SVV.lu
software verification & validation

SnT
securityandtrust.lu

Software Verification and Validation Laboratory

svv.lu

uni.lu
UNIVERSITÉ DU LUXEMBOURG

SnT Centre for Security, Reliability and Trust

- Part of the University of Luxembourg
- Focused on industry-driven research and innovation
- 24 MEUR turnover in 2017 (70% competitive)
- Headcount >260, >45 nationalities
- Most highly-cited scientists in Luxembourg
- CS @ UL ranked 58 in the world
- 31 industry and public-service partners
- 60 industrial PhD candidates (60% of total)
- 50 Research Associates (60% of total)
- FNR Public-Private Partnership programs

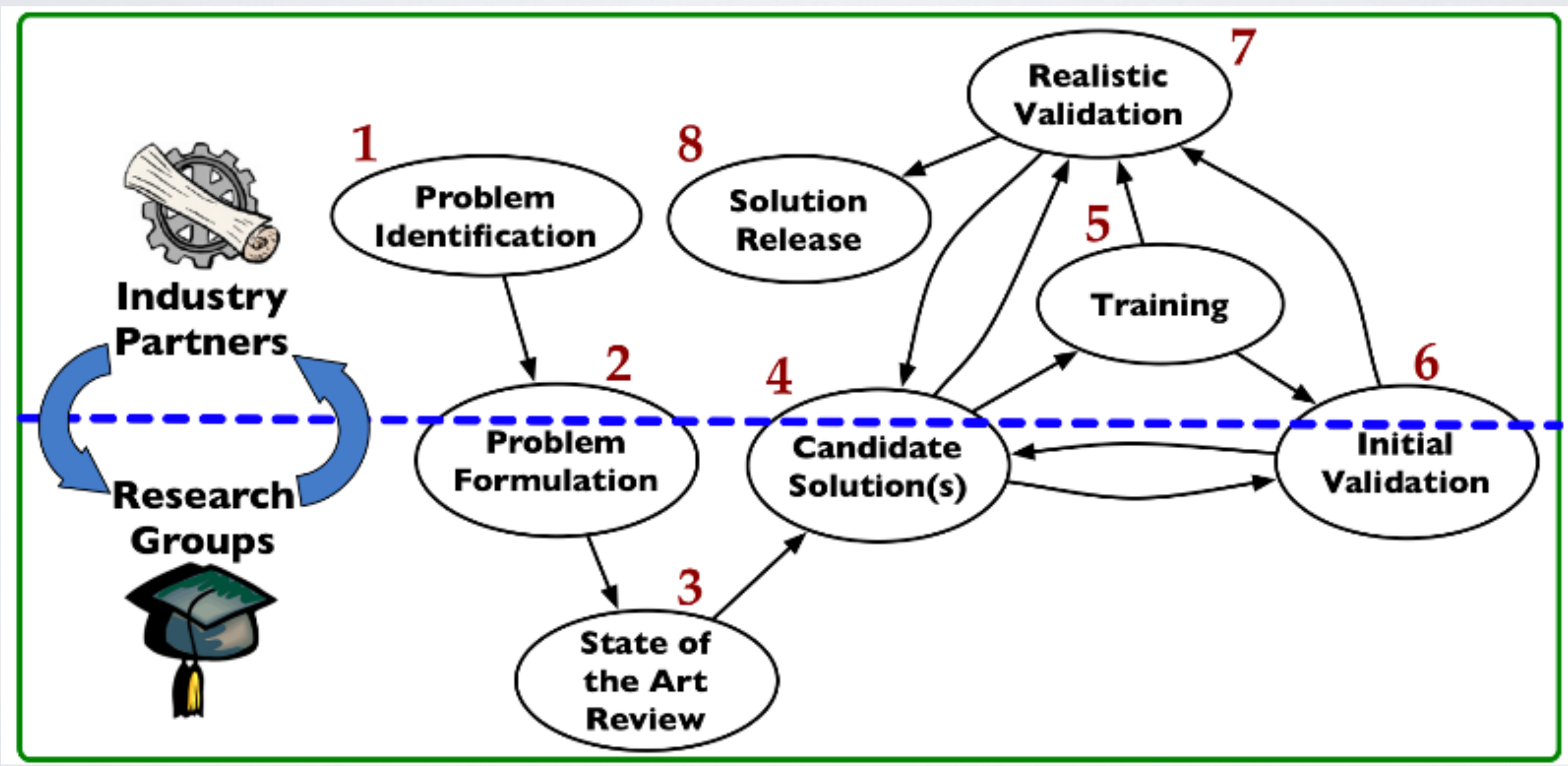
securityandtrust.lu

SVV Lab Overview

- Established in 2012
- Requirements Engineering, Security Analysis, Design Verification, Automated Testing, Runtime Monitoring
- ~ 25 lab members
- Eight partnerships
- ERC Advanced Grant
- Budget 2016: ~2 Meuros

Mode of Collaboration

- Strong emphasis on applied research, driven by needs
- Tight, large-scale industrial collaborations



Meeting Objectives

- Discuss partnership on testing and verifying Simulink models (i.e., controllers, plant)

- Significant funding with ERC grant and co-funding with FNR (Luxembourg funding agency)

- Refine the objectives in current proposal

- Demo our existing tools

Meeting Agenda

- March 2nd
  - 11:30am – 12:30pm Discussing ASTech models
  - 2pm – 3pm Meeting with Bjorn Ottersten, SnT Director
  - 3 pm – 5pm Technical presentations (Audi, SnT)
- March 3rd
  - 9am – 11am Technical presentation (Cnted) + demos
  - 11 am – 12pm Discussions (collaboration, partnership)
  - 2pm – 4pm Discussions (collaboration, partnership)

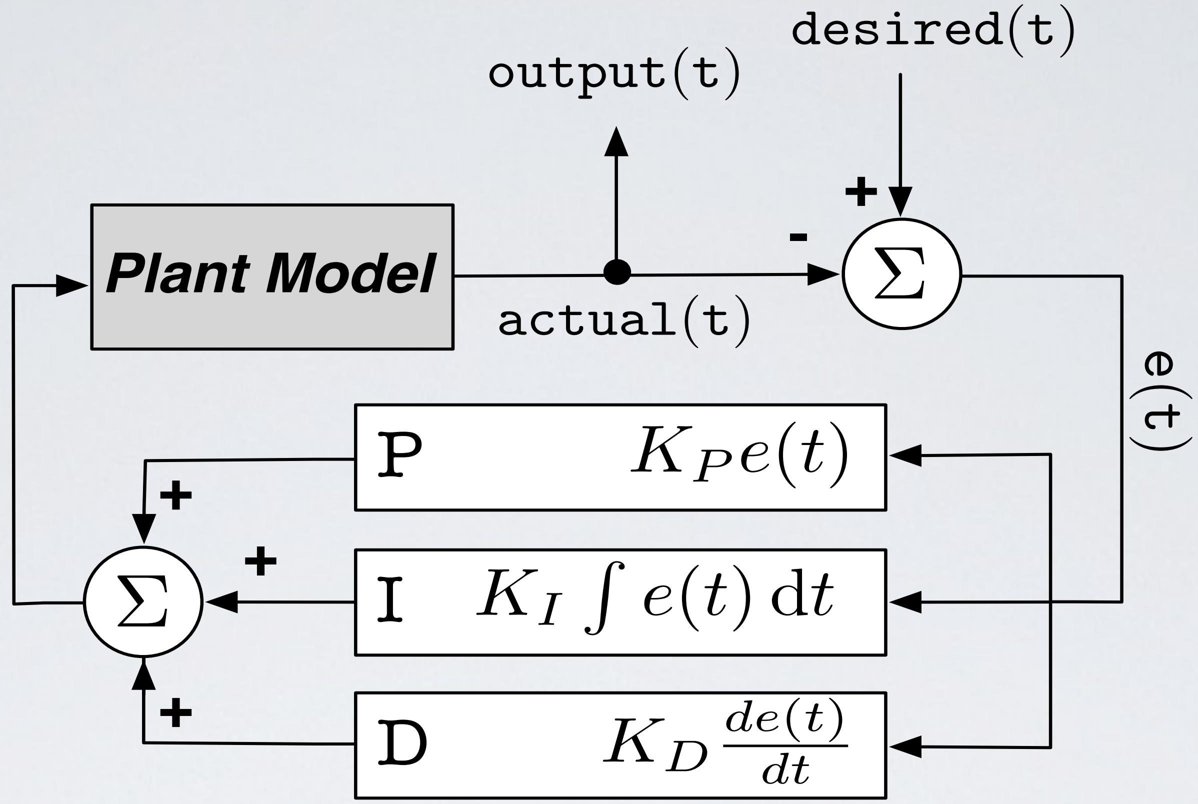
ASTech Models

SVV Technical Presentation

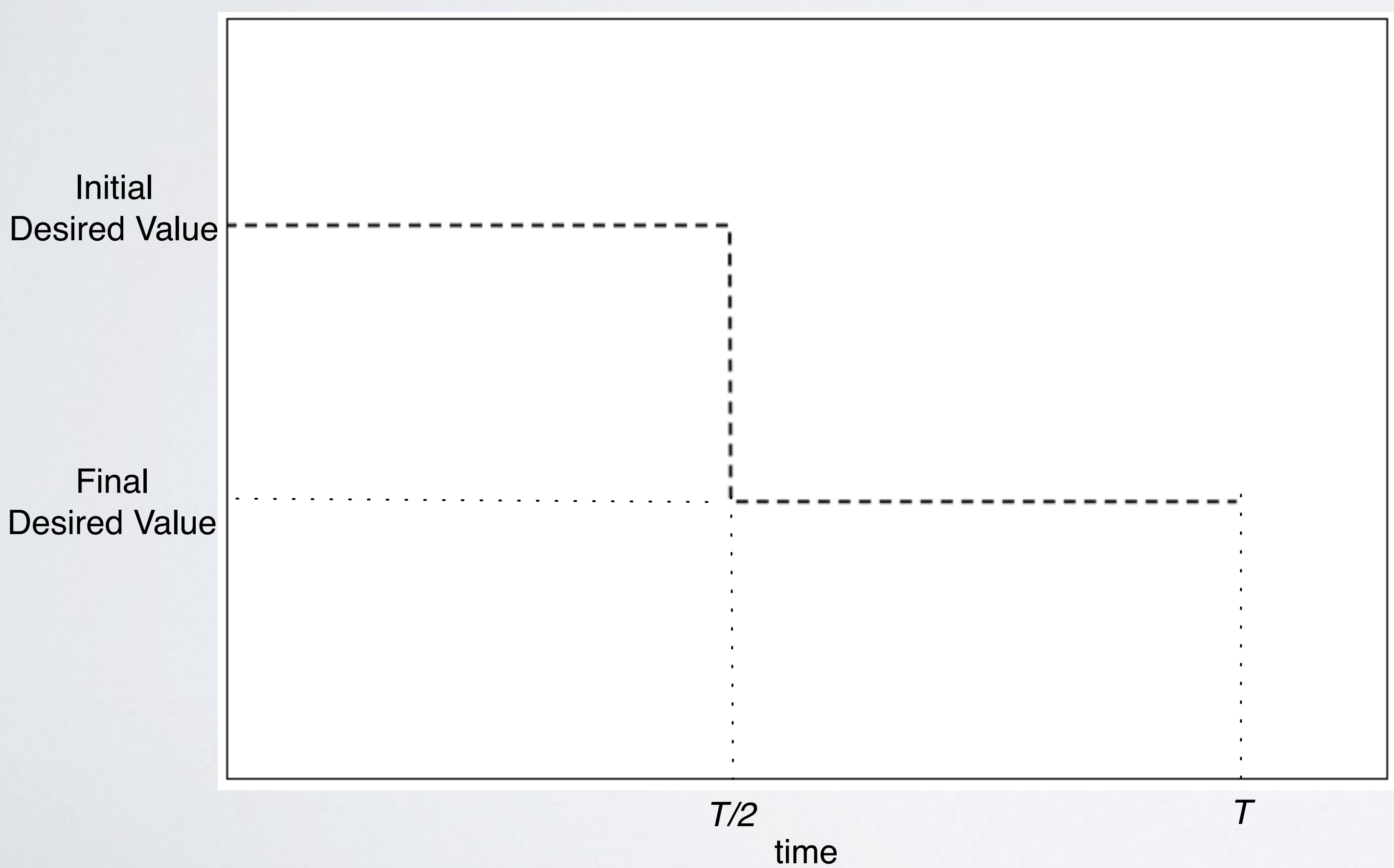
Outline

- Introduction
    - Testing cyber-physical systems (ERC advanced grant)
- Automated testing of embedded software systems
    - Testing closed-loop controllers
    - Testing Simulink models
    - Fault Localization of Simulink models
- Other projects in the automotive domain
- Proposals for ASTech-SnT collaboration

