Verified Artificial Intelligence and Autonomy

Sanjit A. Seshia

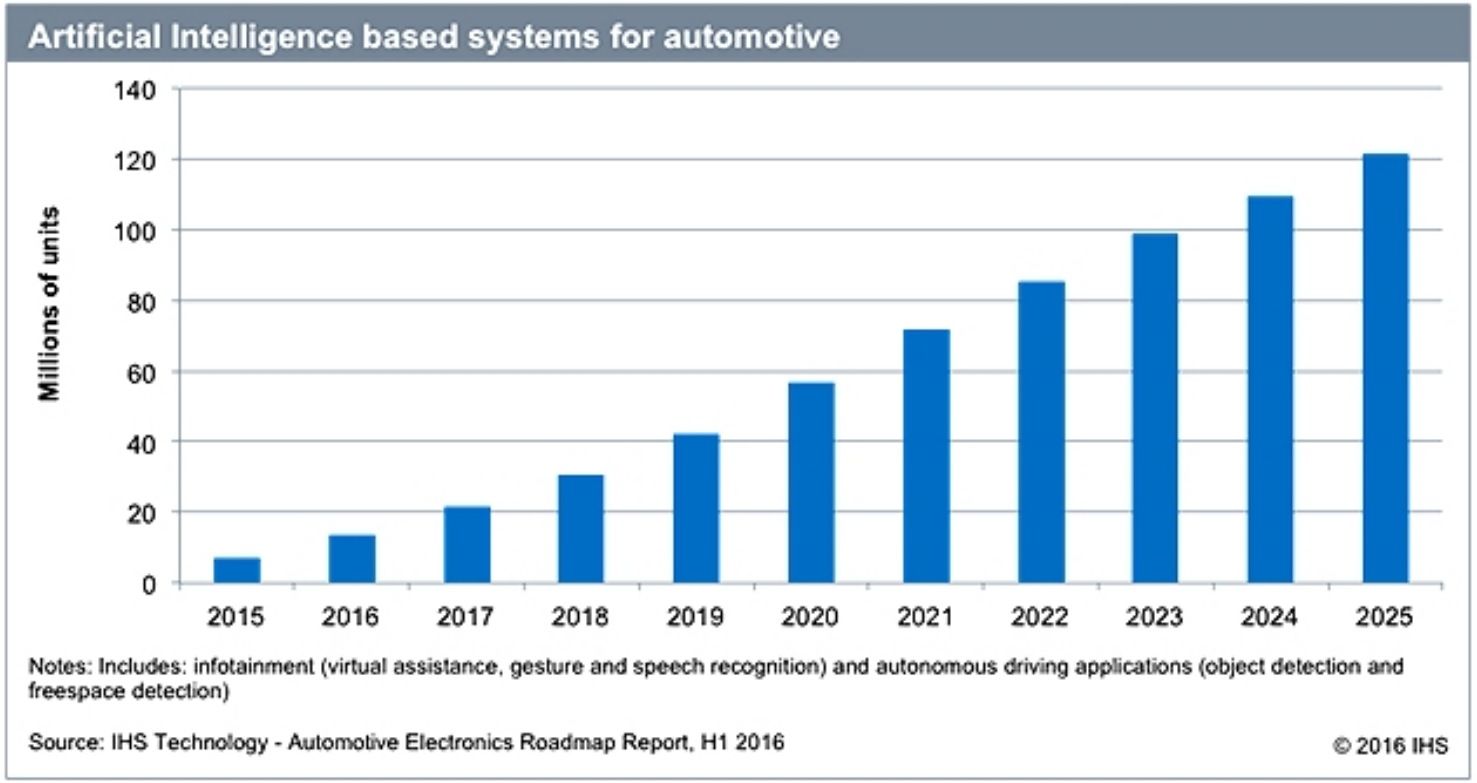
Professor

EECS Department, UC Berkeley

NFM 2020 Keynote
May 13, 2020

VeHiCaL
http://vehical.org

Growing Use of Machine Learning/Artificial Intelligence in Safety-Critical Autonomous Systems



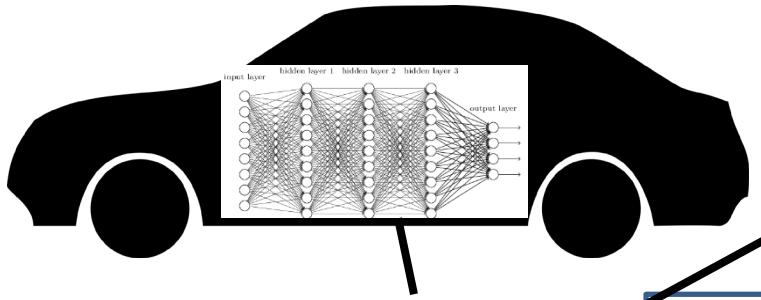
Growing Concerns about Safety:

- Numerous papers showing that *Deep Neural Networks can be easily fooled*
- *Fatal accidents* involving potential failure of AI/ML-based perception systems in self-driving cars

Can we address the Design & Verification Challenges of AI/ML-Based Autonomy with Formal Methods?

Challenges for Verified AI

S. A. Seshia, D. Sadigh, S. S. Sastry.
Towards Verified Artificial Intelligence. July 2016. https://arxiv.org/abs/1606.08514.



System **S**

Environment **E**

Specification **φ**

Does S || E satisfy φ?

YES [+ proof]

NO [+ counterexample]

Need to Search Very High-Dimensional Input and State Spaces

Design Correct-by-Construction?

Need Design Principles for Verified AI

Challenges	Principles
1. Environment (incl. Human) Modeling →	
2. Formal Specification →	
3. Learning Systems Representation →	?
4. Scalable Training, Testing, Verification →	
5. Design for Correctness →	

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*. July 2016. https://arxiv.org/abs/1606.08514.

