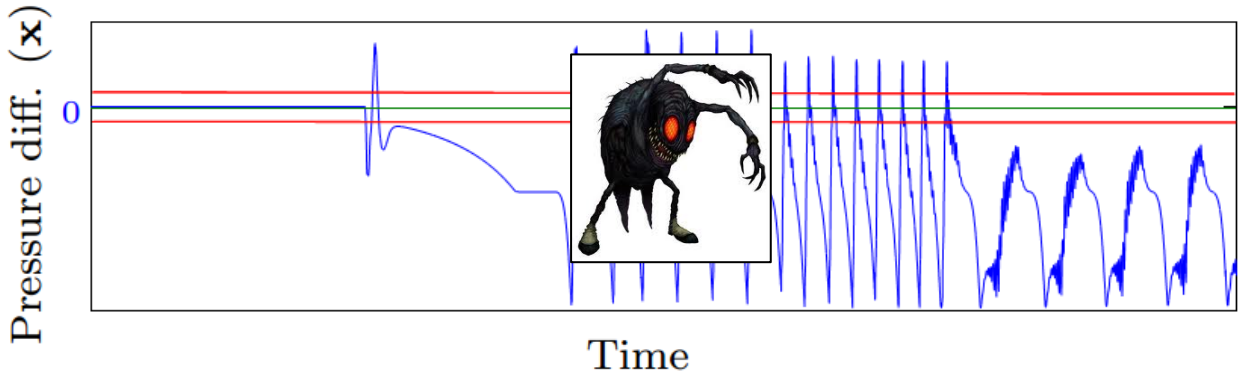


Spikes **Ringing/Hunting** **Rise Time** **Settling Time, Overshoot** ….



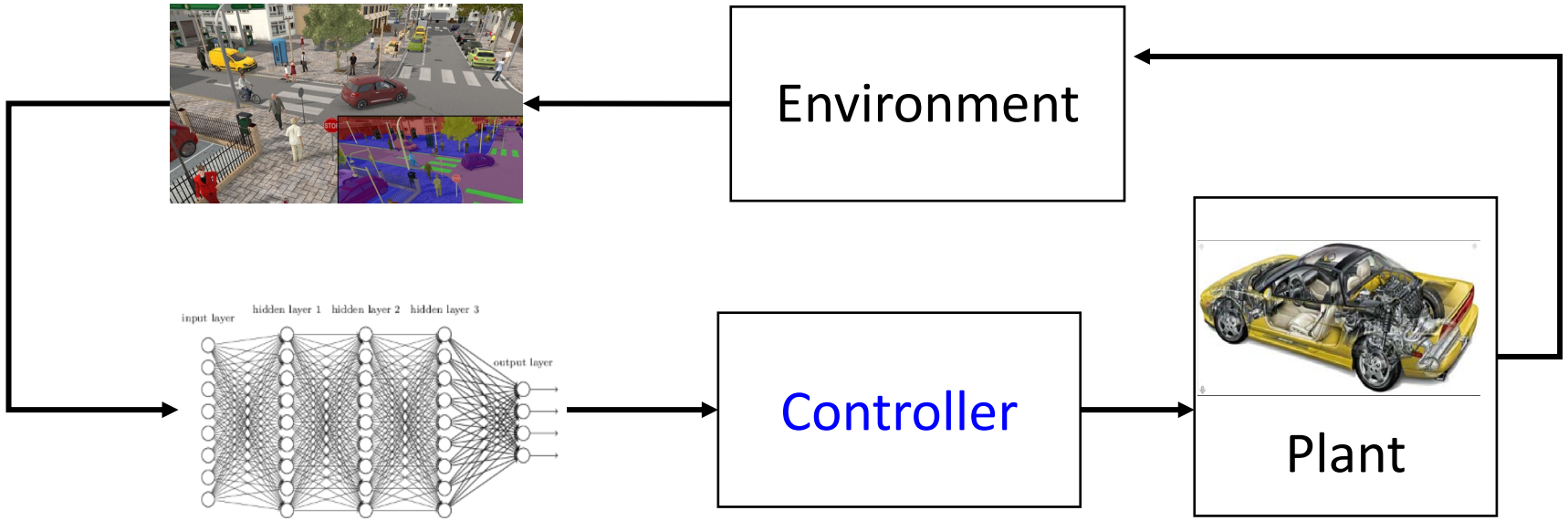
In Practice





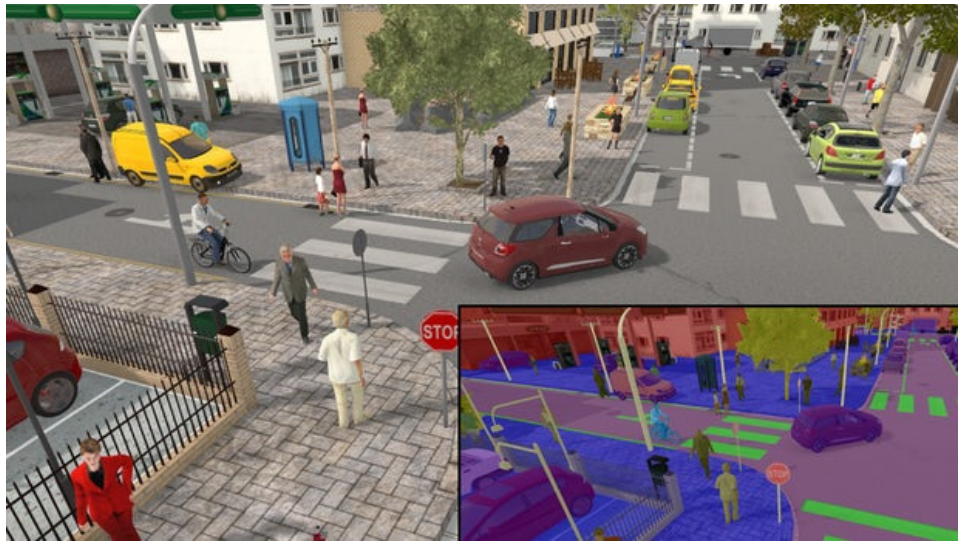
Overshoot bug: (previously found during the vehicle drive testing) reproduced and localized using MILs in a couple of hours