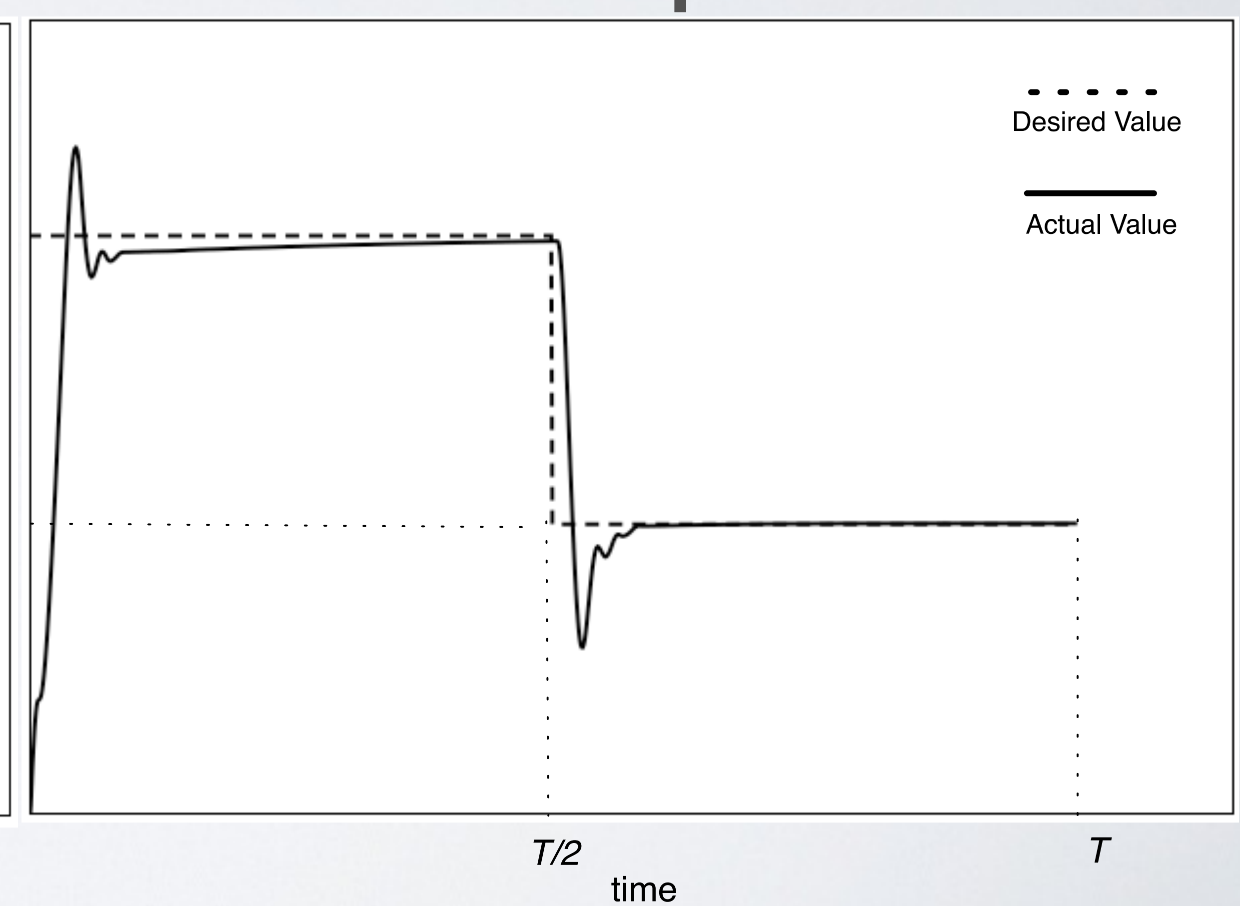
Testing Closed Loop Controllers



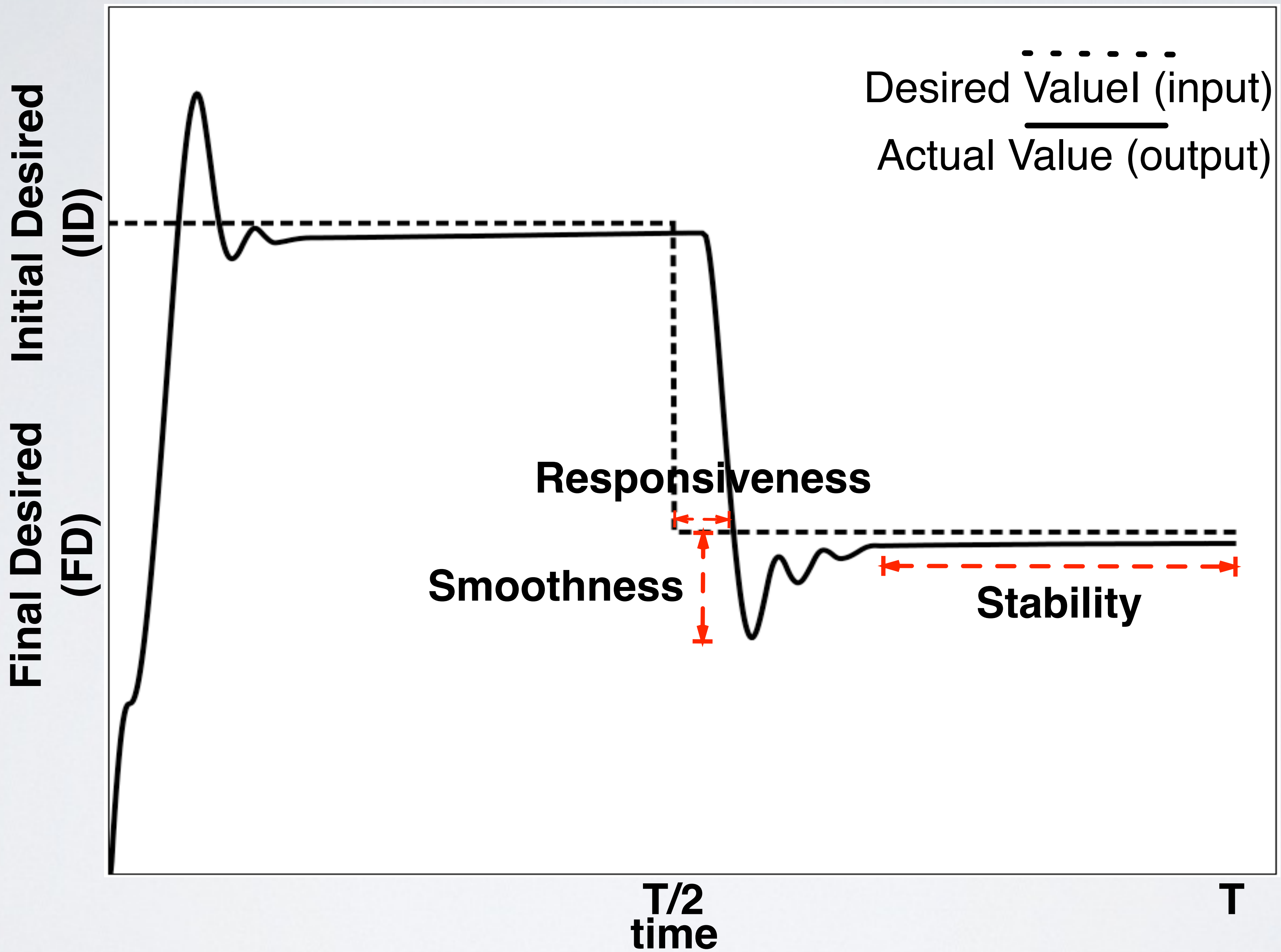
Test Input



Test Output



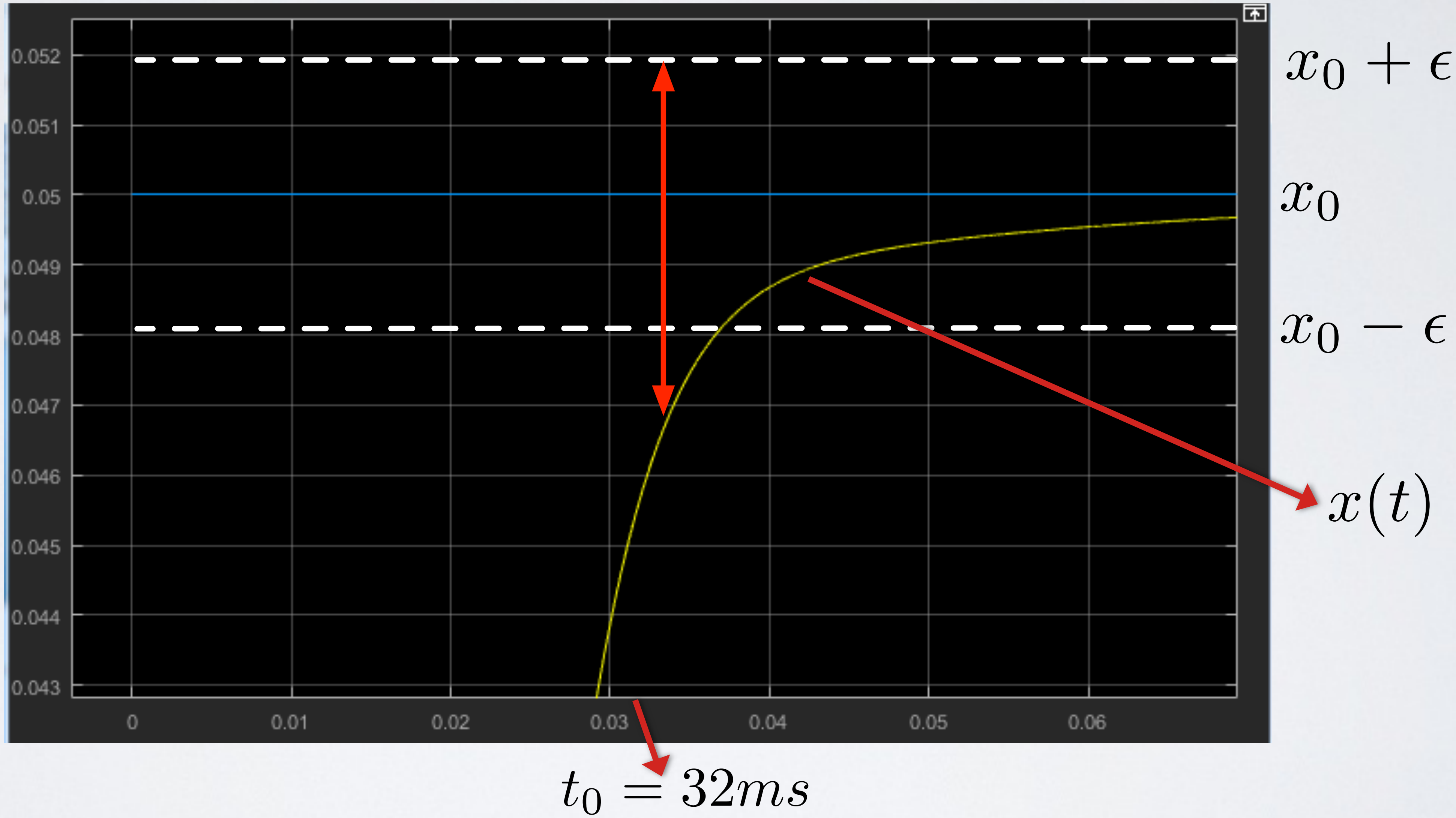
Requirements and Test Objectives



We identify high risk behaviors by maximizing test objectives

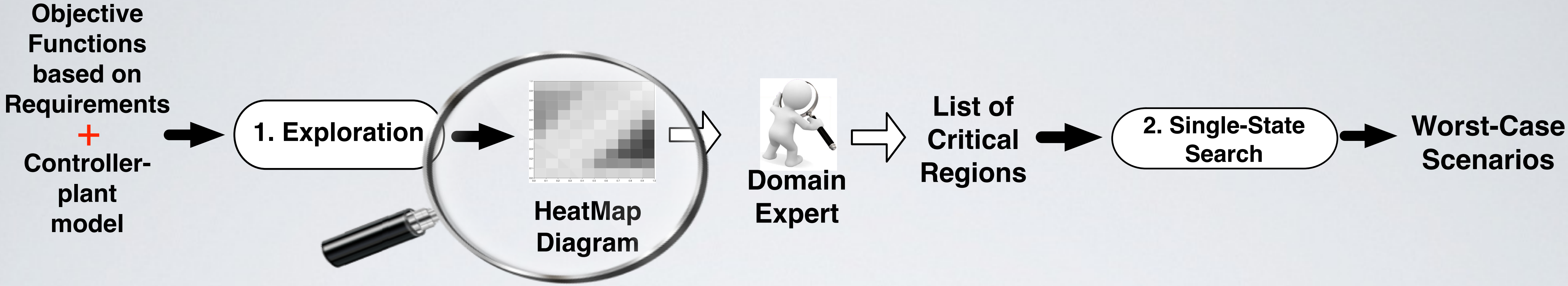
Requirements and Test Objectives

Requirement: As soon as braking is requested, the contact between caliper and disk should occur within 32ms

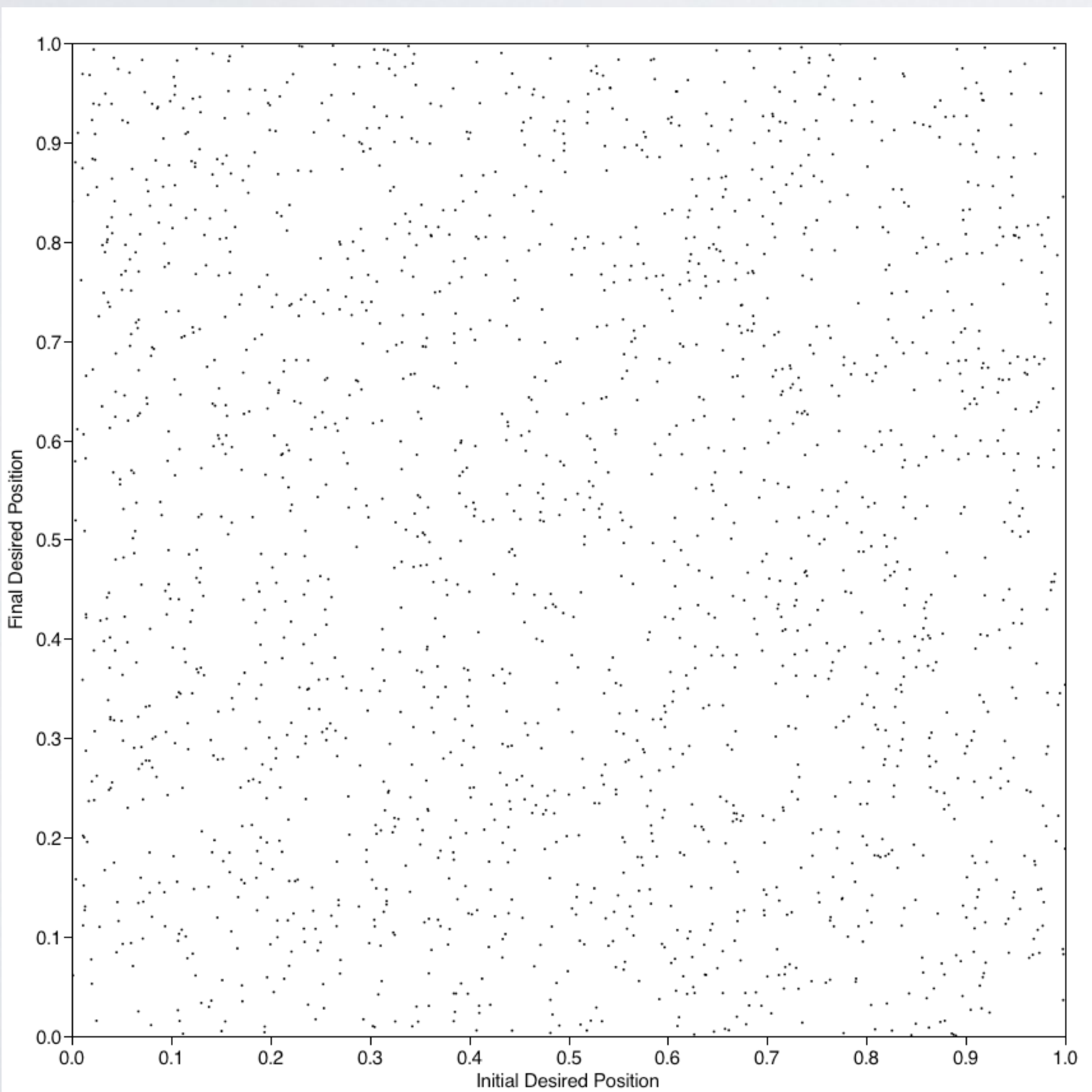


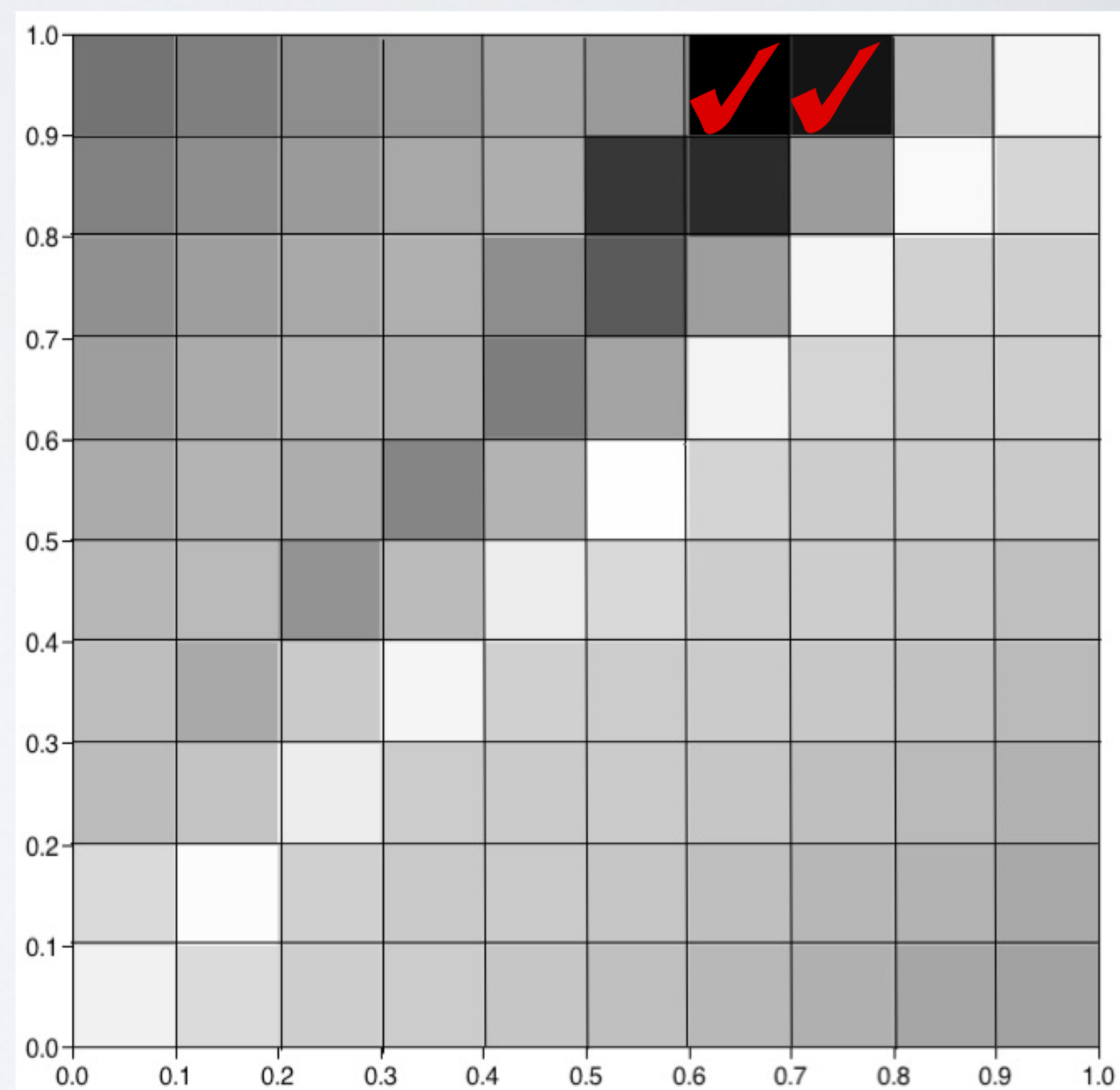
$$\text{Min}\{\text{Max}\{\text{Max}\{|x(t) - (x_0 + \epsilon)|, |x(t) - (x_0 - \epsilon)|\}\}_{t_0 \leq t \leq T}\}$$