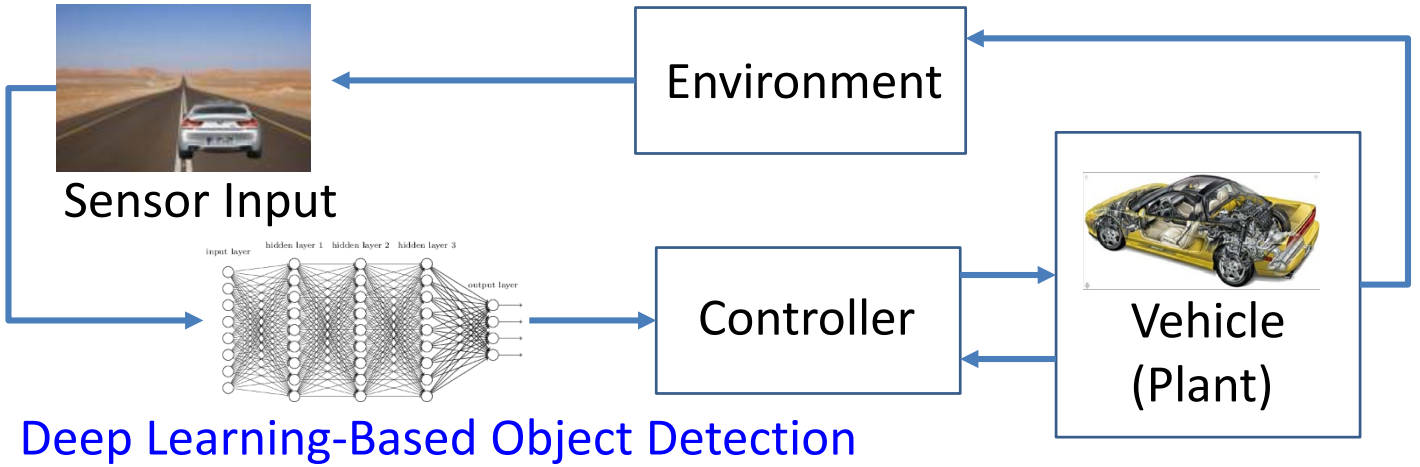
Outline

- Challenges for Verified AI

- Formal Methods for Cyber-Physical Systems (CPS) with AI/ML Components
  - Specification, Verification, Synthesis
  - Autonomous Vehicles (ground and air)
  - Deep Learning for Perception

- Principles for Verified AI
  - Summary of Ideas
  - Future Directions

Example: Automatic Emergency Braking System (AEBS) using Deep Learning for Perception



- Goal: Brake when an obstacle is near, to maintain a minimum safety distance
- Models: Controller, Plant, Env modeled in a software-in-the-loop simulator
  (created in Matlab/Simulink, Udacity, Webots, CARLA, LGSVL, …)
- Perception: Object detection/classification system based on deep neural networks
  - Inception-v3, AlexNet, … trained on ImageNet
  - more recent: squeezeDet, Yolo, … trained on KITTI

[Dreossi, Donze, Seshia, "Compositional Falsification of Cyber-Physical Systems with Machine Learning Components", NASA Formal Methods (NFM), May 2017.]

Challenge: Formal Specification

Principle: Start at the System Level (i.e. Specify *Semantic Behavior* of the Overall System)

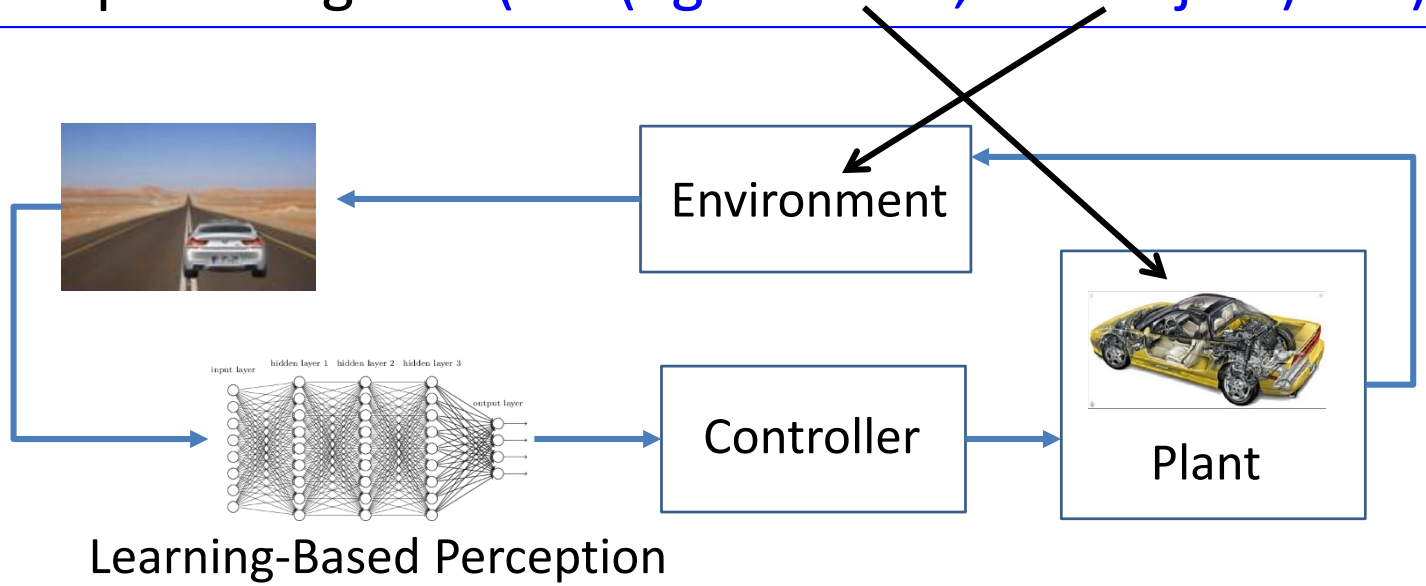
Our Approach: Start with a System-Level Specification

✗ "Verify the Deep Neural Network Object Detector"

✓ "Verify the System containing the Deep Neural Network"

Formally Specify the *End-to-End Behavior* of the System

Temporal Logic: **G** (*dist*(ego vehicle, env object) > Δ)



Property does not mention inputs/outputs of the neural network

Do We Need to Formally Specify **ML Component** Requirements?

Sometimes…

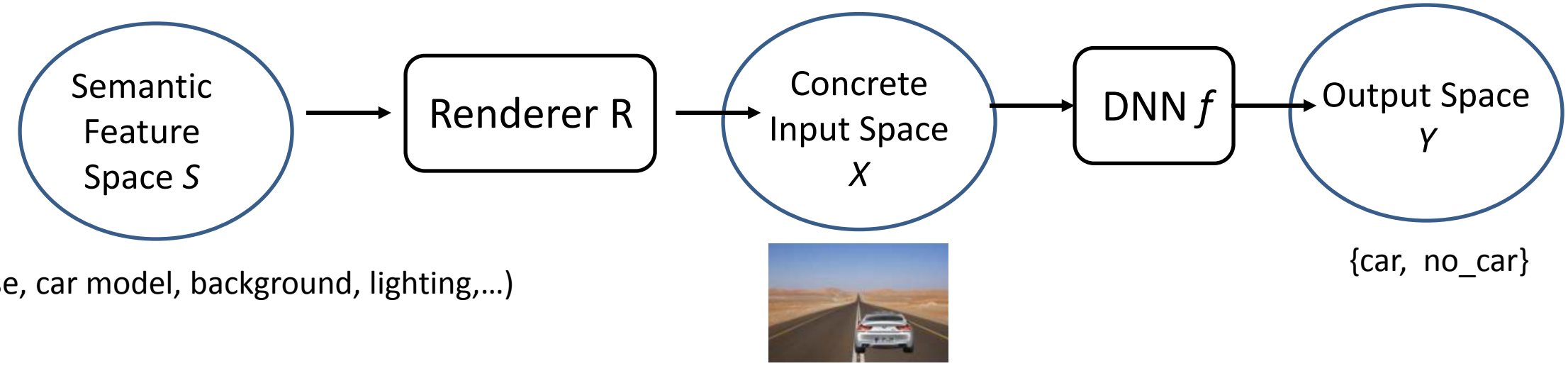
1. When Component Specifications are Meaningful
   - ML-based Planners/Controllers
   - Semantic Robustness*, Input-Output Relations, Monotonicity, Fairness, etc.