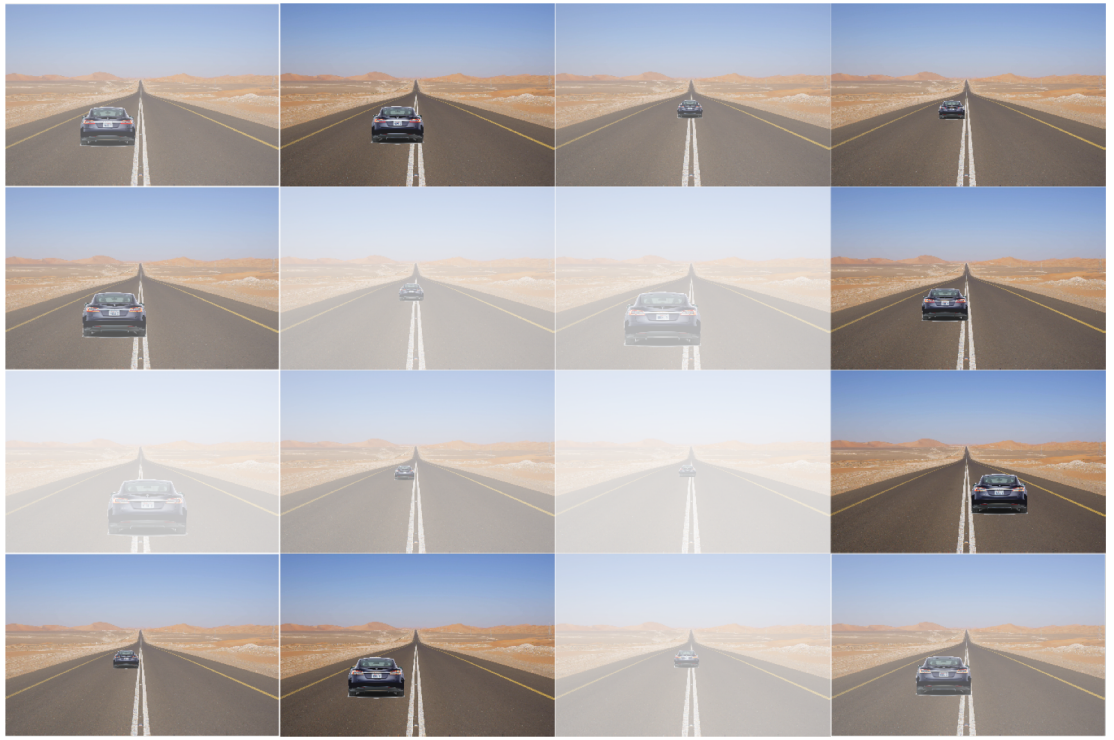
Modern CPS: ADAS and Autonomous Driving



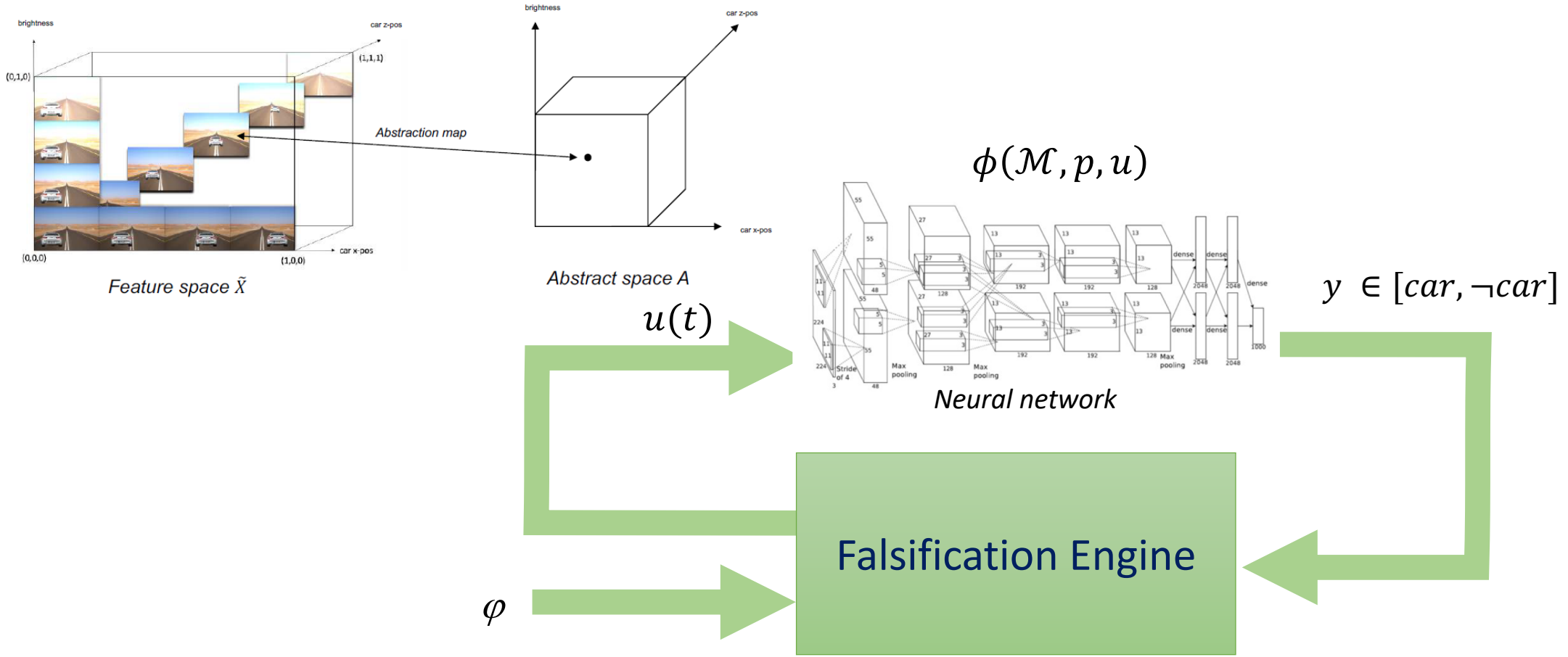
Machine Learning Based Perception

What Are We Looking At?