Continuous Controller Tester

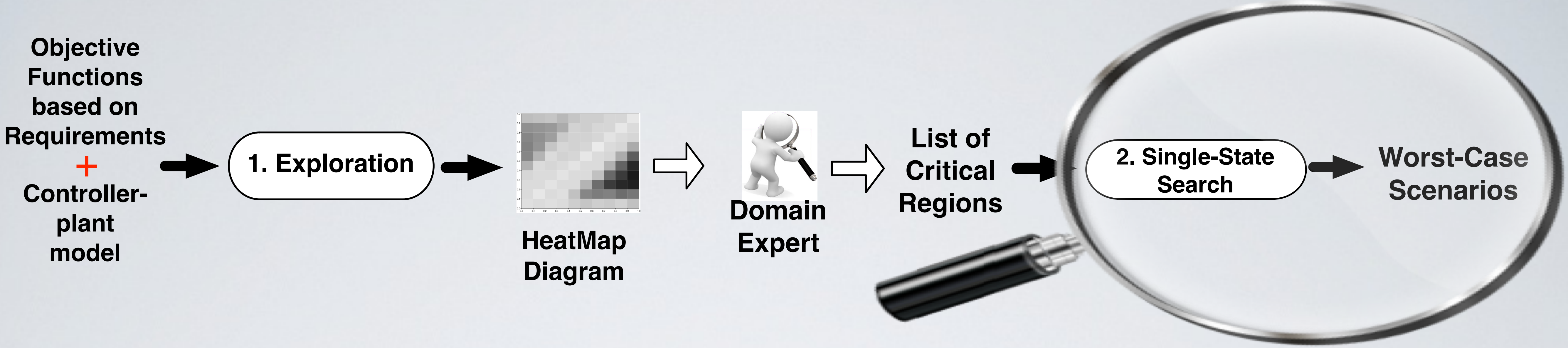


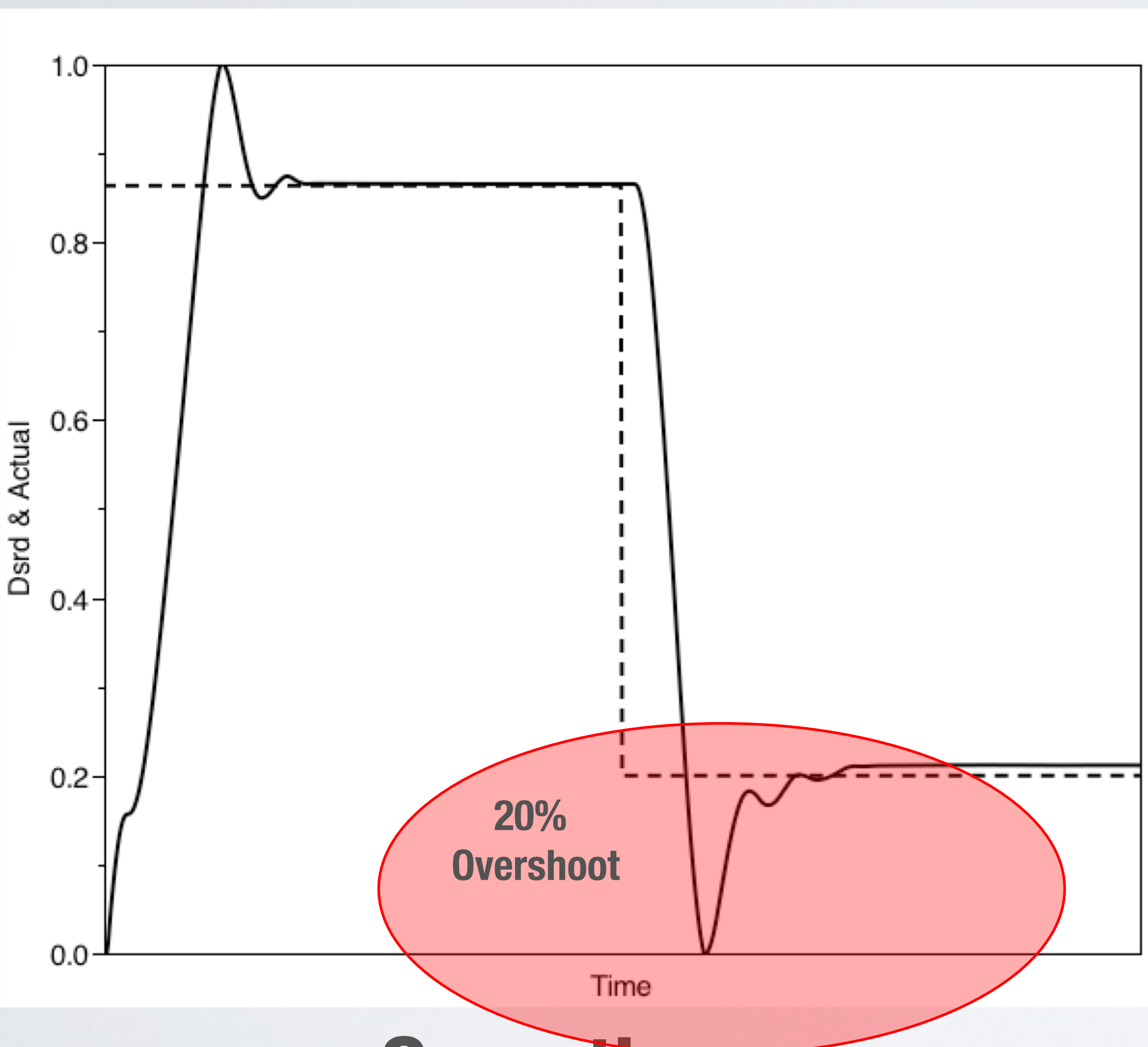
Initial Desired Value



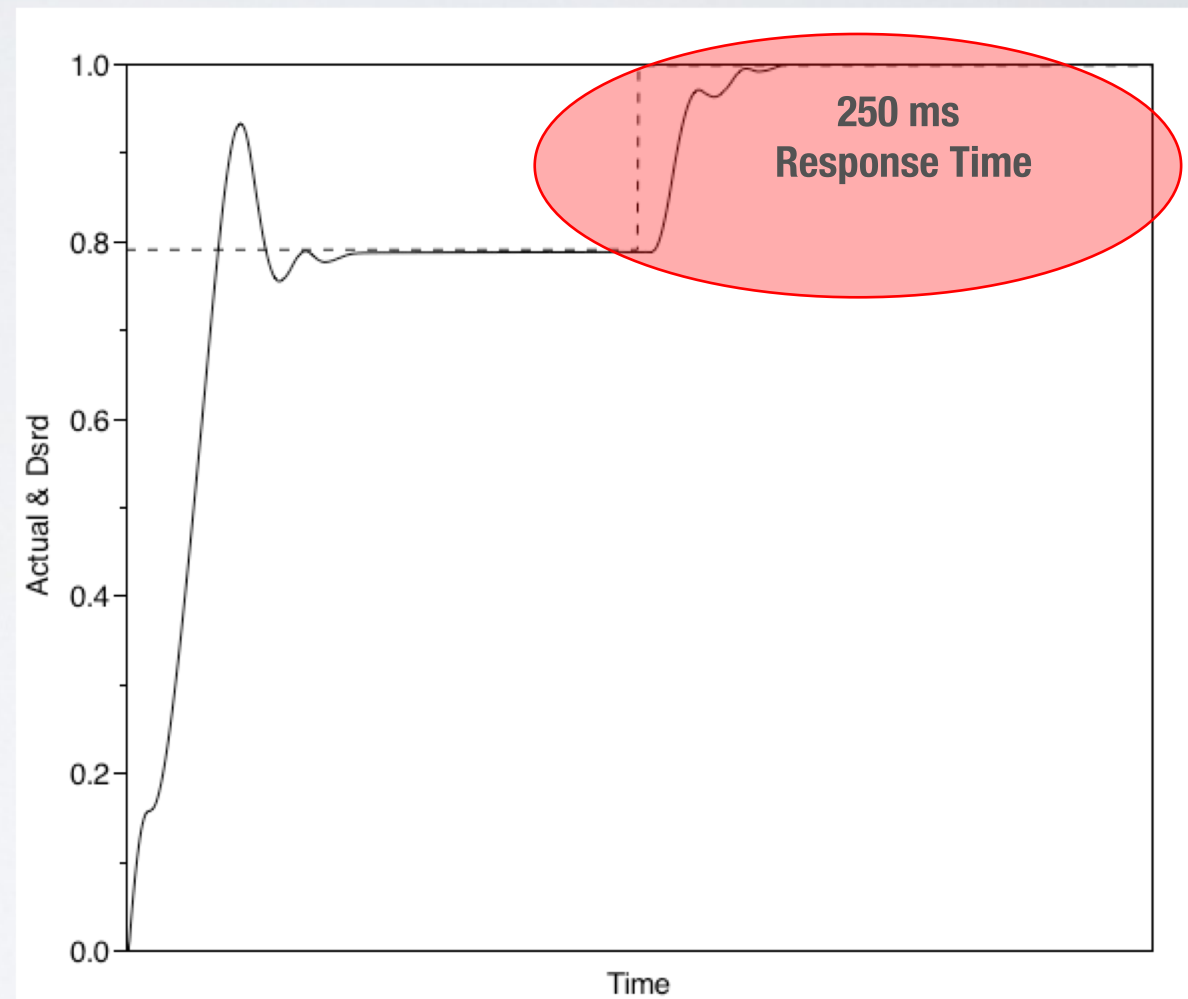


Final Desired Value



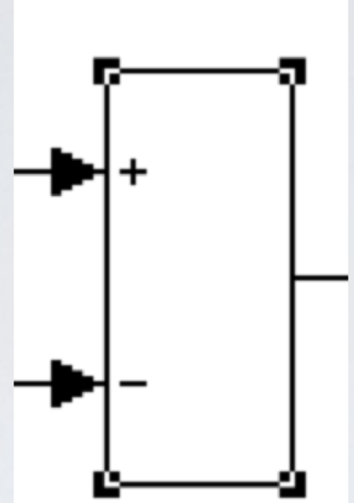


Smoothness

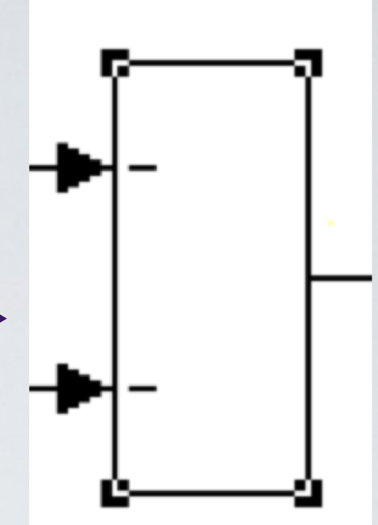


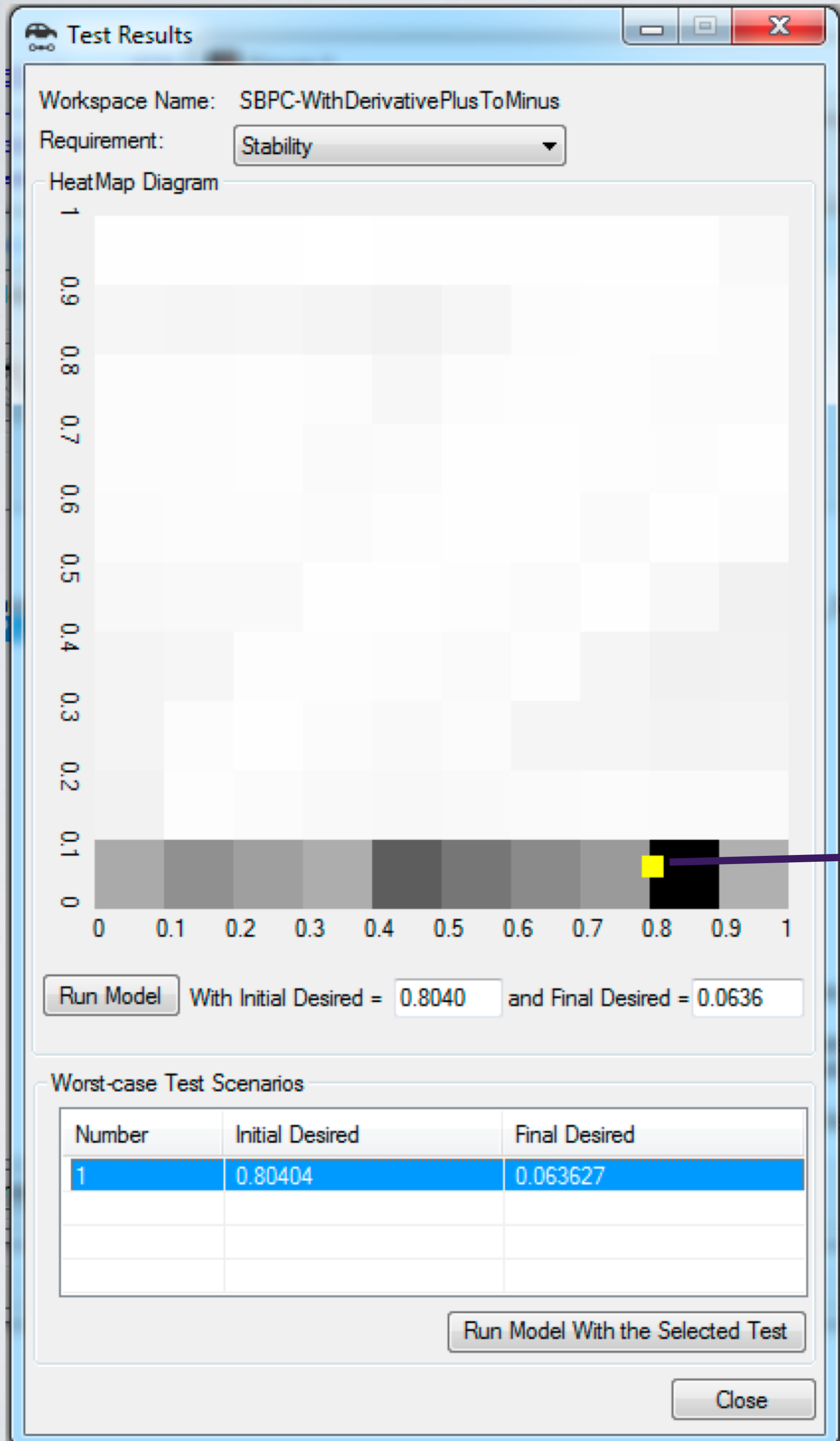
Responsiveness

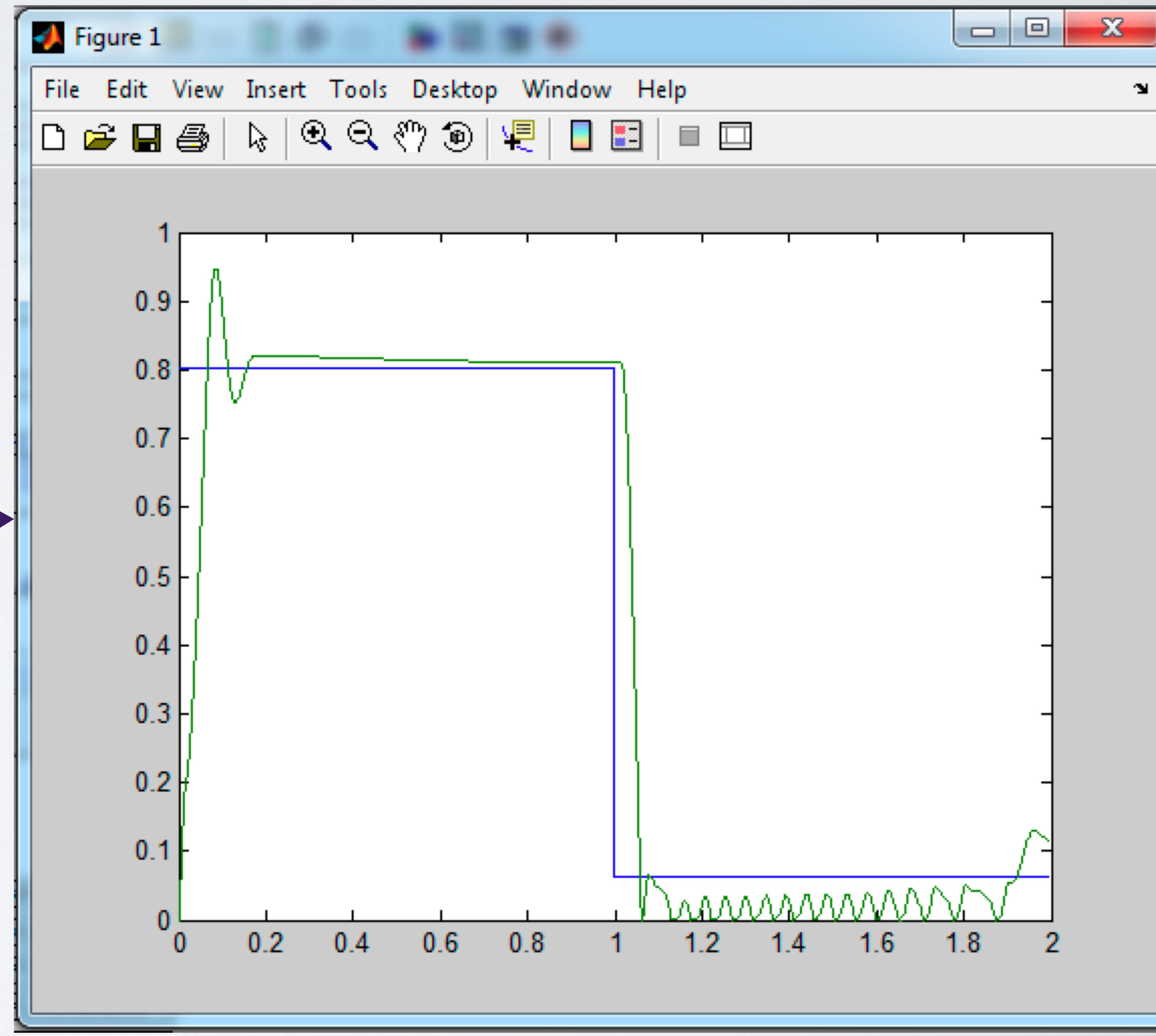
Finding Seeded Faults



Inject Bug







Summary

- We found several interesting test scenarios (worst cases) during Model-in-the-Loop (MiL) testing compared to what our partner had found so far

- These scenarios are also run at the Hardware-in-the-Loop (HiL) level, where testing is much more expensive:

MiL results -> test selection for HiL

Testing Simulink Models (Closed Loop and Open Loop Controllers)

Objectives

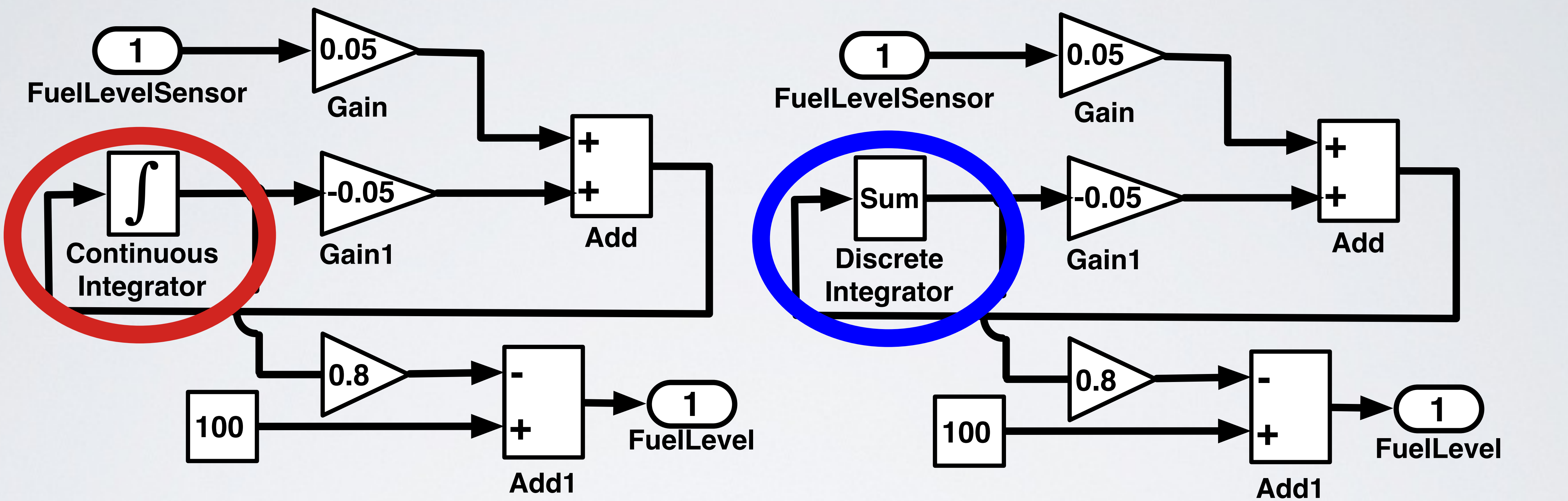
- Testing **Simulink/Stateflow models in their entirety**
- **Without requiring plant models**
- **Without requiring automated test oracles**

Simulink Testing Challenge I

Incompatibility

Existing testing techniques are not applicable to **simulation models (with time-continuous behaviors)**