2. For Compositional/Modular Analysis
   - Derive component specifications from system-level specifications

[S. A. Seshia, et al., "Formal Specification for Deep Neural Networks", ATVA 2018.]

[Dreossi, Ghosh, Sangiovanni-Vincentelli, Seshia, "A General Formalization of Robustness for Deep Neural Networks", VNN'19]

Semantic Robustness



Semantic Robustness:

$$[\, s \approx_S s' \wedge \mathrm{R}(s) = x \wedge \mathrm{R}(s') = x' \,] \Rightarrow f(x) \approx_Y f(x')$$

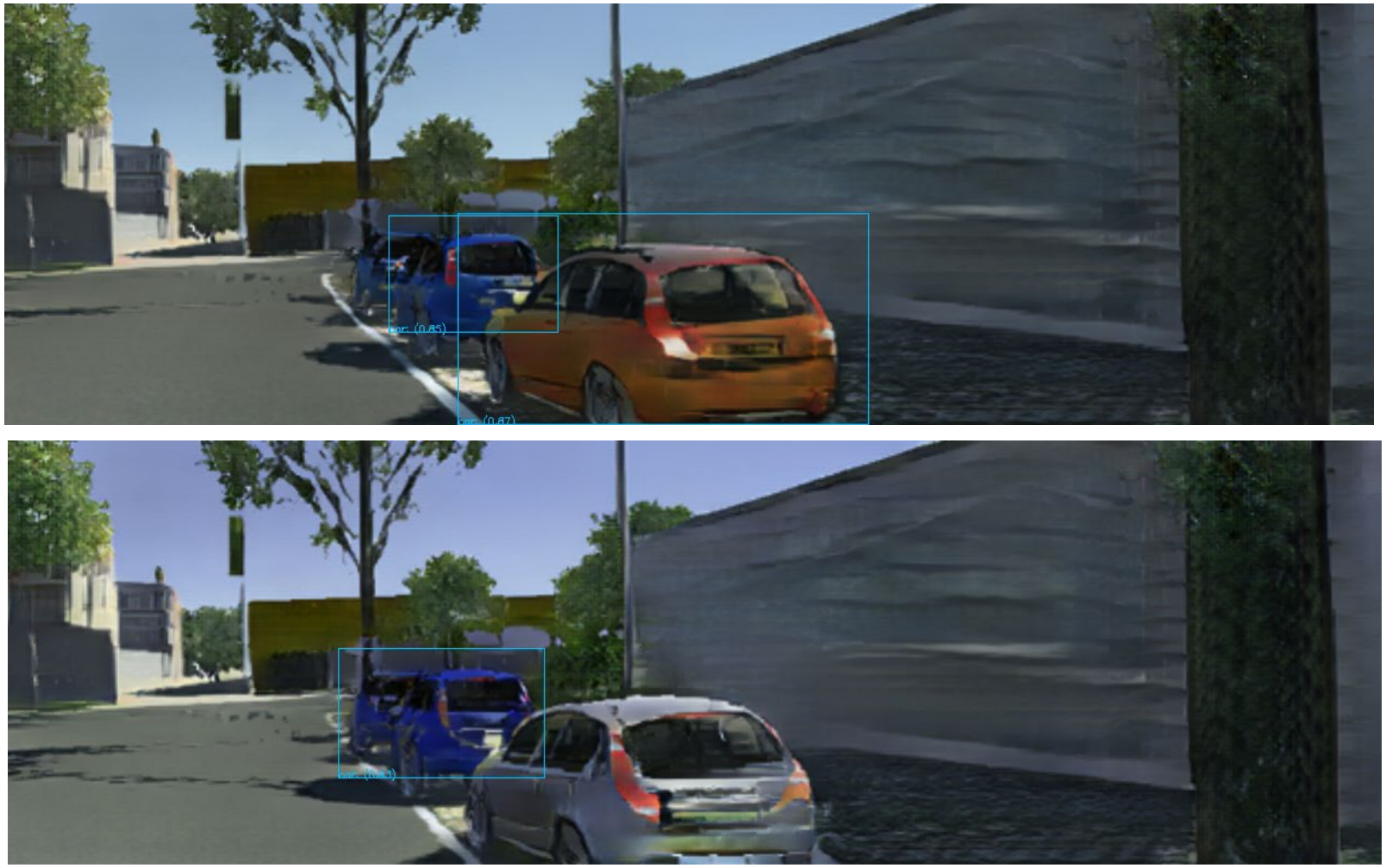
Can apply techniques from standard adversarial analysis, provided R is differentiable

[S. A. Seshia, et al., "Formal Specification for Deep Neural Networks", ATVA 2018.]

Semantic Adversarial Analysis with Differentiable Rendering

SqueezeDet NN for object detection on Virtual KITTI data set

Uses 3D-SDN differentiable renderer [Yao et al. NeurIPS'18] and FGSM on semantic feature space



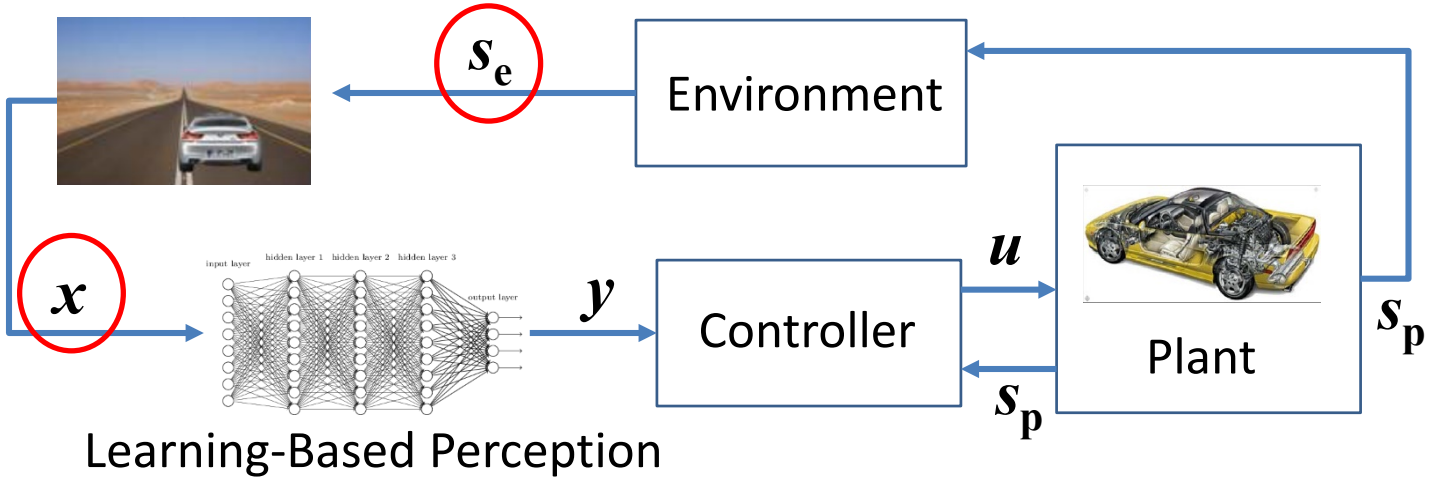
[Jain, Wu, Chandrasekharan, Chen, Jang, Jha, Seshia, 2019]