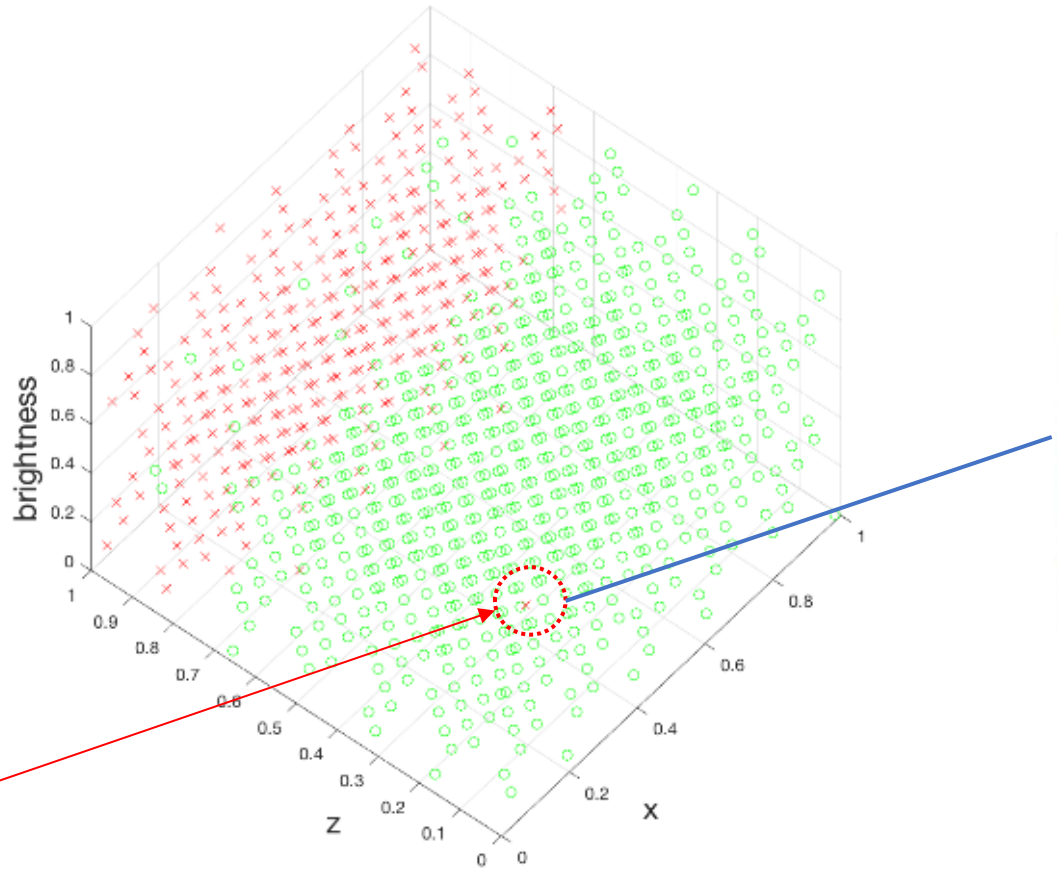
Finding Adversarial Inputs of DNNs



T. Dreossi, A. Donzé, S. A. Seshia, Compositional Falsification of Cyber-Physical Systems with Machine Learning Components, (NFM 2017)

Case Study with DNN

Inception-v3 *Neural Network* *(pre-trained on ImageNet using TensorFlow)*

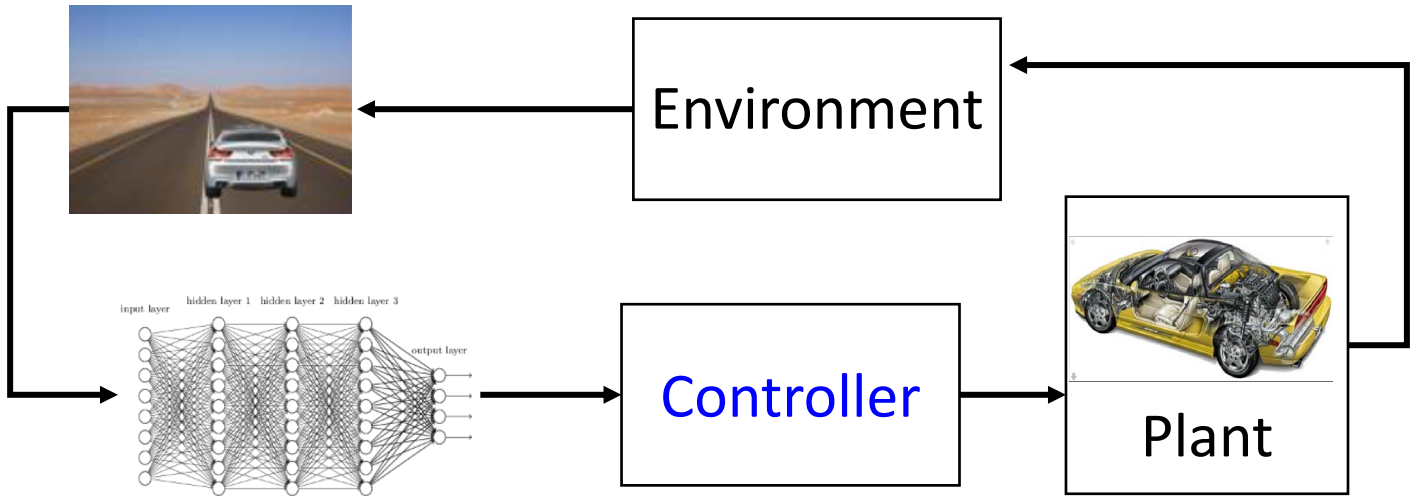


Corner case Image

But this one may be a real hazard!

T. Dreossi, A. Donzé, S. A. Seshia, Compositional Falsification of Cyber-Physical Systems with Machine Learning Components, (NFM 2017)