Incompatibility Challenge -- Example



Simulation Model

Code Generation Model

Not Applicable

Applicable

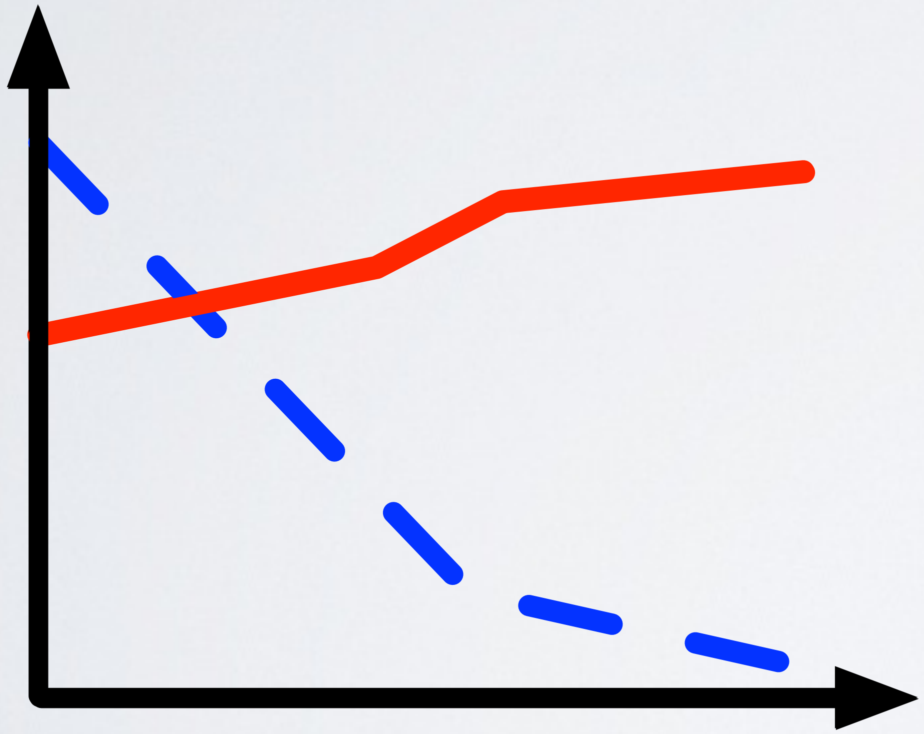
Simulink Testing Challenge II

Low Fault-Revealing Ability

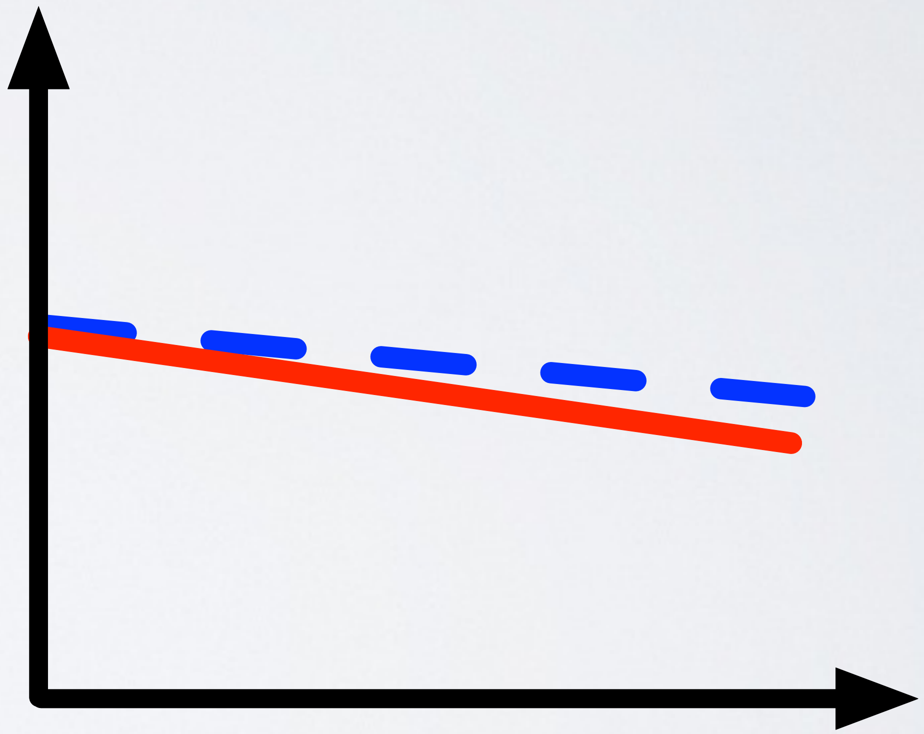
Existing testing techniques make **unrealistic assumptions** about **test oracles**

Low Fault-Revealing Ability Example

Covers the fault **and is Likely** to reveal it



Covers the fault **but is very unlikely** to reveal it



Faulty Model Output

Correct Model Output

Our Approach

Search-based Test Generation Driven by **Output-Diversity** and **Anti-Patterns**

Search-Based Test Generation

Search Procedure

S ⟵ Initial *Candidate Solution*

Repeat

R ⟵ *Tweak* (S)

if *Fitness* (R) > *Fitness* (S)

S ⟵ R

Until *maximum resources spent*

Return S

Initial Test Suite

Slightly Modifying Each Test Input

Output-based Heuristics

Output-Based Heuristics

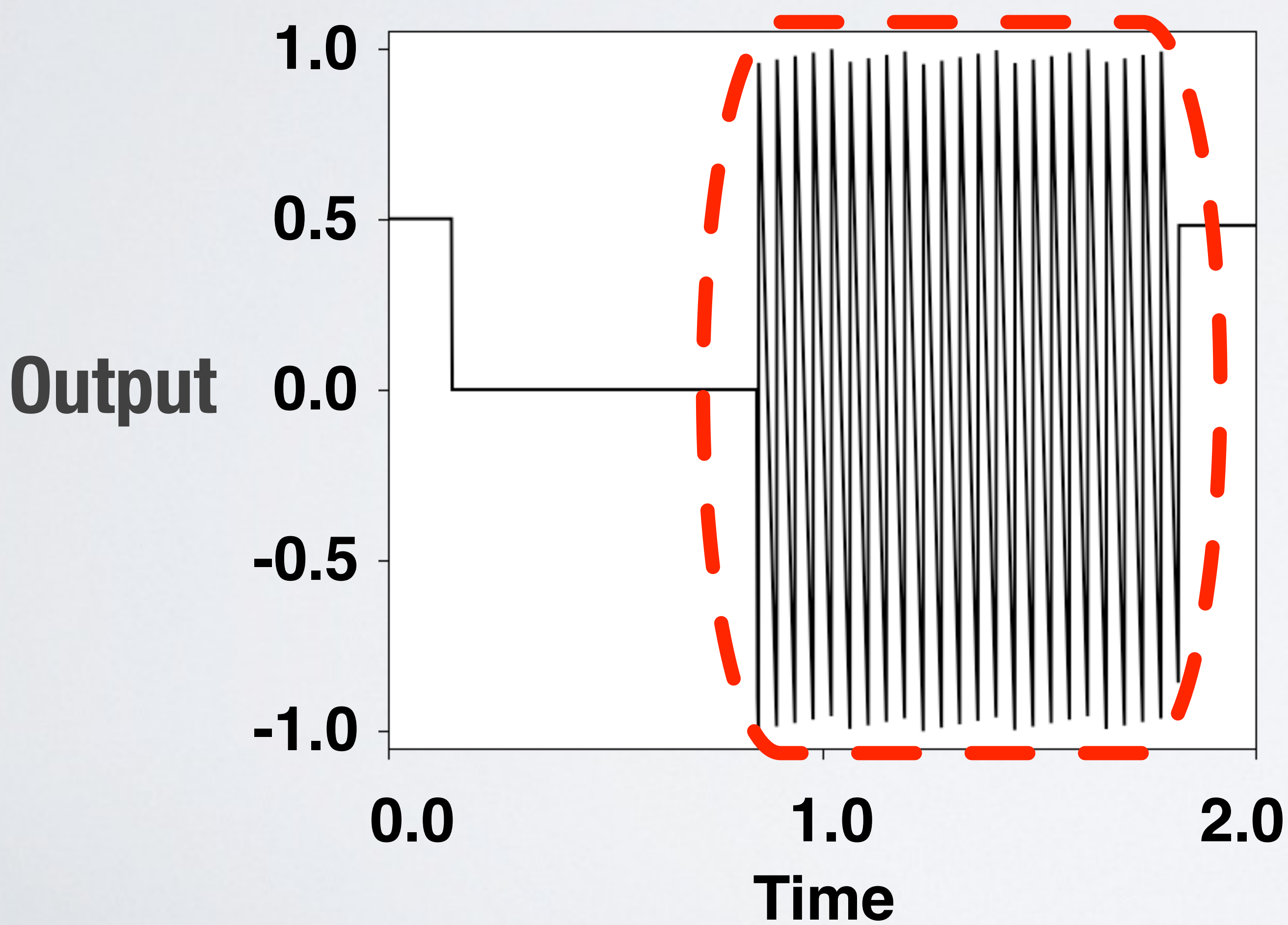
Failure Patterns

Output Diversity

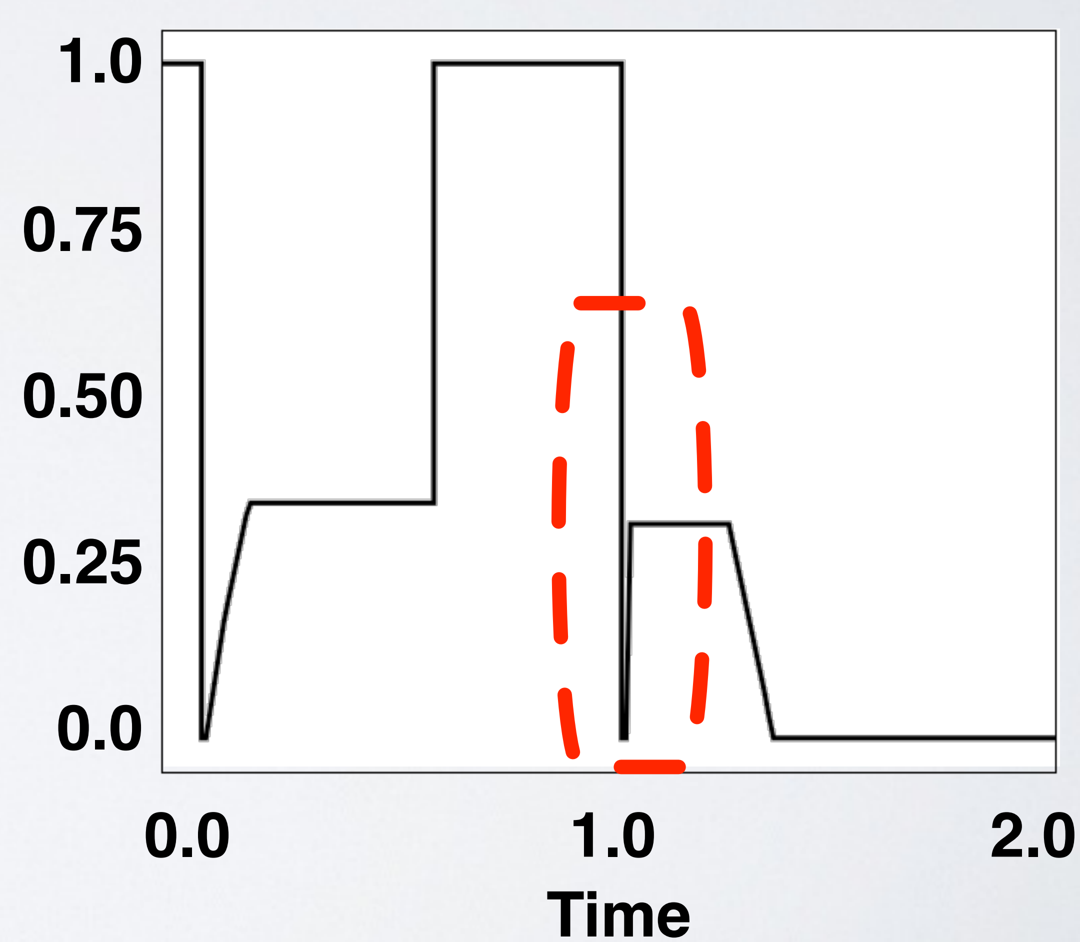
Failure-based Test Genration

- Maximizing the likelihood of presence of specific failure patterns in output signals

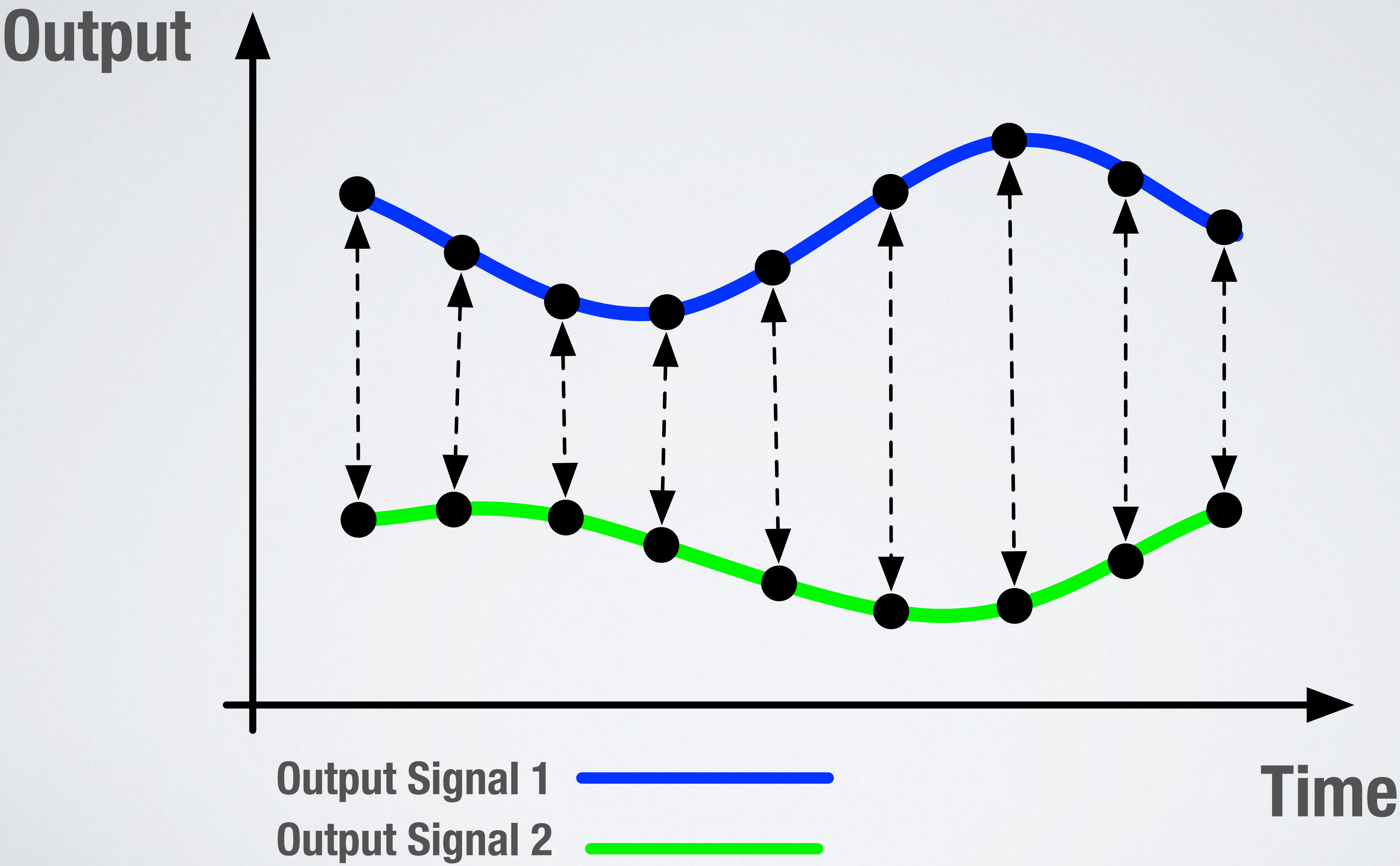
Instability



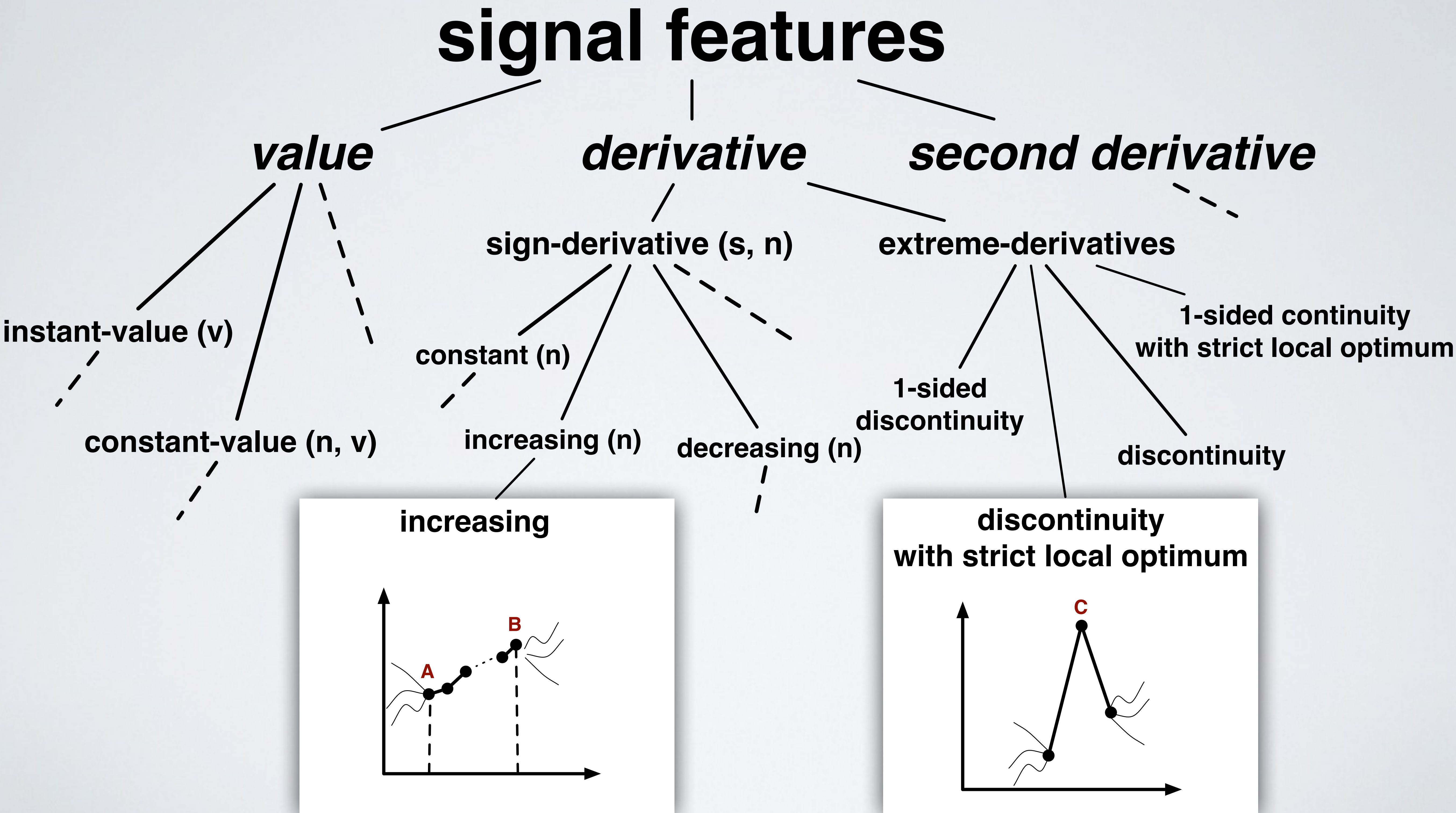
Discontinuity



Output Diversity -- Vector-Based



Output Diversity -- Feature-Based



Evaluation

How does the fault revealing ability of our algorithm compare with that of Simulink Design Verifier?