Challenge: Scalability of Verification

Principle: Compositional Simulation-Based Verification (Falsification)

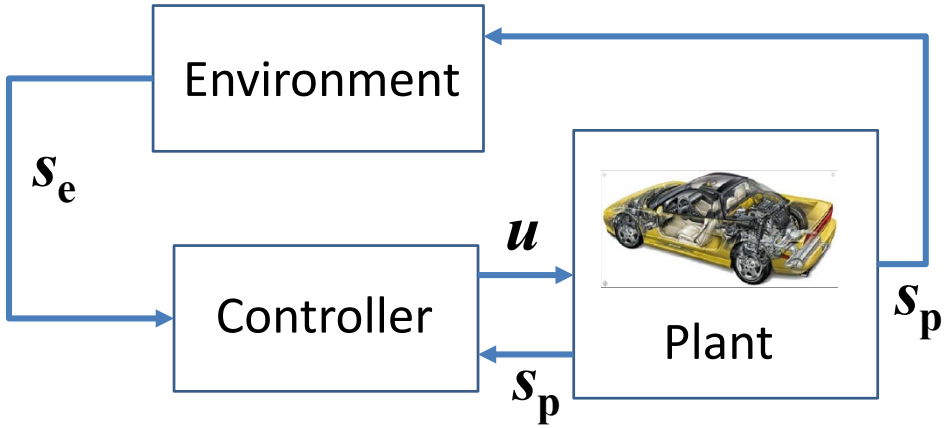
T. Dreossi, A. Donze, and S. A. Seshia. *Compositional Falsification of Cyber-Physical Systems with Machine Learning Components*, In NASA Formal Methods Symposium, May 2017. (Extended version: Journal of Automated Reasoning, 2019.)

Automotive system with ML-based perception (CPSML)



Challenge: $s_e << x$

Traditional closed-loop model (CPS)



Our Approach: Three Key Ideas

1. Reduce CPSML falsification problem to combination of CPS falsification and ML analysis by *abstraction*

2. Simulation-based *temporal logic falsification* for CPS model
   - Scalable technology already used on production CPS

3. *Semantic feature space* analysis of ML component
   - *Derive constraints* on input feature space of neural network (pixels) from semantic constraints on environment model

[Dreossi et al., NFM'17, JAR'19; Dreossi et al., CAV'18]

Simulation-based Falsification: Logical Formulas to Objective Functions

- Use Temporal Logics with Quantitative Semantics (STL, MTL, etc.)

- Example:

  $\boldsymbol{G}_{[0,\tau]}$(dist(vehicle, obstacle) > δ)

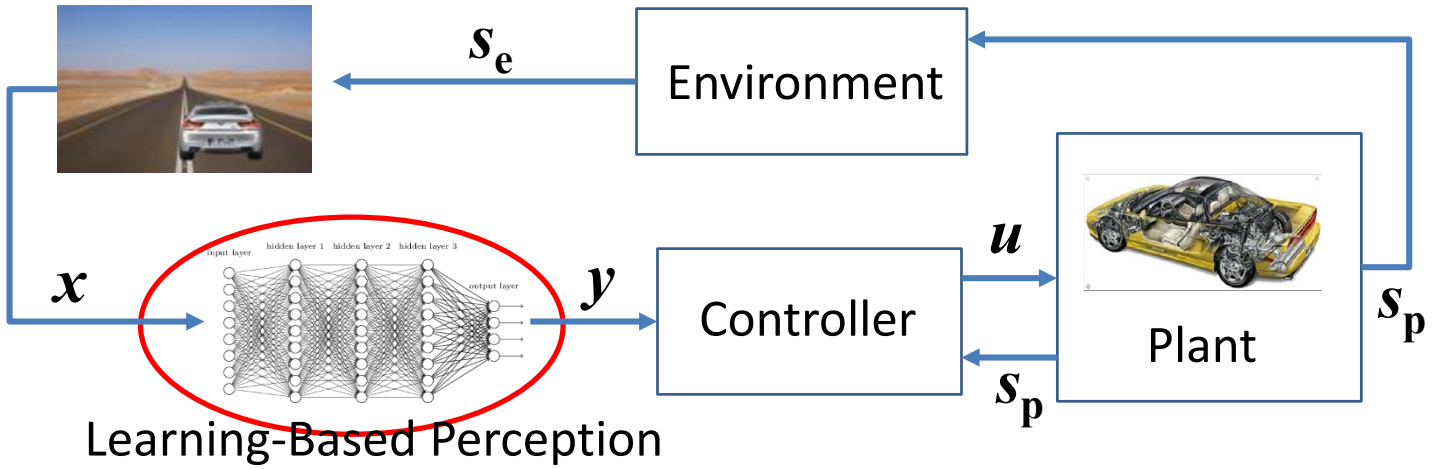
  

  $\inf_{[0,\tau]}$ [dist(vehicle, obstacle) - δ]