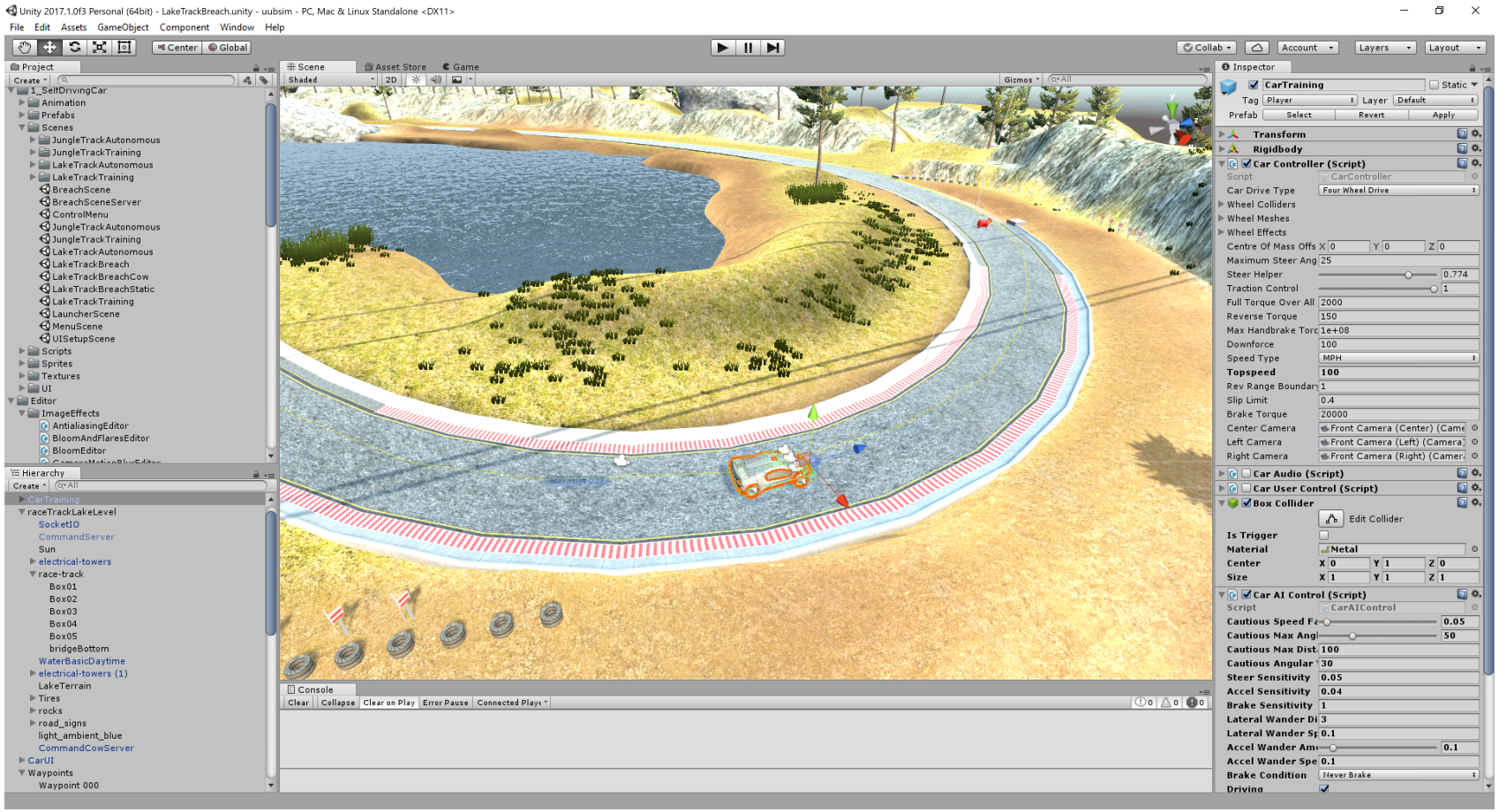
Closing the Loop ?



- Signals are now streams of images (frames)
- Traditional CPS design tool (e.g., Simulink) not optimized for simulation with frame analysis in the loop