Simulink Design Verifier (SLDV)

Simulink Design Verifier

MathWorks®

- Underlying Technique: Model Checking and SAT solvers
- Test objective: Testing is guided by structural coverage

Our Approach vs. SLDV

- Our approach outperformed SLDV in revealing faults

Faults	1	2	3	4	5	6	7	8	9	10	11
SLDV	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Our Approach	20	16	20			20	14	17		20	
Faults	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
SLDV	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Our Approach		14		20	20	20	20	20	20	15	15

✗ SLDV could not find the fault

✓ SLDV found the fault

The number of fault-revealing runs of our algorithm (out of 20)

SimCoTest Tool

https://sites.google.com/site/simcotesttool/

https://sites.google.com/site/cocotesttool/

Summary

- We evaluated SimCoTest on **seven representative Delphi Simulink models and one model from Bosch research lab**

- Hands-on tutorial to ten Delphi engineers:

  - “SimCoTest is useful for early stages of controller design to identify and detect design flaws.”

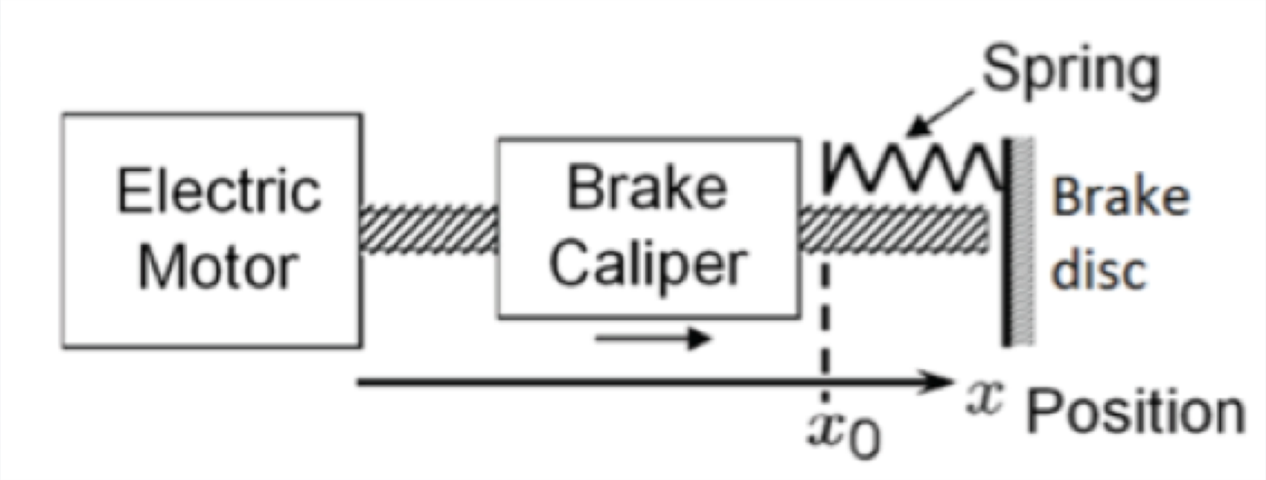
- We found some issues in Delphi models

- We plan to follow further development of SimCoTest and its commercialization

Tool Demos

Case Study

- Electro-Mechanical Braking (EMB)
- A public-domain model developed by the Bosch Research lab (http://cps-vo.org/node/20289)



- EMB Simulink model
  - Consists of a physical plant model, a PID controller and a state-flow
  - Is simulated by a variable-step solver
  - Contains float variables and float computation

Demo of SimCoTest

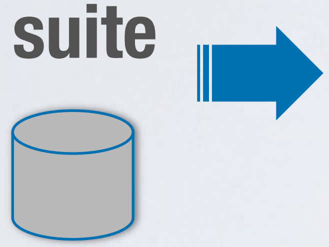
- SimCoTest is able to identify the following error patterns in EMB output
  - Oscillation (marginal stability)
  - Discontinuity
  - Growth to infinity (instability and not marginally stable)

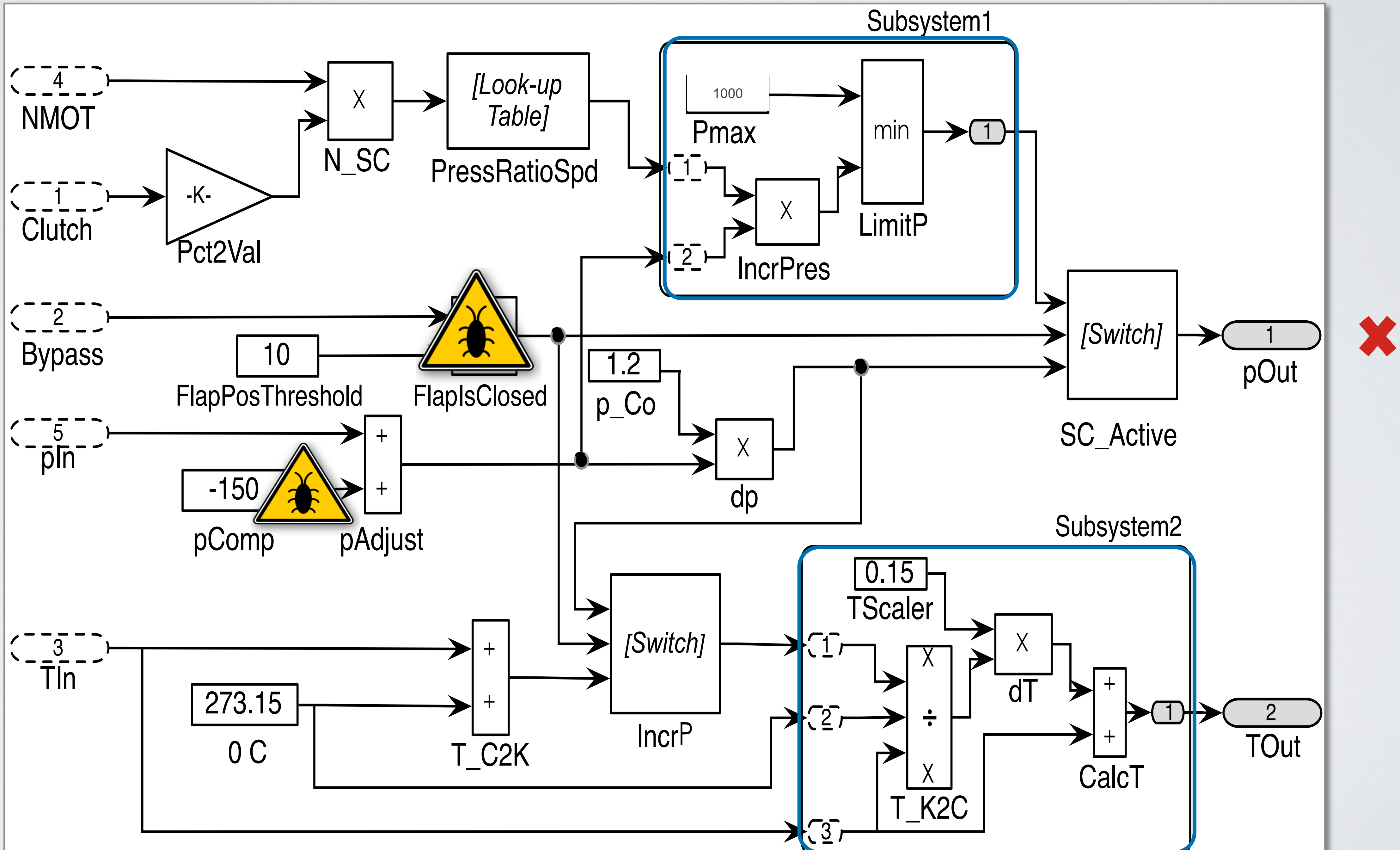
Fault Localization in Simulink Models

Fault Localization

A Simulink model

A test suite





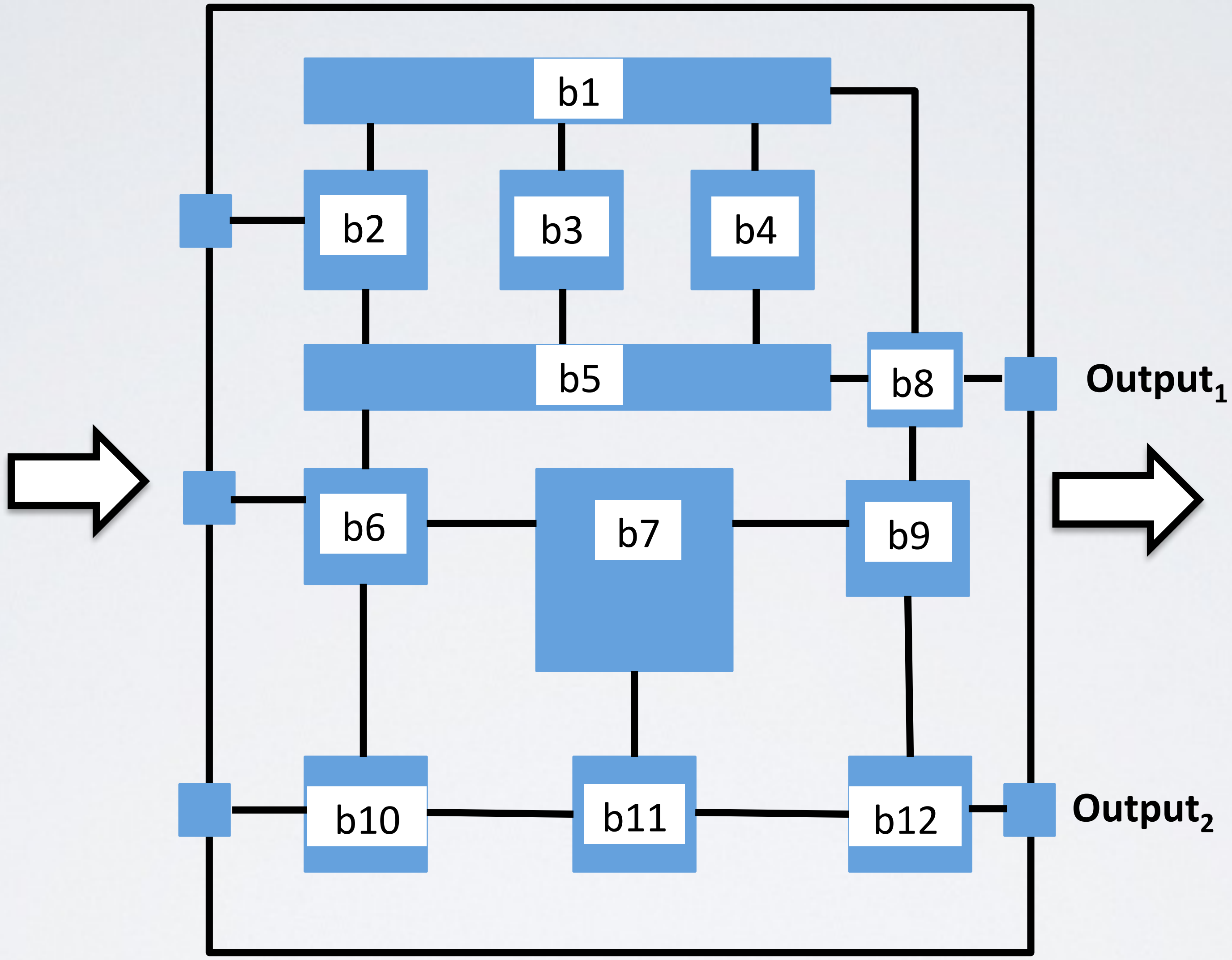
Statistical Debugging

Test suite

t1

t2

t3



For O_2 :

b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

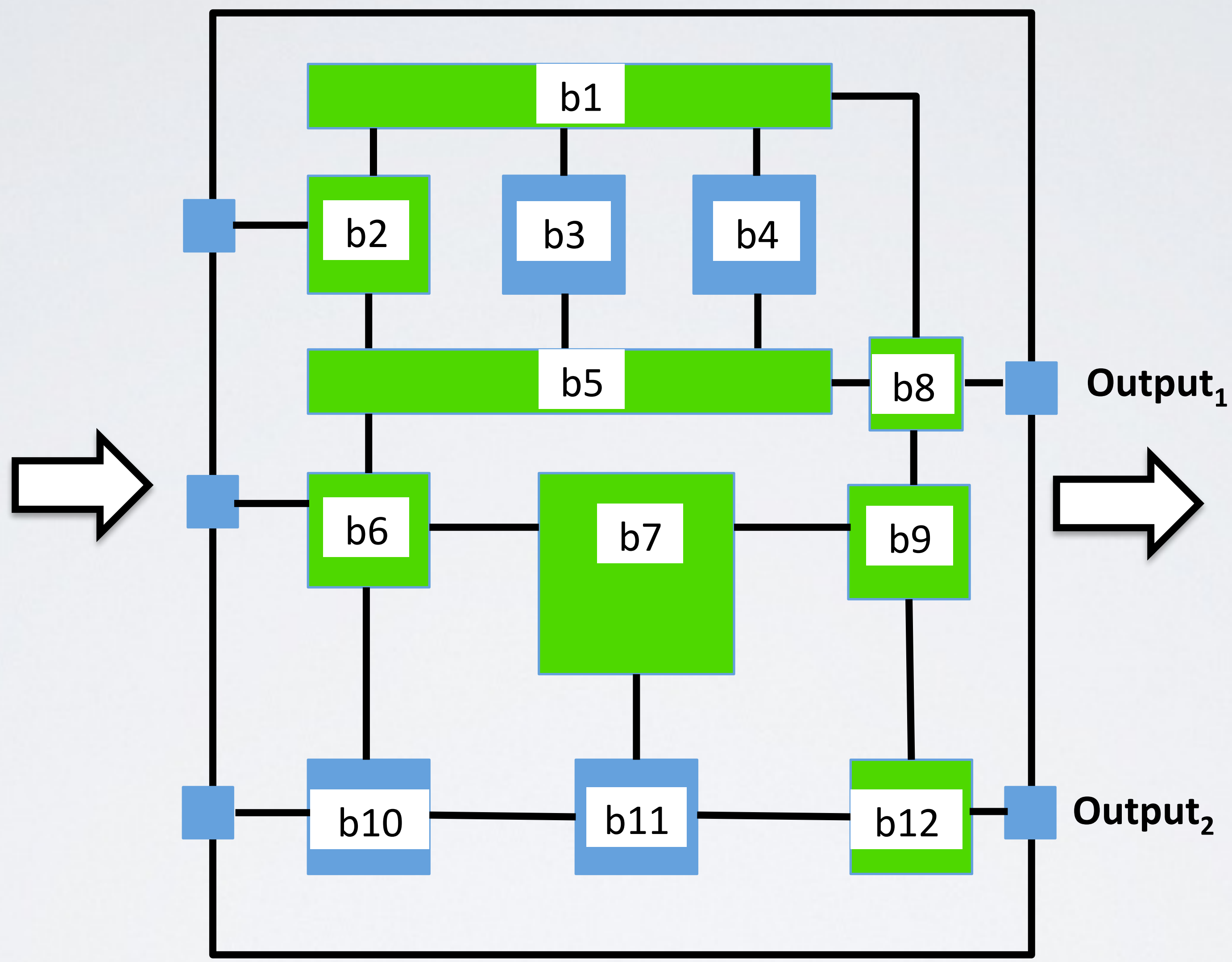
Not executed
Executed, pass
Executed, fail

Statistical Debugging

Test Suite

t2

t3



Status

t1

For O_2 :

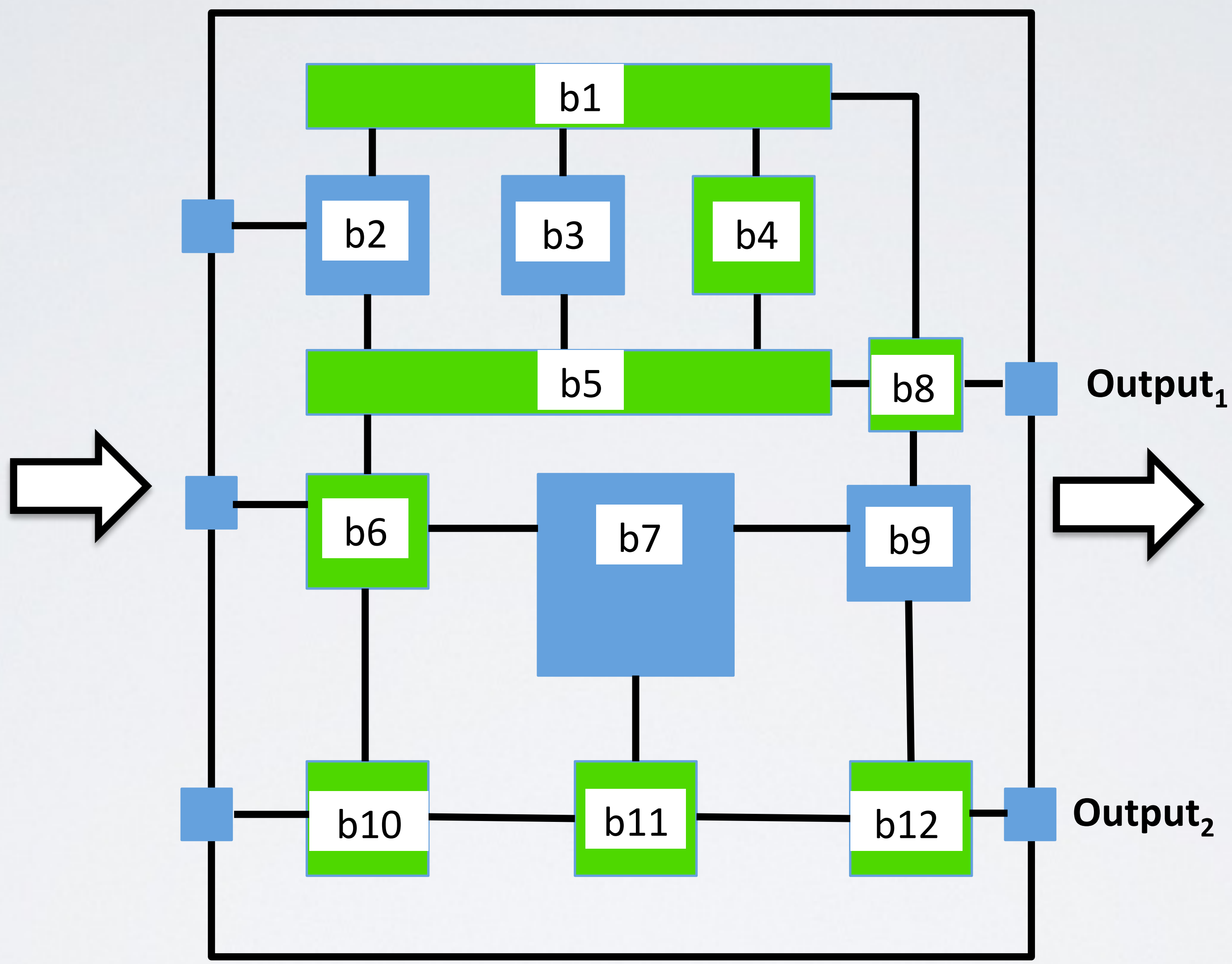
b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
1	1	0	0	1	1	1	1	1	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0