- Verification → Optimization

Need *Compositional* Falsification of CPSML

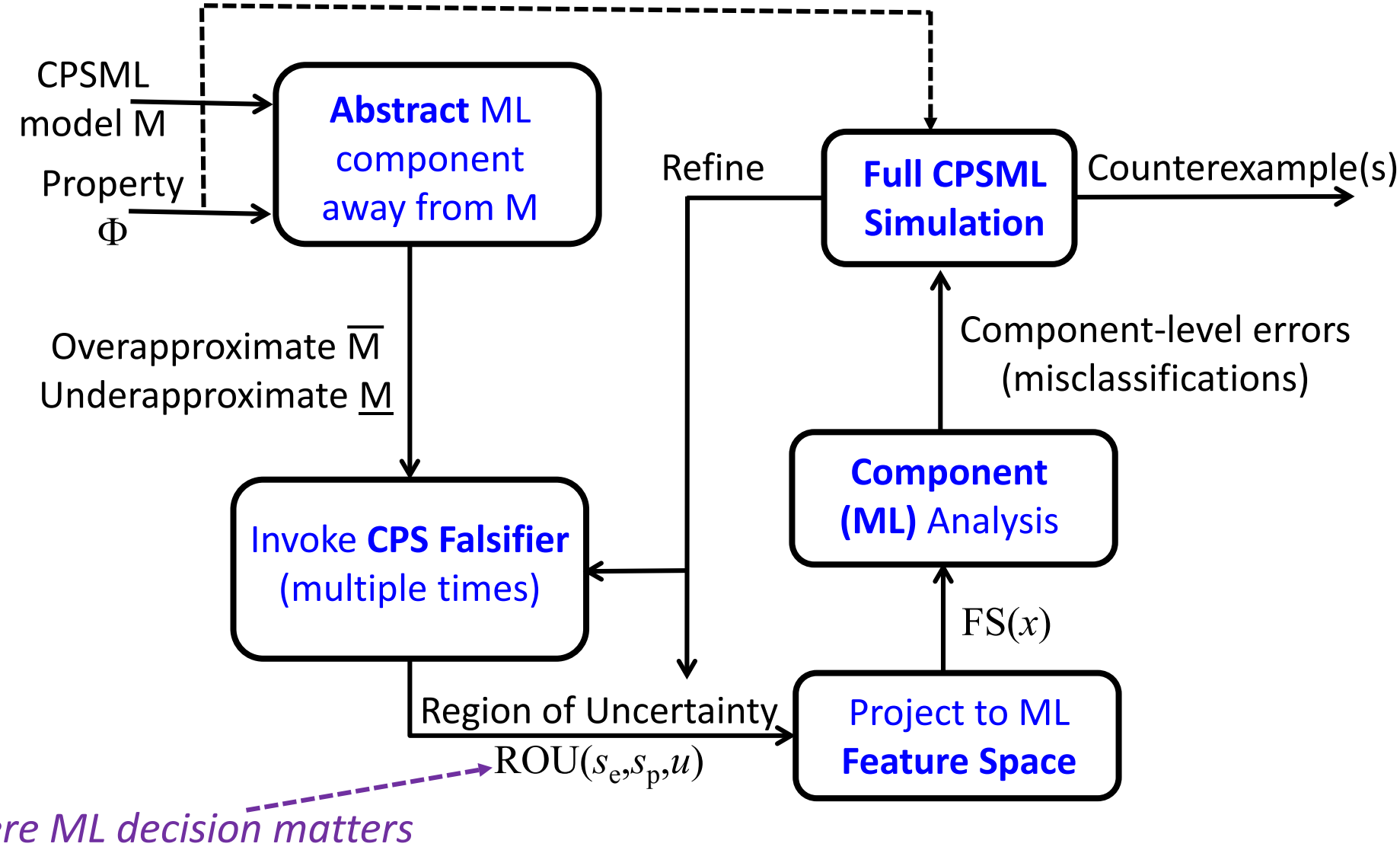
- *Challenge:* Very High Dimensionality of Input Space!
- Standard solution: Use *Compositional (Modular)* Verification



- However: *no formal spec*. for neural network!
- Compositional Verification *without* Compositional Specification?!!

(see [Seshia, UCB/EECS TR 2017])

Compositional Approach: Combine Temporal Logic CPS Falsifier with ML Analyzer

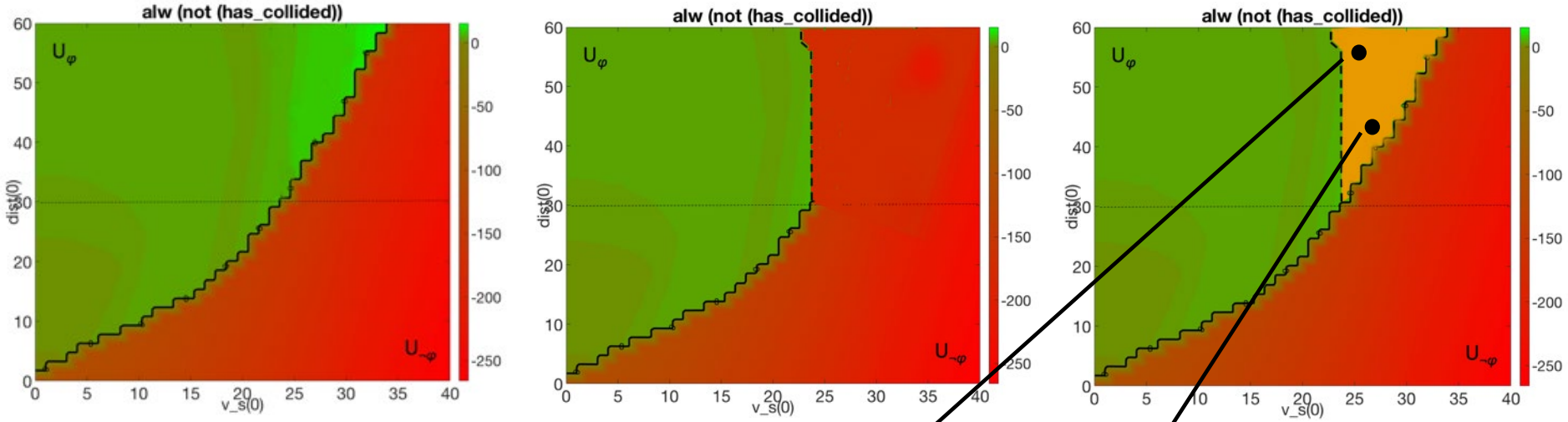


[Dreossi, Donze, Seshia, NFM 2017]

Identifying Component-Level Input Constraints (ROU) for Automatic Emergency Braking System

Green → environments where system-level safety property is satisfied

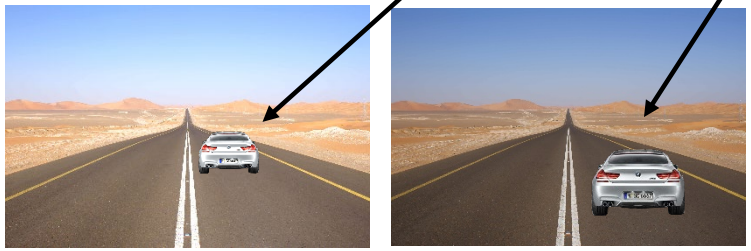
Underapproximate $\underline{M}$ Overapproximate $\overline{M}$



ML always correct

ML always wrong

Potentially unsafe region depending on ML component (yellow)