➢ Need for specialized simulation engine – e.g., Unity

Unity Udacity Simulator

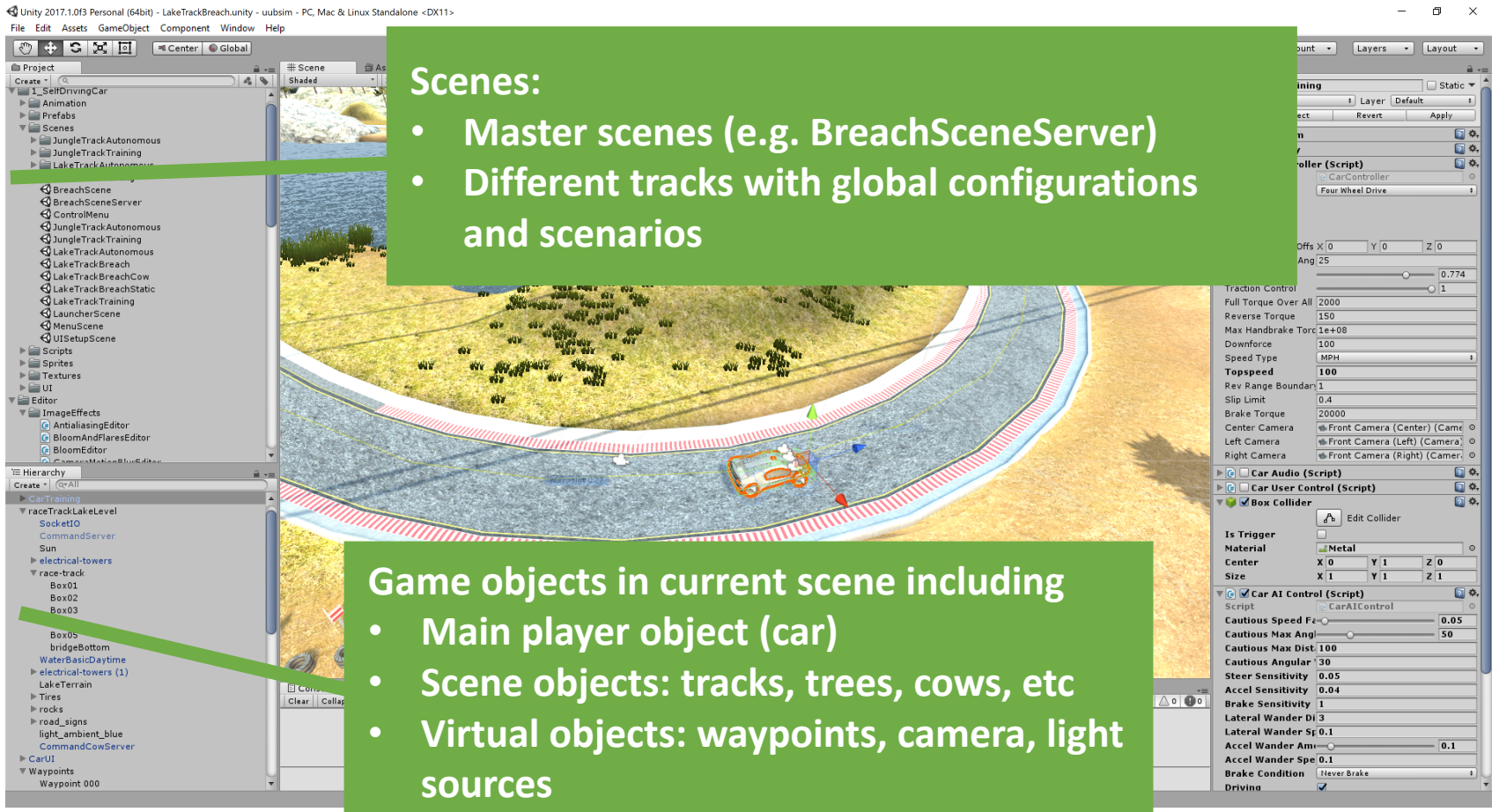


https://eu.udacity.com/course/self-driving-car-engineer-nanodegree--nd013

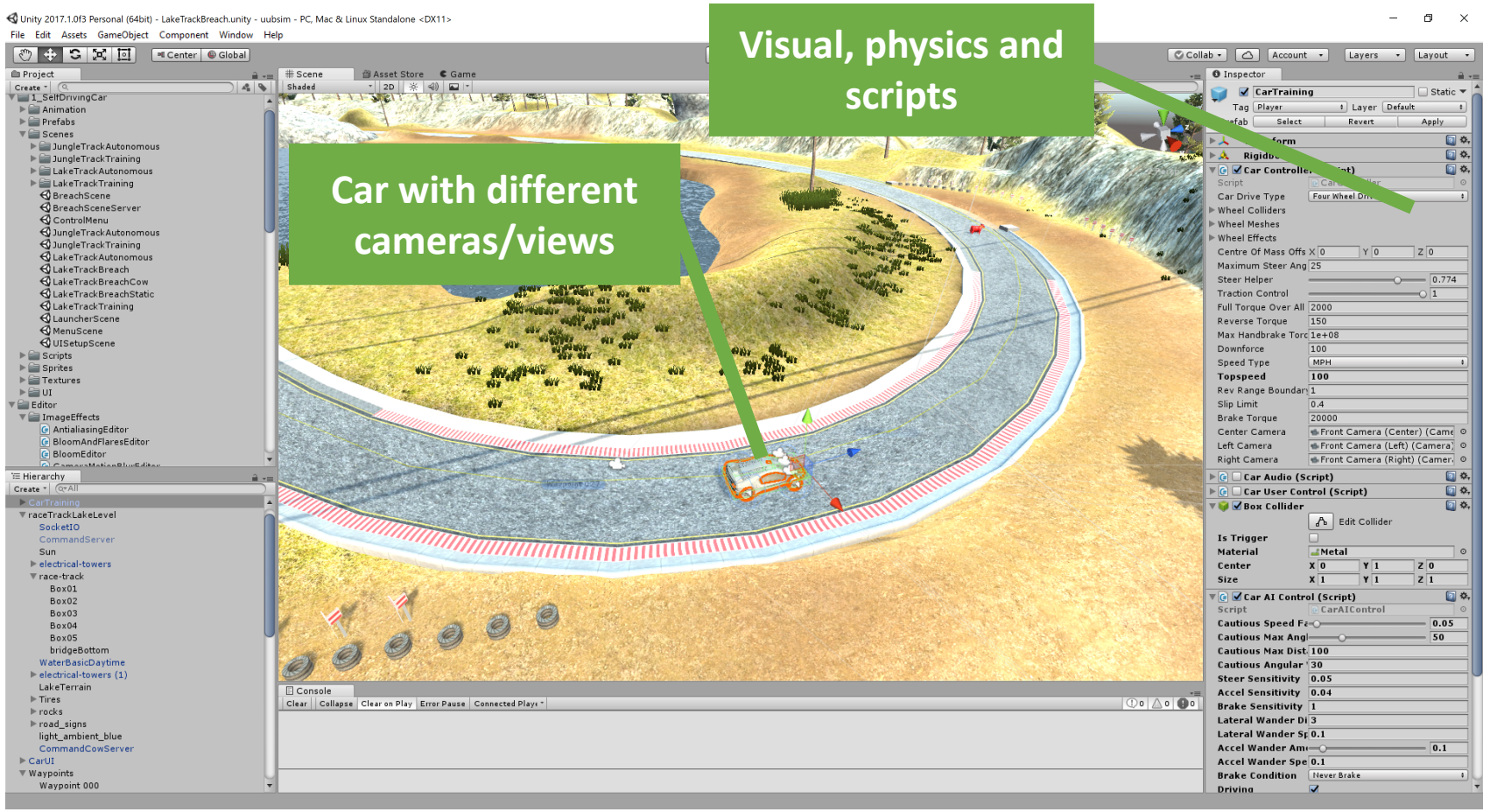
Unity Game Engine – Quick Overview

- A **game** is a collection of **scenes**
- A **scene** is a collection of **game objects (objects)**
- Objects have **visual attributes**, **physics**, and **behavioral scripts**
- The **Unity Engine** is **a hybrid dynamical system simulator** which
  - Simulates the behaviors of all objects based on their physics and scripts in real-time (adjustable)
  - **Is optimized to render as many frames as possible per second**