Not executed
Executed, pass
Executed, fail

Statistical Debugging

Test Suite

t3



Status

t1 t2

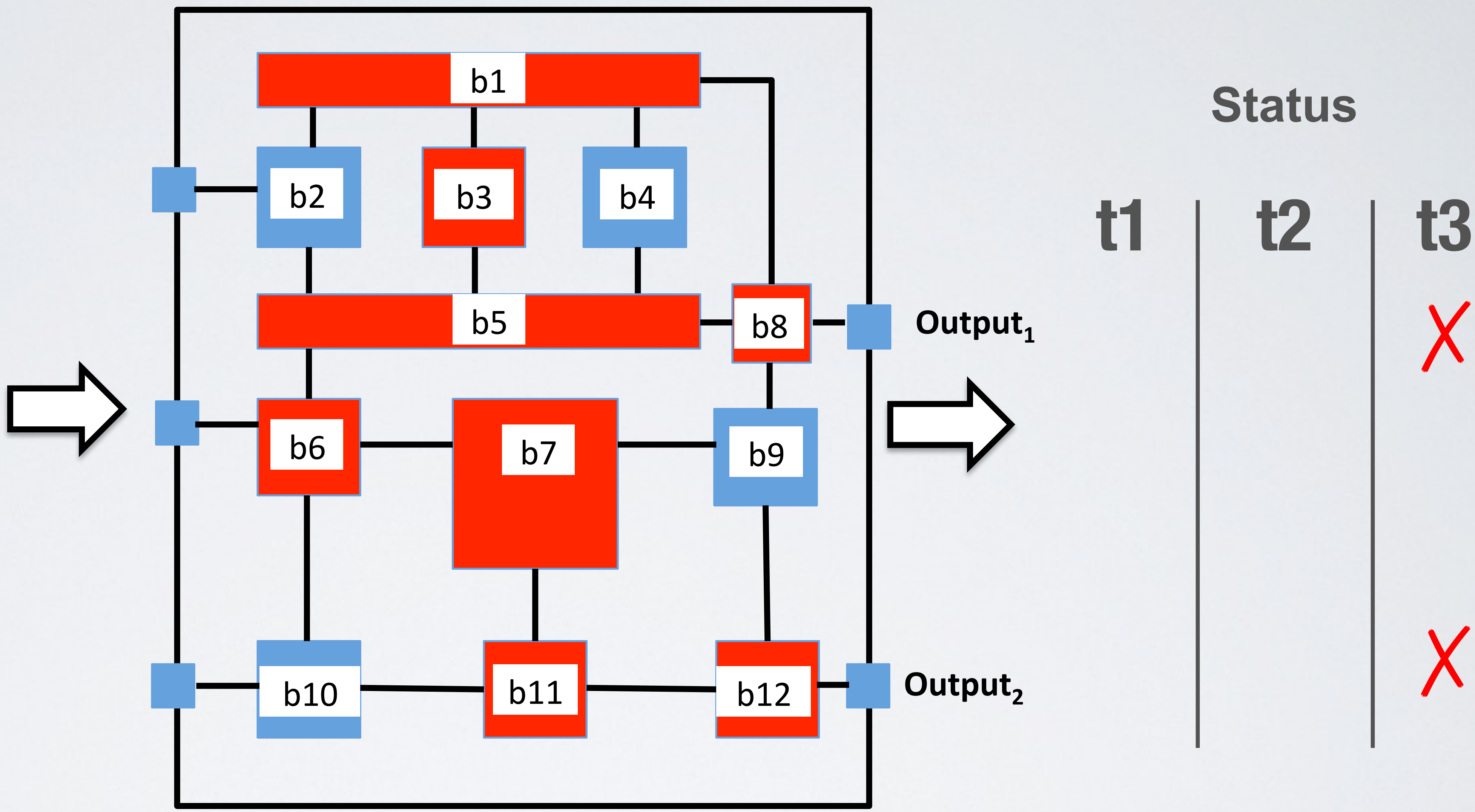
For O_2 :

b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
2	1	0	1	2	2	1	2	1	1	1	2
0	0	0	0	0	0	0	0	0	0	0	0

Not executed
Executed, pas
s
Executed, fail

Statistical Debugging

Test Suite



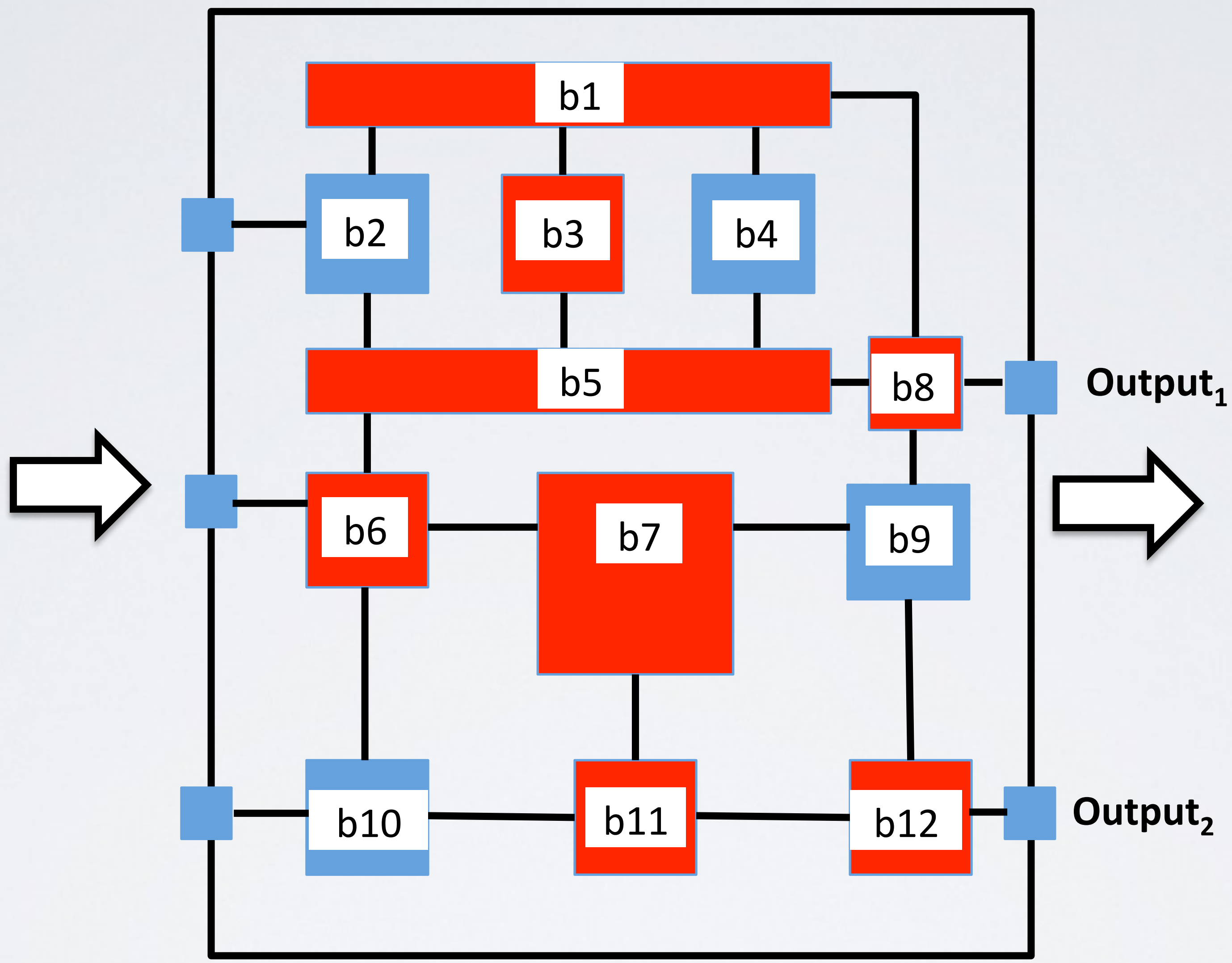
For O_2 :

b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
2	1	0	1	2	2	1	2	1	1	1	2
1	0	1	0	1	1	1	1	0	0	1	1

Not executed
Executed, pas
s
Executed, fail

Statistical Debugging

Test Suite



Status

t1	t2	t3
		✗
		✗

For O_2 :

b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12
2	1	0	1	2	2	1	2	1	1	1	2
1	0	1	0	1	1	1	1	0	0	1	1

- Not executed
- Executed, pass
- Executed, fail

Ranking

- Simulink blocks are ranked according to likelihood of causing output failures

- Engineers inspect Simulink blocks using the generated ranking to identify faulty block(s)

Block	Score
b3	1.0
b7	0.6667
b11	0.6667
b1	0.5
b5	0.5
b6	0.5
b8	0.5
b12	0.5
b2	0.0
b4	0.0
b9	0.0
b10	0.0

Results

- We have developed, SimFL, a tool for automated fault localization of Simulink models

- Our tool is able to help localize multiple faults in Simuilnk models

- SimFL has been applied to three large Simulink models with 400 to 800 blocks containing two to five faults

- Using SimFL engineers need to inspect less than 3% of the model blocks on average to localize faults

Statistical Debugging: Drawbacks

- Faulty blocks may not be ranked high
- Many blocks may have the same score
- Engineers may have to inspect many blocks until they find the faulty block(s)

Ranking	Score
b6	1
b2	0.5
b3	0.5
b5	0.5
b7	0.5
b8	0.5
b9	0.5
b10	0.5
b12	0.5
b13	0.5
b14	0.5
b1	0
b4	0
b11	0

Performance Improvement

- Engineers may fail to find fault(s) after inspecting the top rank blocks
- Our improvements:
  - Extending the test suites with test cases targeted at revealing test cases
  - Identifying heuristics to help engineers know situations where statistical ranking cannot be further refined/improved

Tool Demo

ASTech-SnT collaboration

Topics

- Stability analysis of nonlinear systems
- Analysis of performance and simulation (execution) time of Simulink models

Stability Analysis

- Identifying the parts of the input space of a Simulink model that may lead to violation of system stability requirements
- Identifying root causes of erroneous/unstable behaviors
- Investigating whether the problematic input regions can be reduced by automatic parameter tuning

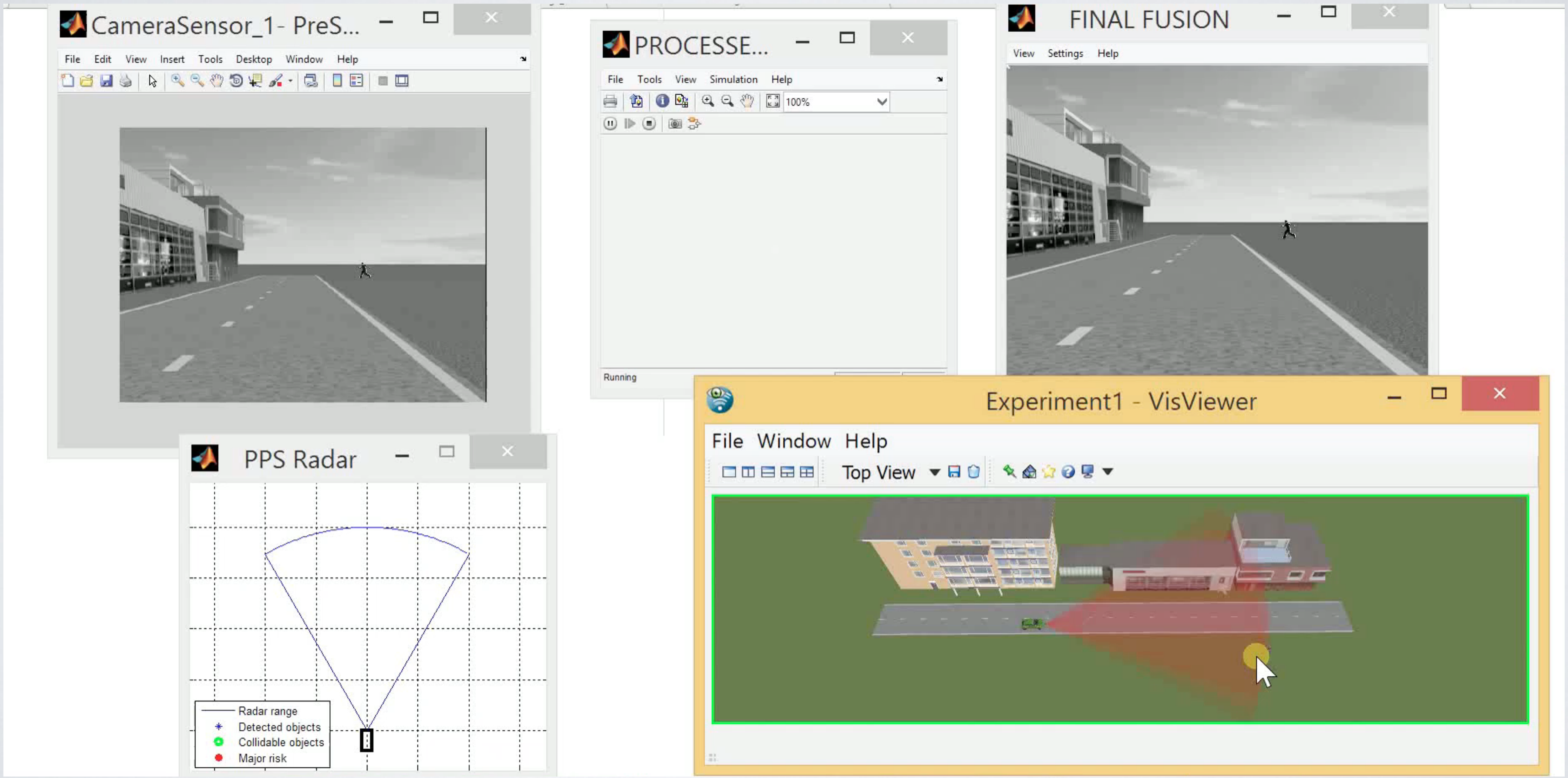
Analysis of Performance and Simulation Time

- Identifying inputs, configurations, or blocks that are responsible for simulation slowdown
- Severe performance degradations after changes in models (regression performance analysis)
- Proposing ways to improve the simulation performance

Automated Testing and Verification of Cyber-Physical Systems

Software Verification and Validation (SVV) Lab
Interdisciplinary Centre for Security, Reliability and Trust (SnT)
University of Luxembourg

Testing via Physics-based Simulation



Testing via Simulation (Limitations)

- Simulation scenarios are generated **manually**
- There are **many** simulation scenarios
- **No guidance** as to which scenarios should be selected to test the system

Our Approach

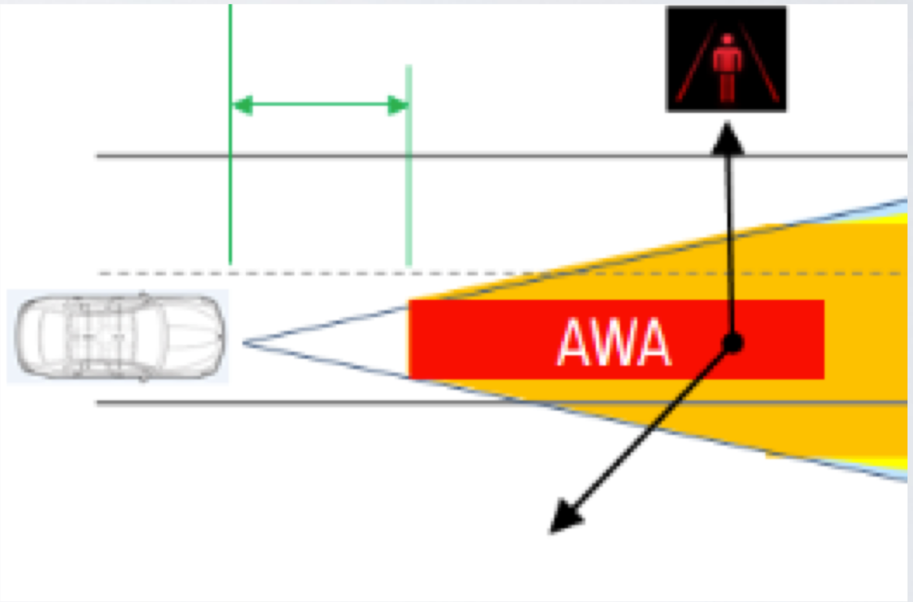
- **Automated testing of complex ADAS via physics-based executable models of these systems and their environments**

Challenges	Our solution
Test input is large	**Metaheuristic search** to focus testing on **worst case/critical behaviors**
Simulation takes time	**Surrogate models** to predict the simulation outcome **without running** simulations

PeVi Requirement

The PeVi system shall detect any pedestrian located in the Acute Warning Area (AWA) of a vehicle

- Test objectives critical aspects:
  - Time-To-Collision (TTC) is small
  - The pedestrian is located near the car
  - The pedestrian is at the boundary of the AWA