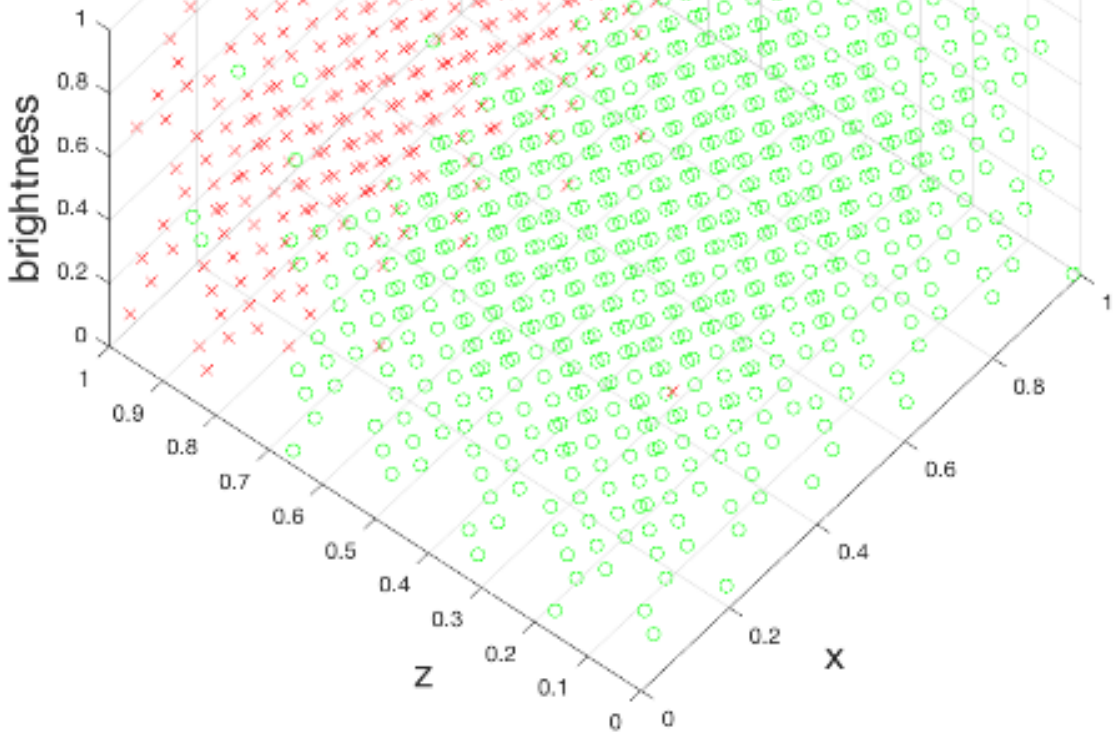
[Dreossi et al., NFM'17, JAR'19; Dreossi et al., CAV'18]

Result on AEBS Example

This misclassification not of concern

Misclassifications

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

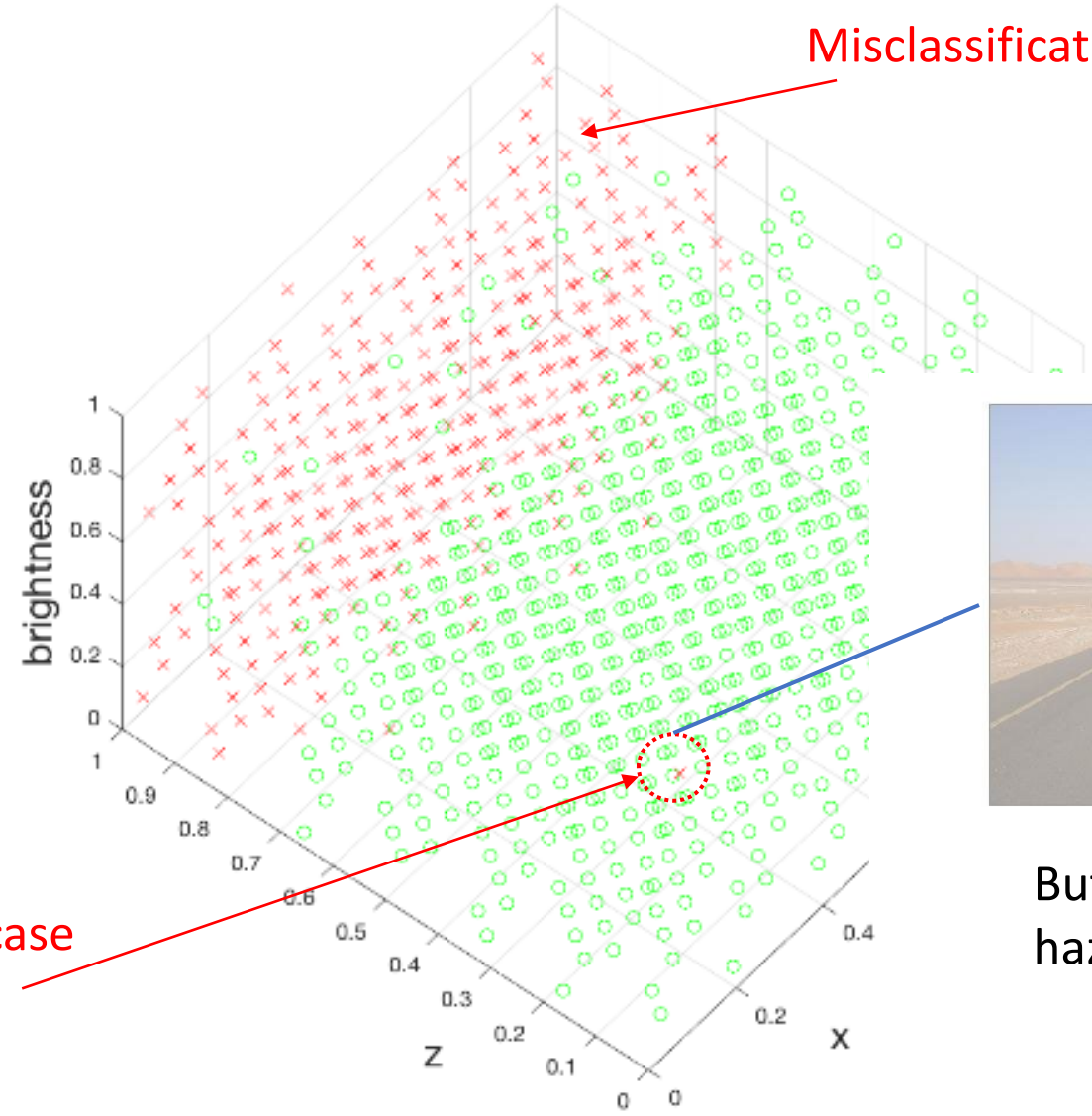


Sample image

[Dreossi, Donze, Seshia, NFM 2017; J. Automated Reasoning 2019]