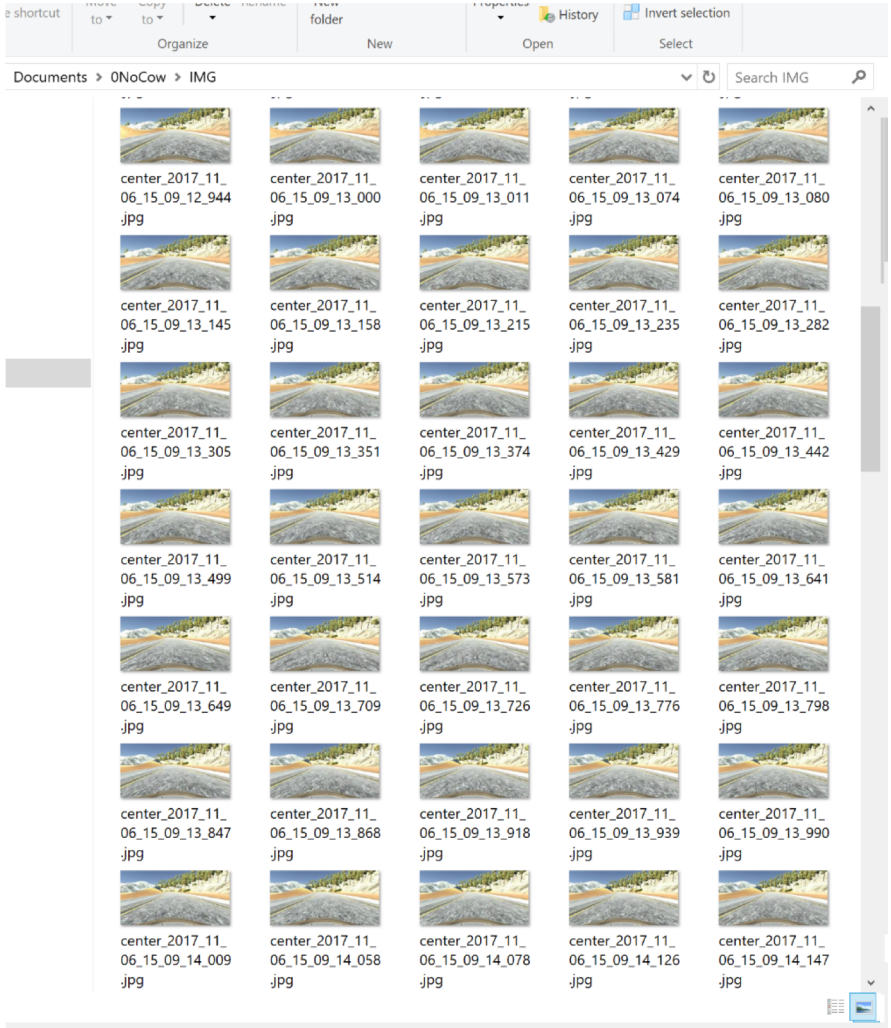
Unity Udacity Simulator



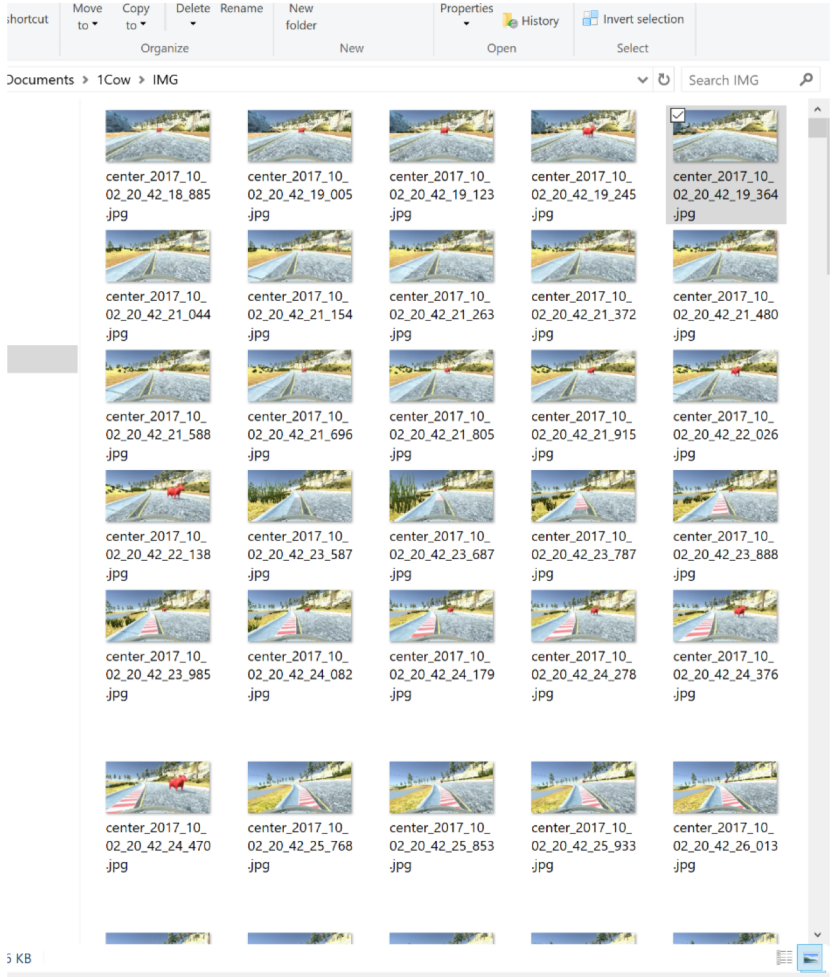
Unity Udacity Simulator



Simulation Example



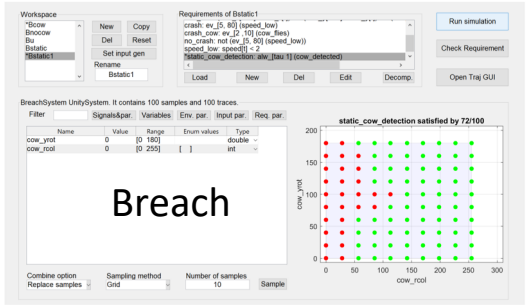
Adding an Obstacle



ACAS: Automatic Cow Avoidance System

- Trained a (Deep) Neural Network on cow/no-cow images all around the track
- Implemented a simple system sending braking commands when a cow is detectecd