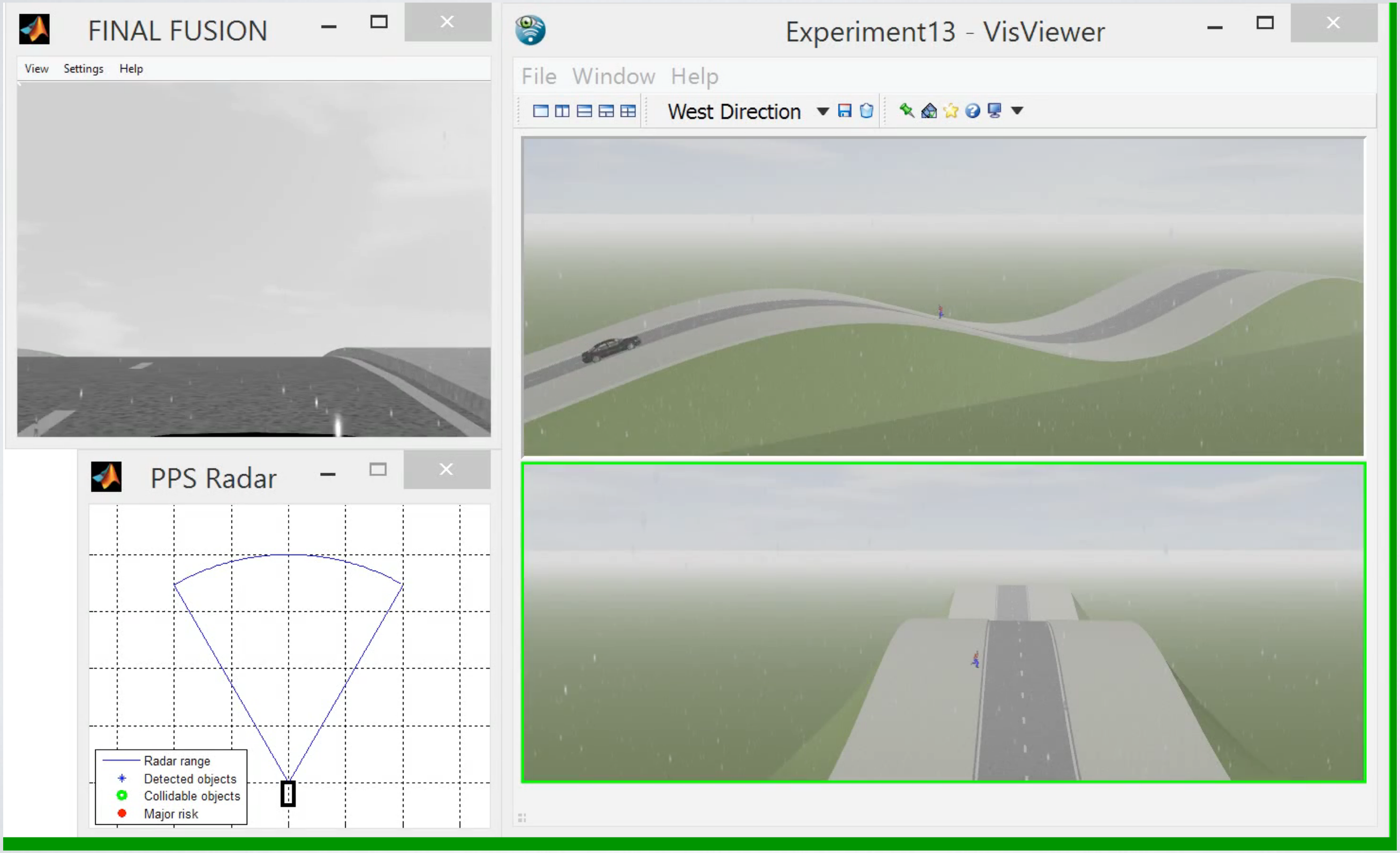
Our Search-based Test Generation

- We use **multi-objective** search algorithm to generate test cases
- Three objectives: Minimum Distance to Car **min{D(P/Car)}**, Minimum Time To Collision **Min{TTC}**, and Minimum Distance to AWA **Min{D(P/AWA)}**
- Input a vector (car-speed, person-speed, person-position (x,y), person-orientation)
- Each search iteration calls simulation to compute objectives

Evaluation

- Given the same execution time, our approach is able to produce higher quality solutions than baseline methods (Random Search)
- We provided engineers with **20 scenarios representing risky situations** (no detection in AWA, collision/no detection) by varying weather conditions, roadside objects and ramped/curved road
- Scenarios helped engineers **identify** several **critical behaviors** of PeVi that have not been previously identified by manual simulation
- The scenarios are available at: **https://sites.google.com/site/testingpevi**

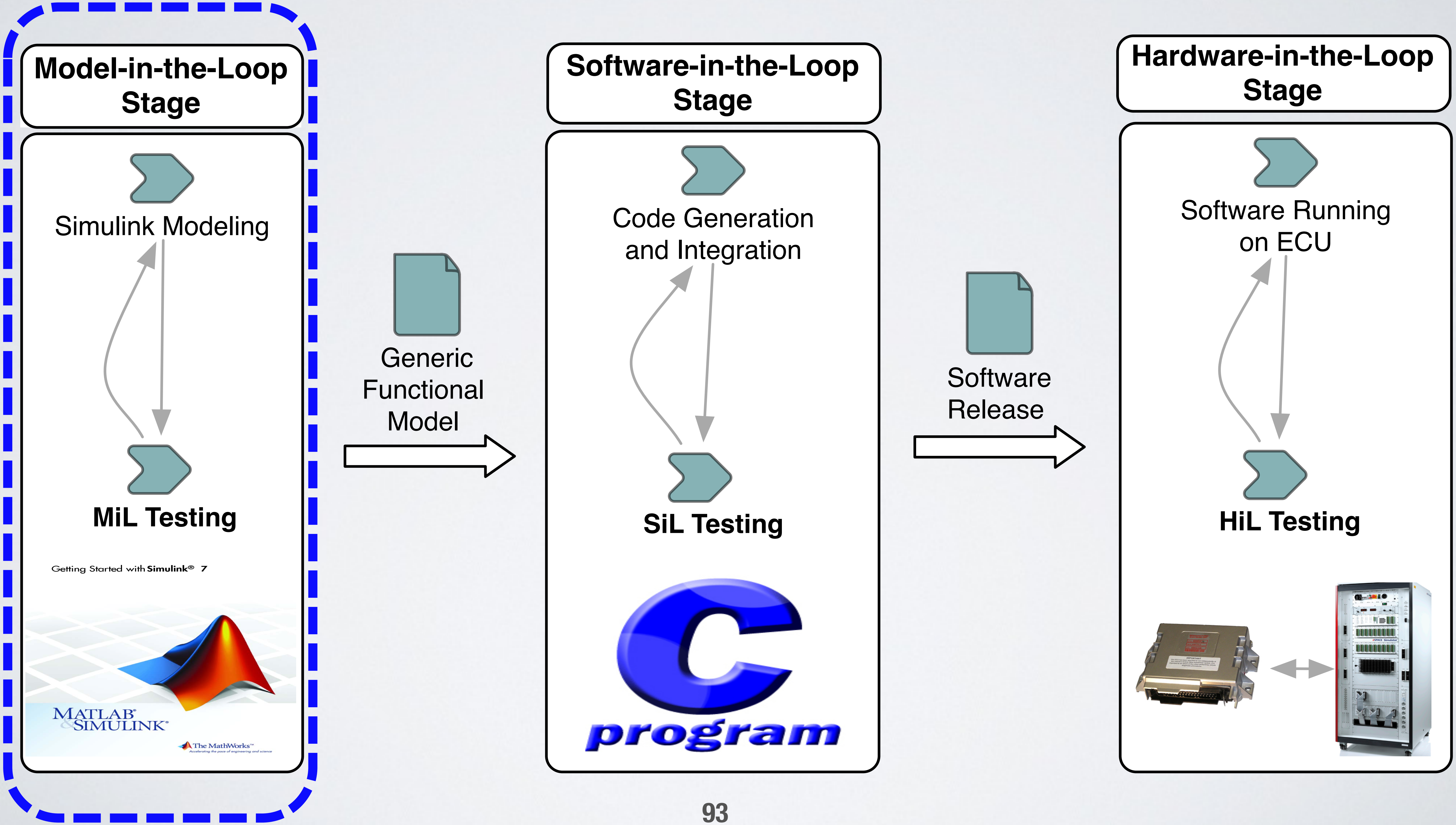
An example critical scenario



Conclusion

- An automated effective testing approach for ADAS
- Formulated the generation of critical test cases as a multi-objective search problem using the NSGA2 algorithm
- Improved the search performance, while maintaining the accuracy, using surrogate models based on neural networks
- Generated some critical scenarios: no detection in the AWA, or collision and no detection

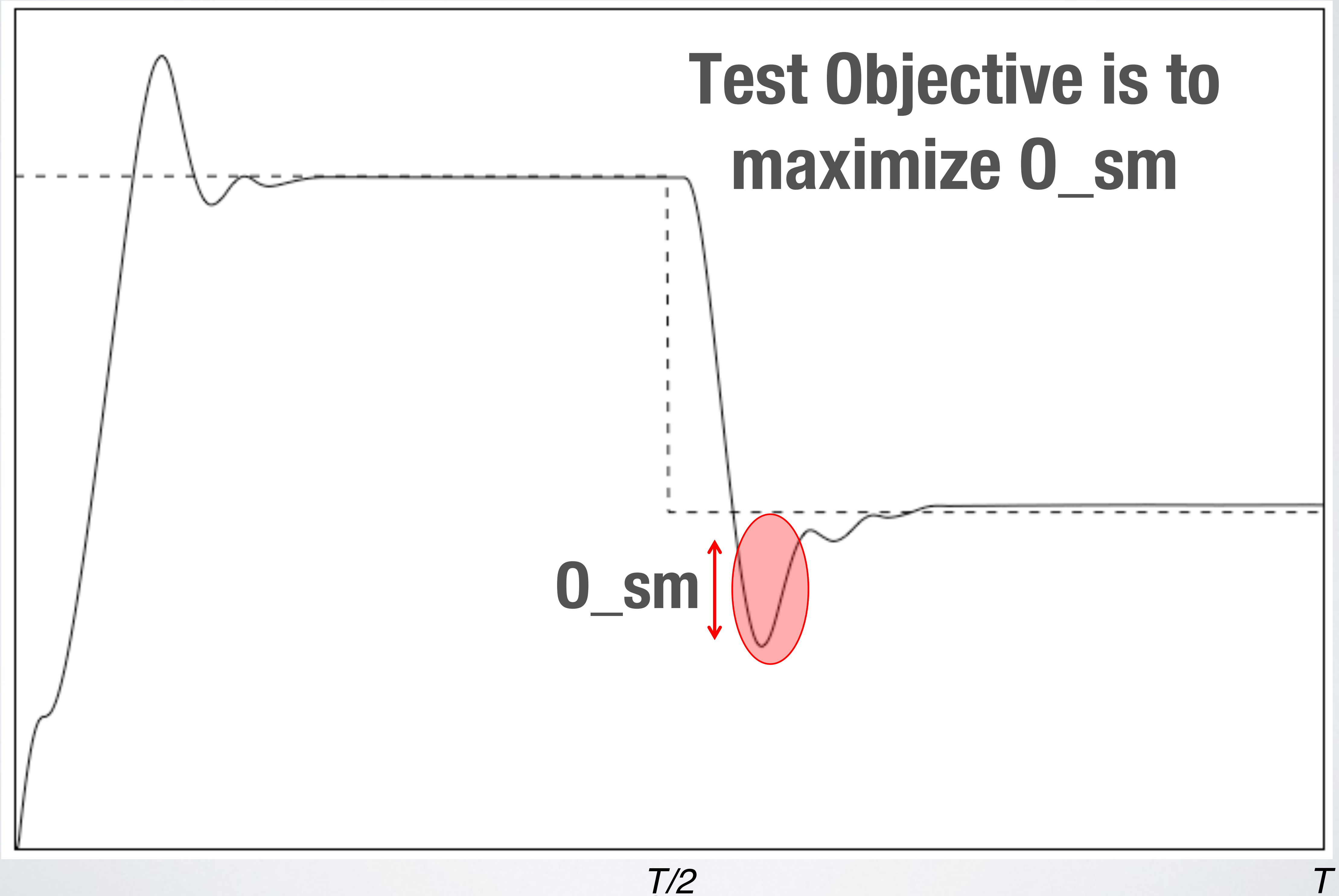
Model-based development of control systems



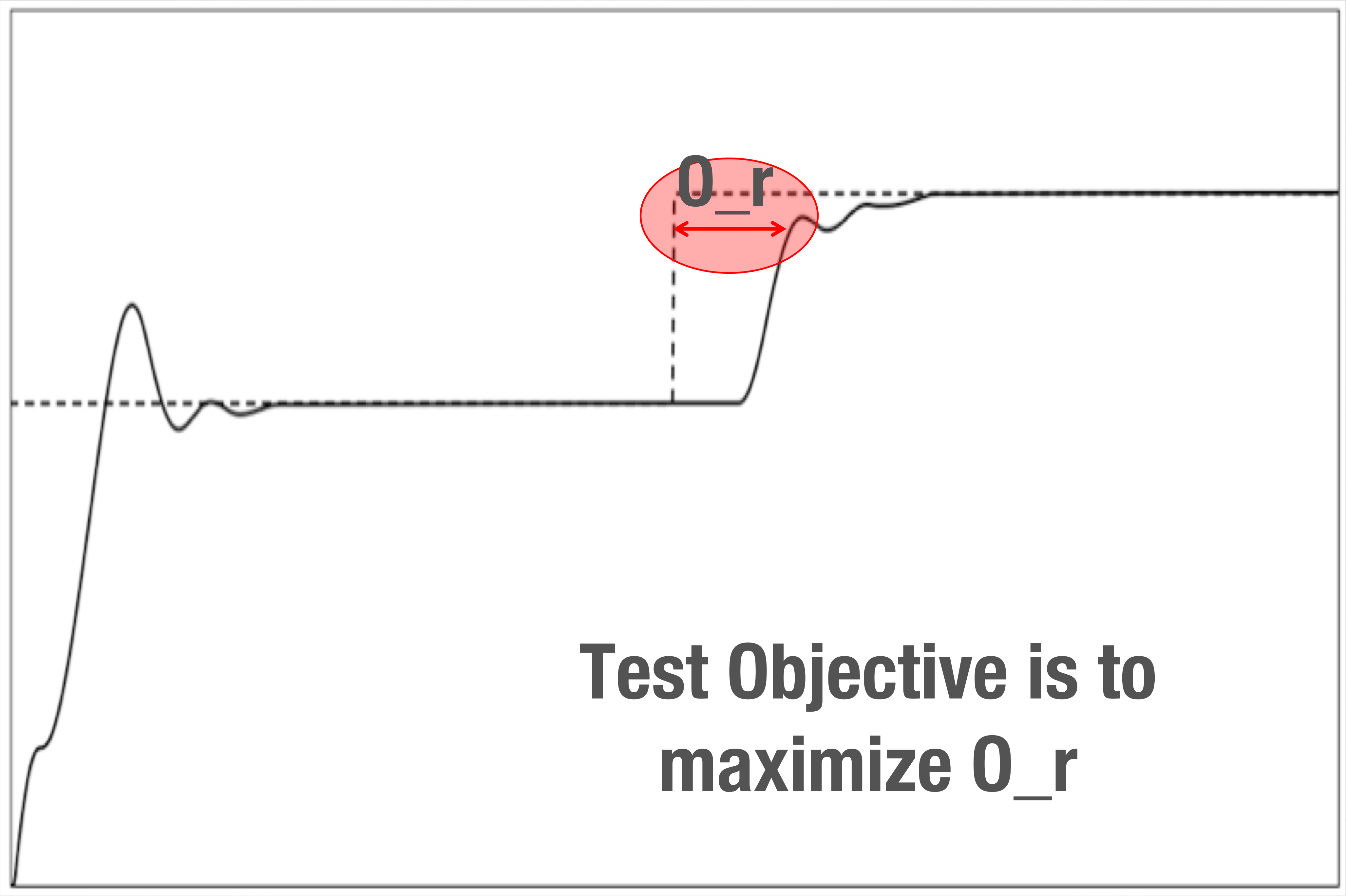
Simulink Testing Challenges

- **Complex** models with **several hundreds of blocks**
- Mix of **continuous and discrete** behaviour
- Captures both **hardware/physical components** and **software/algorithms**
- Mix of **fixed-point** and **floating point** computations