Closing the Loop

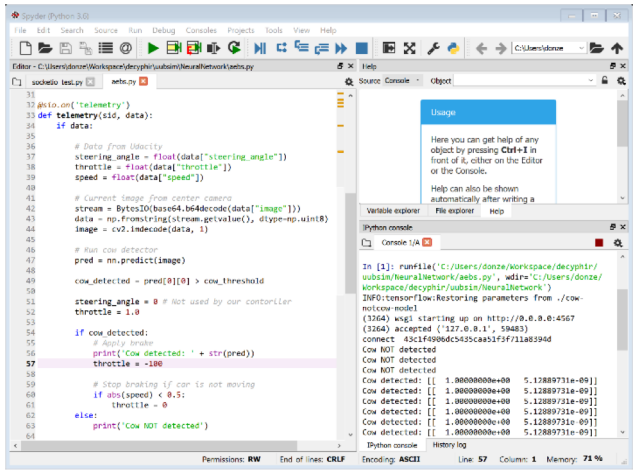


Scene parameters

Simulation Trace

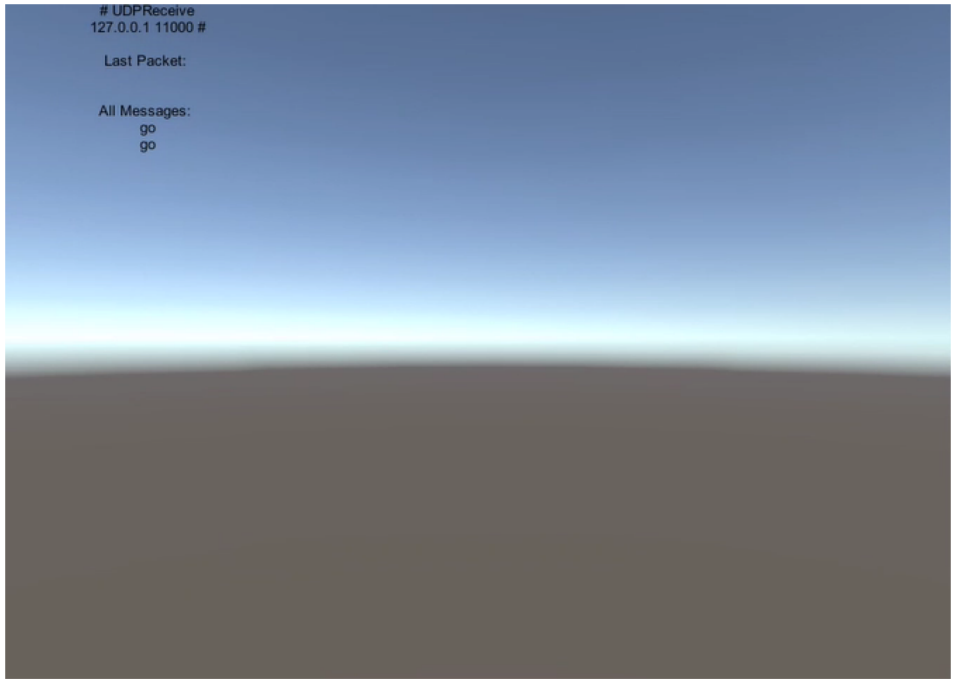
Images

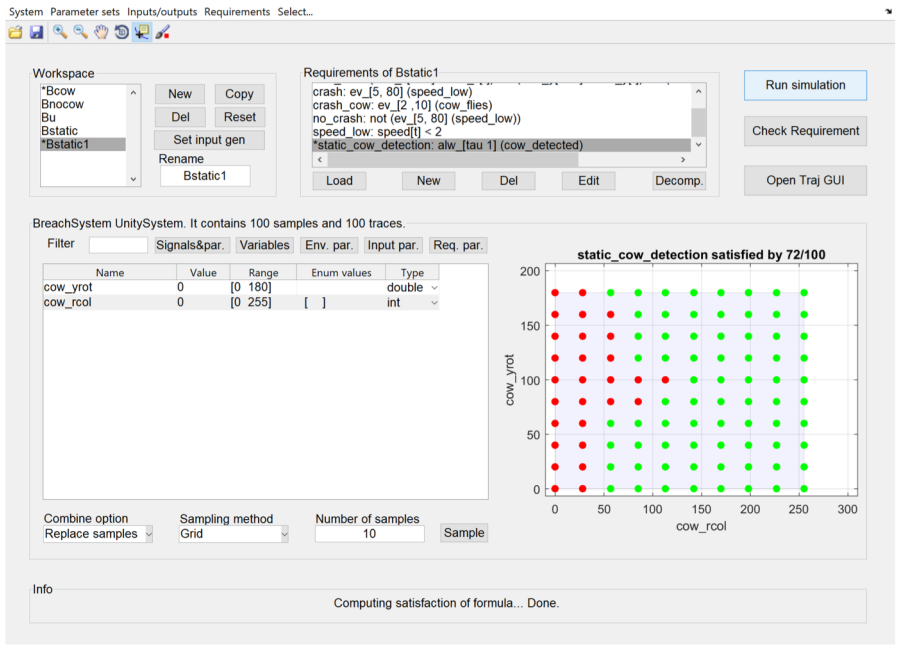
Command



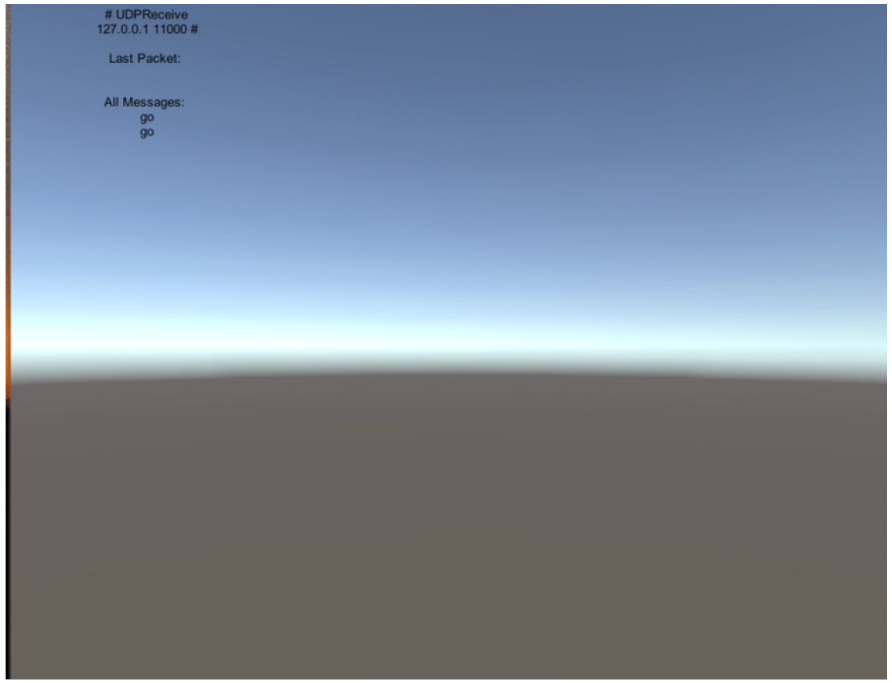
ACAS controller (Python, Tensor Flow)

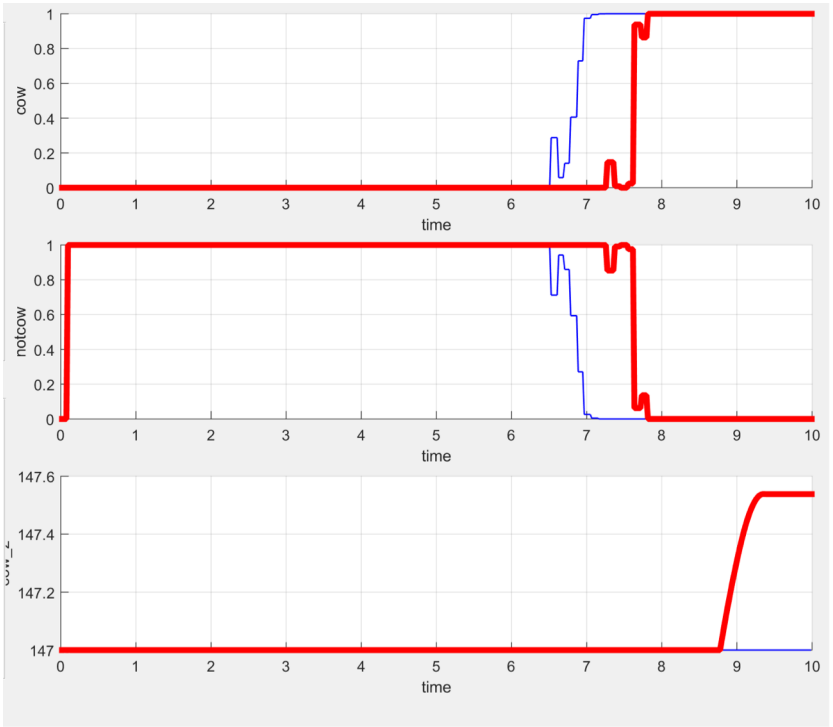
Result on Static Scene





Results on Dynamic Scene





- Cow and no_cow classification scores over time + cow z-position
- Red trace: (black) cow detected too late to avoid collision

Take Away Messages

Traditional CPS

- Established toolchains from models to code
- RV making its way toward adoption
- Always more work to do, in particular usability and expressivity of specification languages

Modern CPS

- No established toolchain
- RV not yet or yet another concern
- Lots of fascinating and very challenging questions involving multiple domains

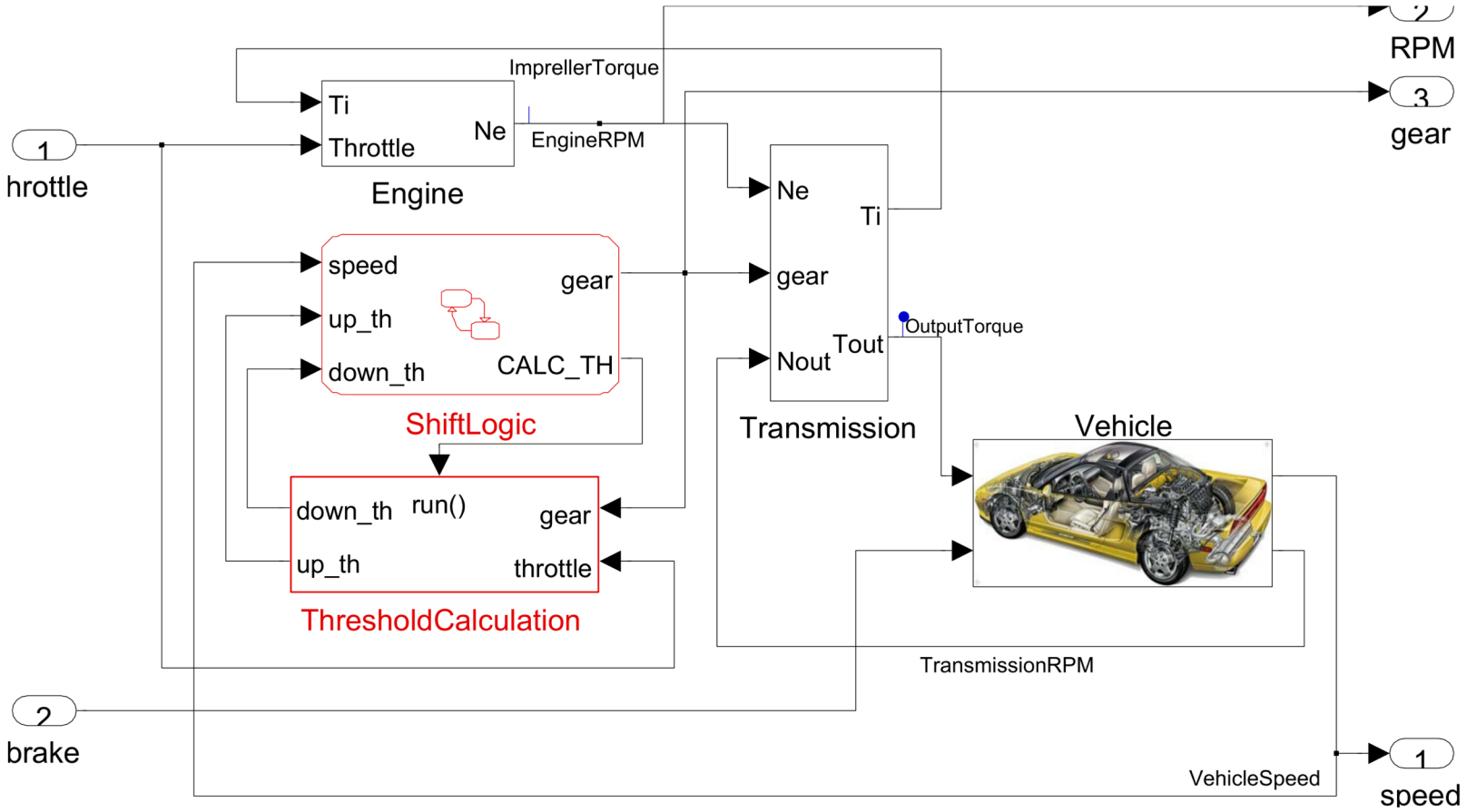
Some Bibliography

- ***Monitoring Temporal Properties of Continuous Signal*,** O. Maler, D. Nickovic, FORMATS/FTRTFT 2004
- ***Robustness of Temporal Logic Specifications,*** G. E. Fainekos, G. J. Pappas FATES/RV 2006
- ***Robust satisfaction of temporal logic over real-valued signals*,** A. Donzé, O. Maler, FORMATS 2010
- ***Mining Requirements From Closed-Loop Control Models,*** X. Jin, A. Donzé, J. V. Deshmukh, S. A. Seshia, IEEE Trans. on CAD of Integrated Circuits and Systems, 2015
- ***Specification-based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications*,** E. Bartocci, J. Deshmukh, A. Donze, G. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan, Lectures on Runtime Verification - Introductory and Advanced Topics, LNCS 10457, Springer, 2018
- ***Compositional Falsification of Cyber-Physical Systems with Machine Learning Components,*** T. Dreossi, A. Donzé, S. A. Seshia,, NFM'17, arXiv:1703.00978
- Breach, http://github.org/decyphir/breach

Thanks for your attention

© Google Image search

Breach Demo


RPM
ImprellerTorque
Ti
Throttle
Ne
EngineRPM
3
gear
1
hrottle
Engine
Ne
Ti
speed
gear
gear
up_th
OutputTorque
down_th
CALC_TH
Nout
Tout
ShiftLogic
Transmission
Vehicle
down_th
run()
gear
up_th
throttle
ThresholdCalculation
TransmissionRPM
2
brake
1
VehicleSpeed
speed