Controller Properties: Smoothness

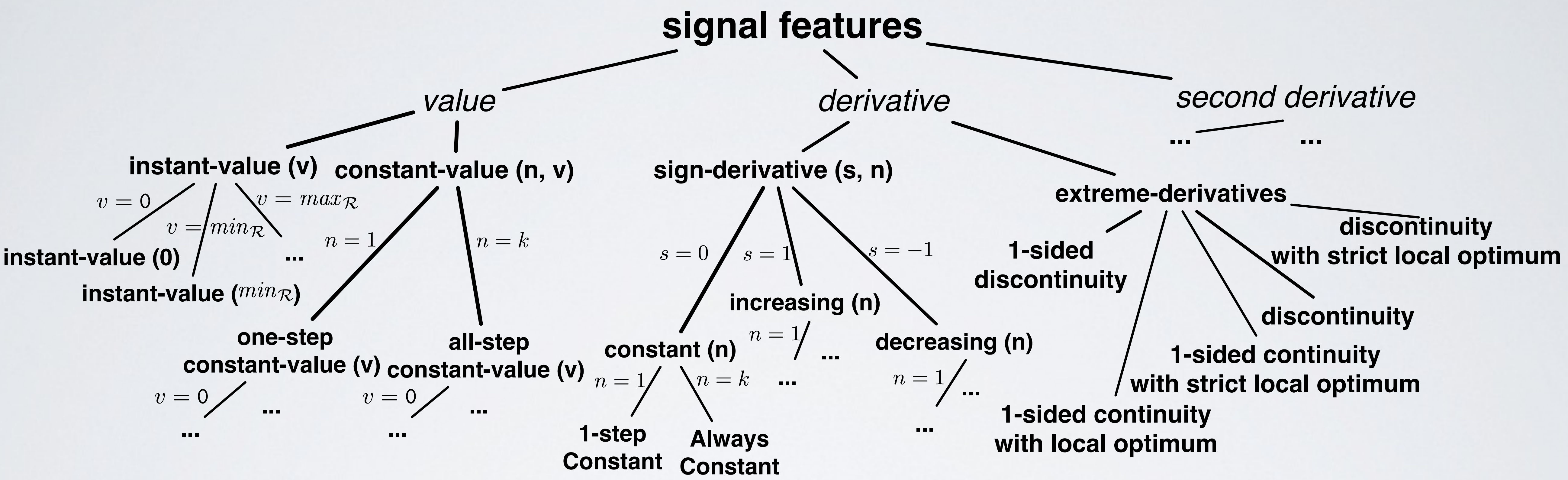


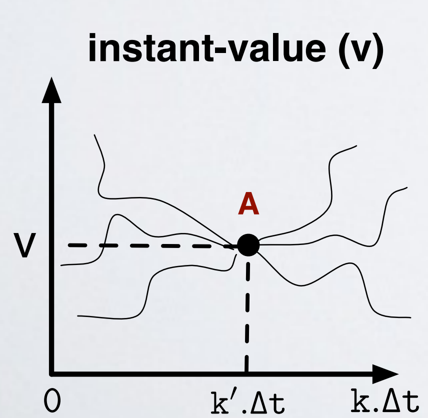
Controller Properties: Responsiveness

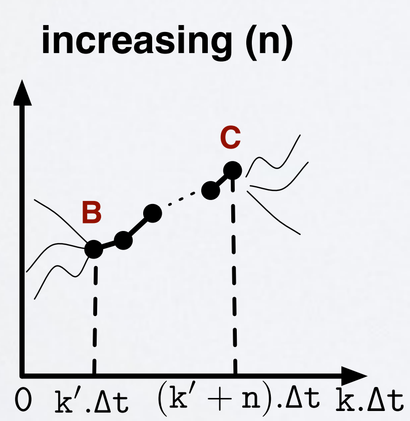


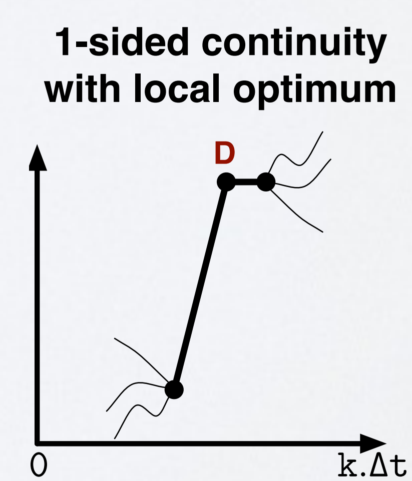
Signal Features Classification

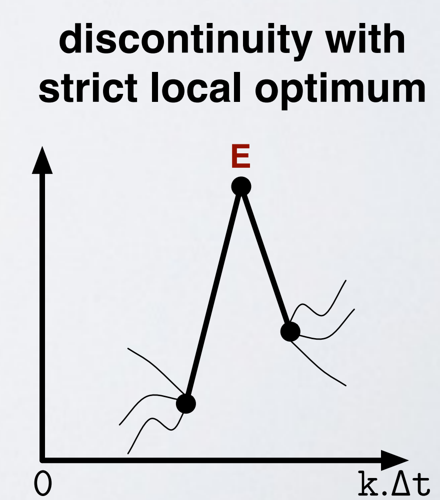
- We define a set of basic features characterizing distinguishable signal shapes









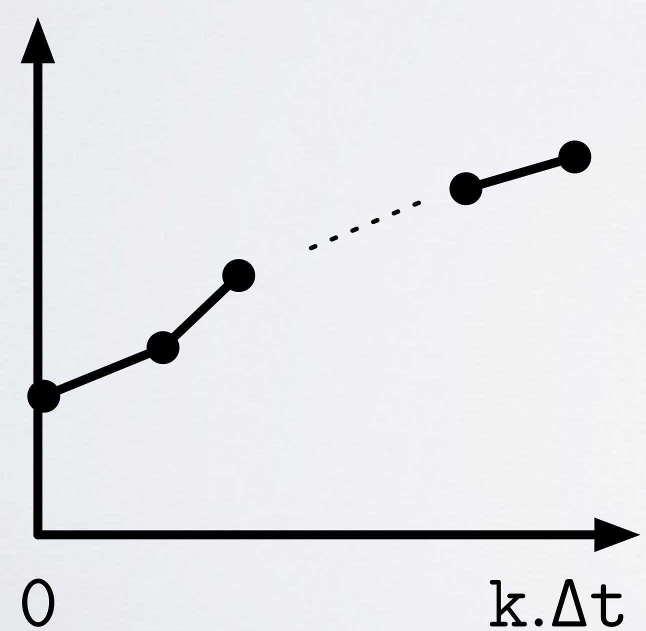


Feature Functions

- For each feature *f* in our feature classification, we define a feature function F_f

- For a signal sg, the value of F_f(sg) quantifies the similarity between shapes of the signal sg and the feature f

Always increasing



$$F_f(sg) = \sum_{i=1}^{K}((sg(i \cdot \Delta t) - sg((i-1) \cdot \Delta t))) - |sg(i \cdot \Delta t) - sg((i-1) \cdot \Delta t))|)$$

The higher the value of F_f(sg), The more similar sg is to f

Output Diversity
Test Suite Generation Algorithm

Test Suite Gneration Algroihtm

P $\leftarrow$ **Initial number of segments**

Best $\leftarrow$ **TS** $\leftarrow$ **Initial test suite**

Repeat

  TS $\leftarrow$ ***Tweak* (TS, P)**

  Execute (TS)

  If *O* (TS) > *O* (Best)

    Best $\leftarrow$ **TS**

  If (*Coverage has reached a plateau less than 100%*)

    P $\leftarrow$ **P + 1**

Until *maximum resources spent*

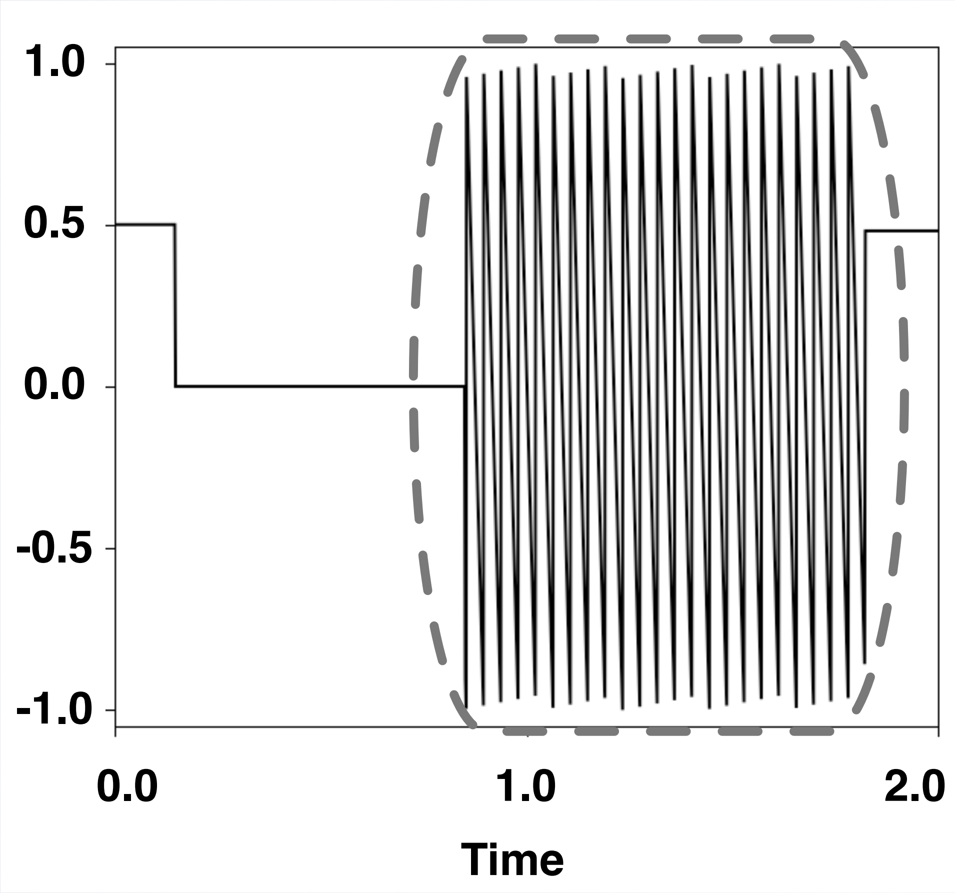
Return TS

O is either O_v or O_f

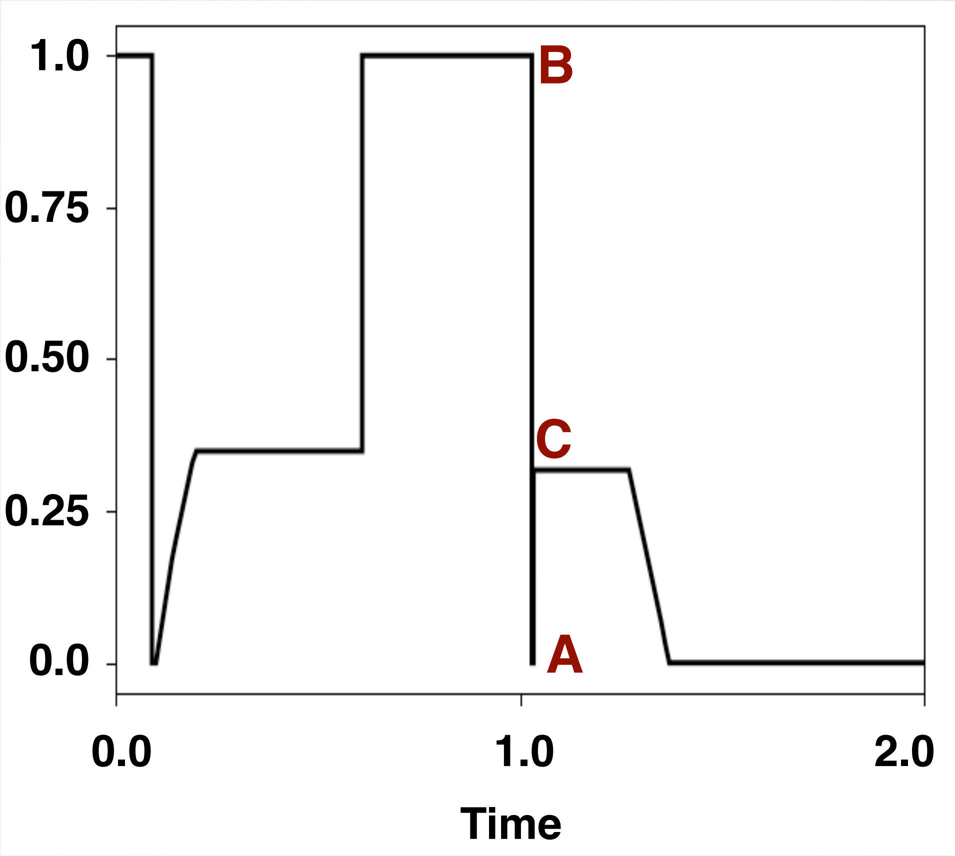
A whole test suite generation algorithm

Failure patterns for Continuous Outputs

Instability



Discontinuity



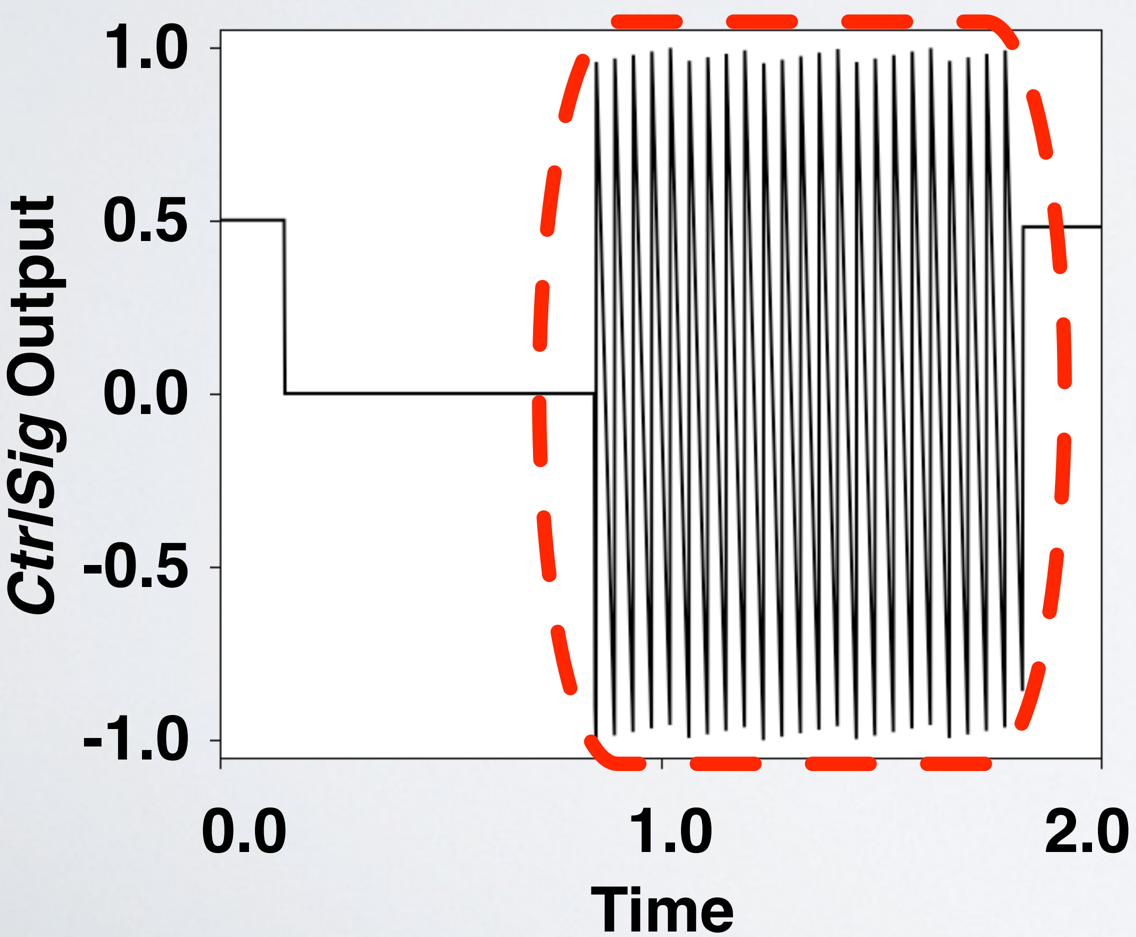
SimCoTest Maturity

- Hands-on tutorial to ten Delphi engineers:
  - “SimCoTest is useful for early stages of controller design to identify and detect design flaws.”
- We are currently applying SimCoTest to a number of newly developed (cutting edge) Delphi Controller models to help engineers with testing/verification, and to better demonstrate practical usefulness of SimCoTest

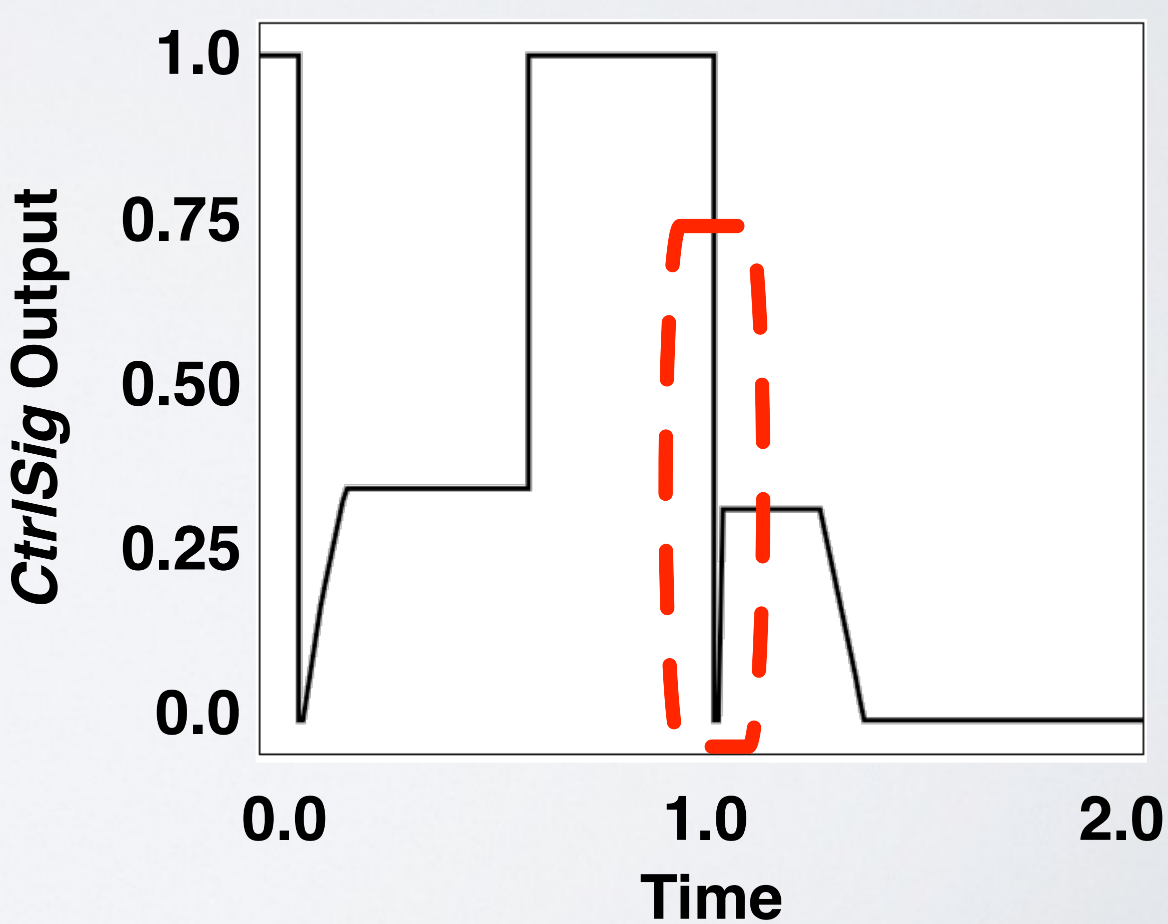
Failure-based Test Generation

- **Maximizing the likelihood of presence of specific failure patterns in output signals**

Instability



Discontinuity



Comparing SimCoTest with Simulink Design Verifier

Fault No.	1	2	3	4	5	6	7	8	9	10	11
#OurApproach	20	16	20	11	5	20	14	17	11	20	4
*SLDV	No	No	No	No	No	No	No	No	No	No	No

Fault No.	12	13	14	15	16	17	18	19	20	21	22
#OurApproach	5	14	2	20	20	20	20	20	20	15	15
*SLDV	No	No	No	No	No	YES	No	No	No	No	No

 # Number of Fault Revealing Runs of our algorithm (Out of 20)

*** Did SLDV reveal the fault? (Yes/No)**

The only fault found by SLDV (fault 17), was also found by SimCoTest with very high probability