Result on AEBS Example

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



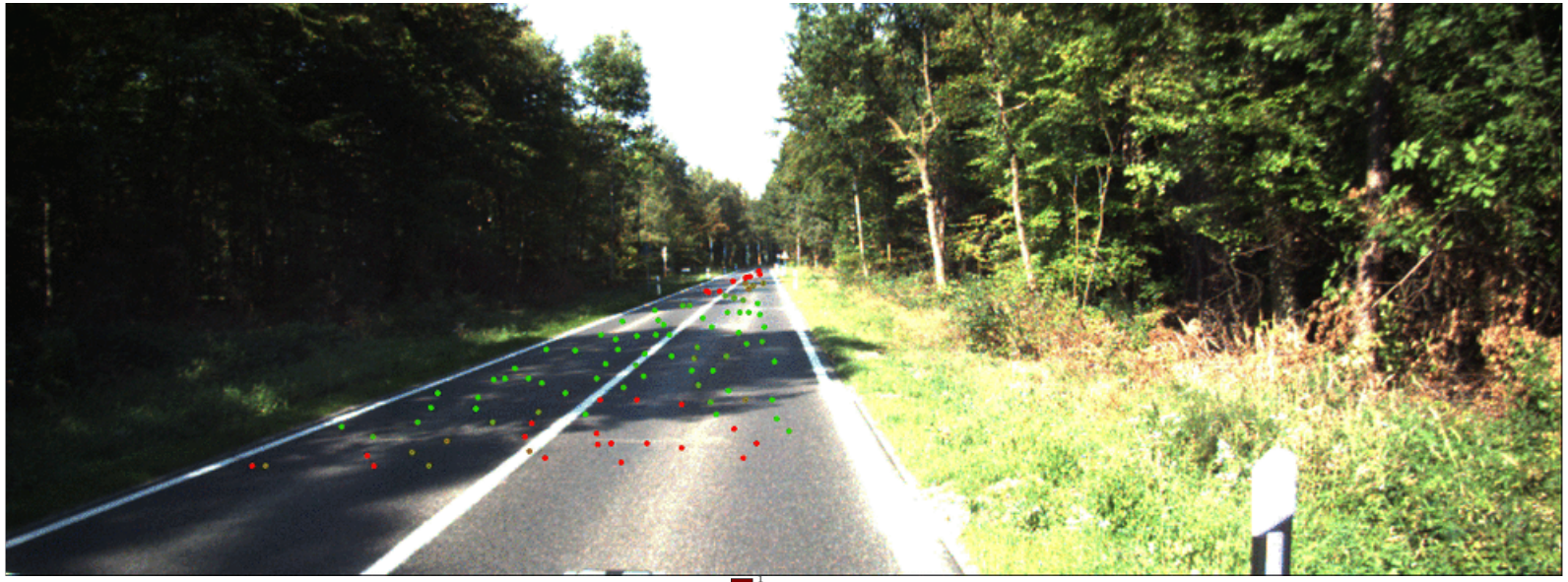
Corner case Image

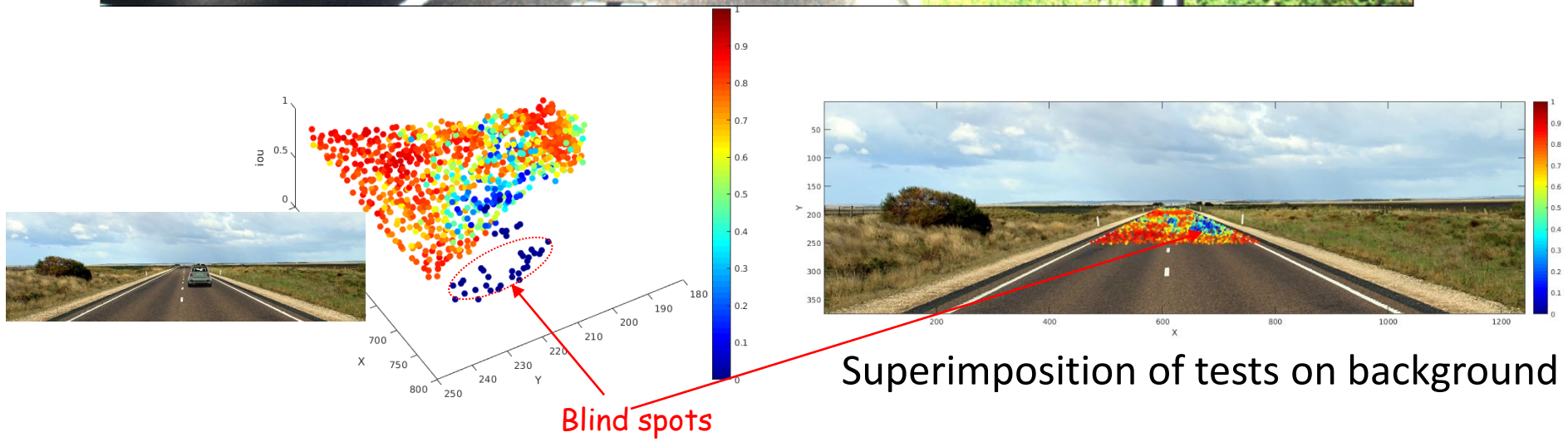
But this one is a real hazard!

[Dreossi, Donze, Seshia, NFM 2017; J. Automated Reasoning 2019]

Extending to Image Streams

[Dreossi, Ghosh, et al., ICML 2017 workshop]





Superimposition of tests on background

Results on squeezeDet NN and KITTI dataset for autonomous driving

Challenge: Environment Modeling

Principles: Data-Driven, Probabilistic, Introspective, Modeling

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*. July 2016. https://arxiv.org/abs/1606.08514.

SCENIC: A Language for Modeling Environment Scenarios

- *Scenic* is a probabilistic programming language defining *distributions over scenes*
- *Use cases:* data generation, test generation, verification, debugging, design exploration, etc.

- Example scenario: a badly-parked car

```
from gta import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * (10, 20) deg
Car left of (spot offset by -0.5 @ 0),
    facing badAngle relative to roadDirection
```

Images created with GTA-V

[D. Fremont et al., "Scenic: A Language for Scenario Specification and Scene Generation", EECS TR March 2018, in PLDI 2019.]

SCENIC: Environment Scenario Modeling Language

Scenic makes it possible to specify broad scenarios with complex structure, then generate many concrete instances from them automatically:

Platoons

Bumper-to-Bumper Traffic

(~20 lines of Scenic code)

Use Case: Retraining with Hard Cases

Improves accuracy on hard cases without compromising accuracy on original training set

e.g. for car detection, one car partially occluding another:

Use Case: Debugging a Known Failure

Use Case: Debugging a Known Failure

Scenic makes it easy to vary a scenario along different dimensions:

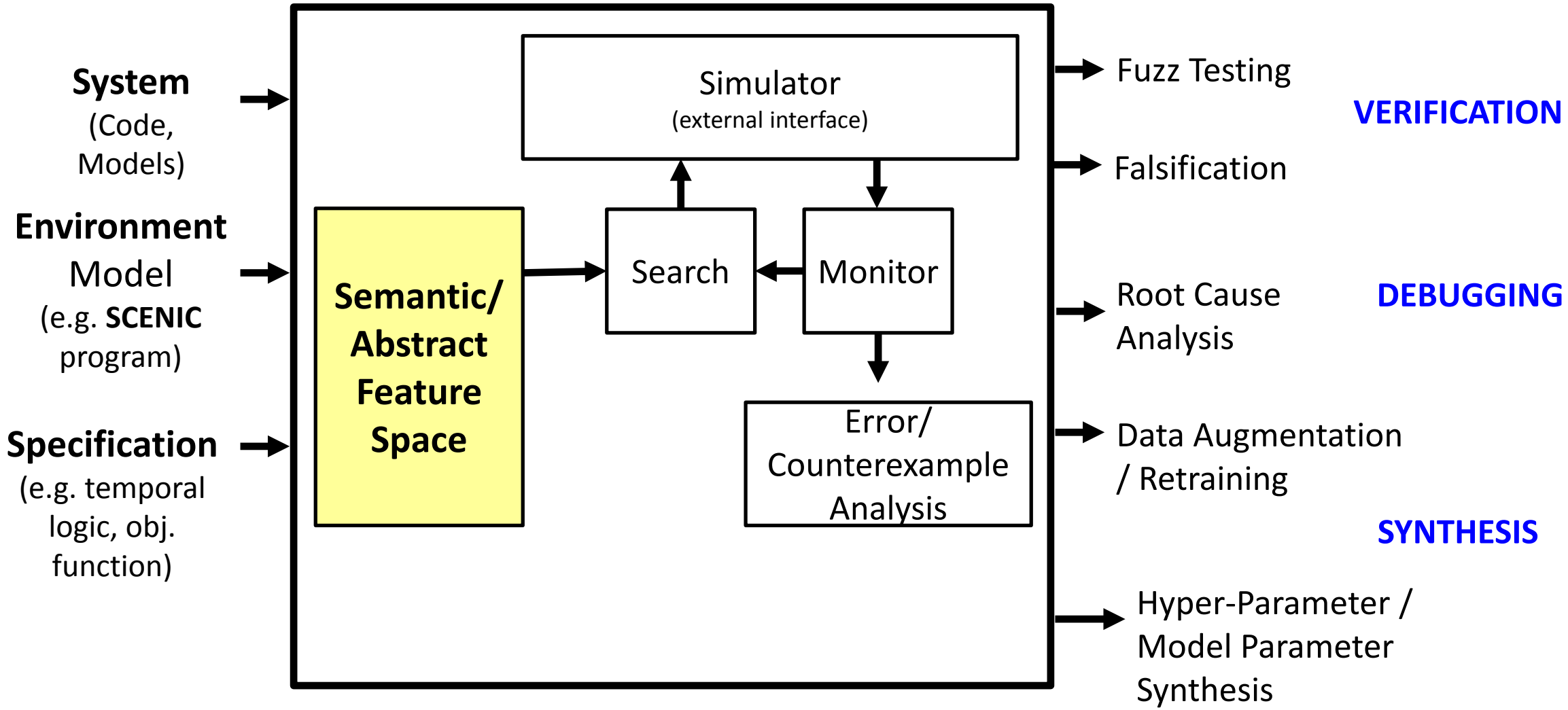
Add noise

Change car model

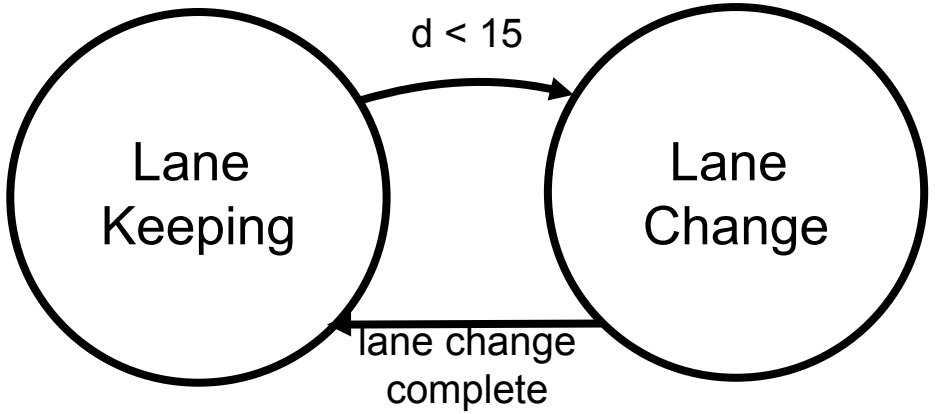
Change global position

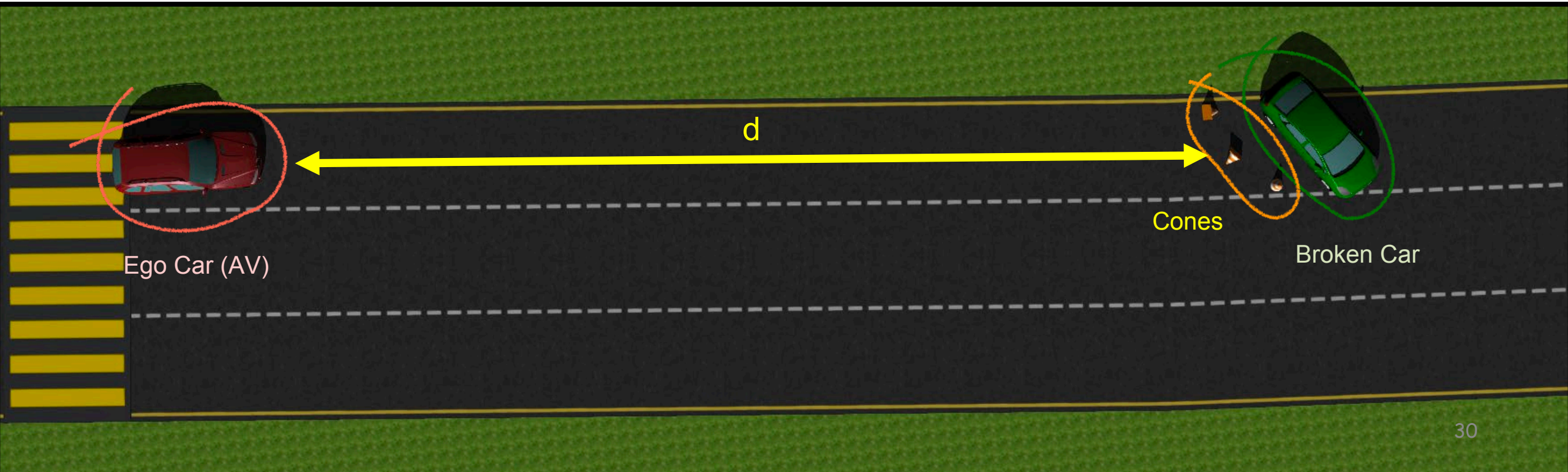
VERIFAI: A Toolkit for the Design and Analysis of AI-Based Systems [CAV 2019]

https://github.com/BerkeleyLearnVerify/VerifAI



Case Study for Temporal Logic Falsification with VerifAI: Navigation around an accident scenario



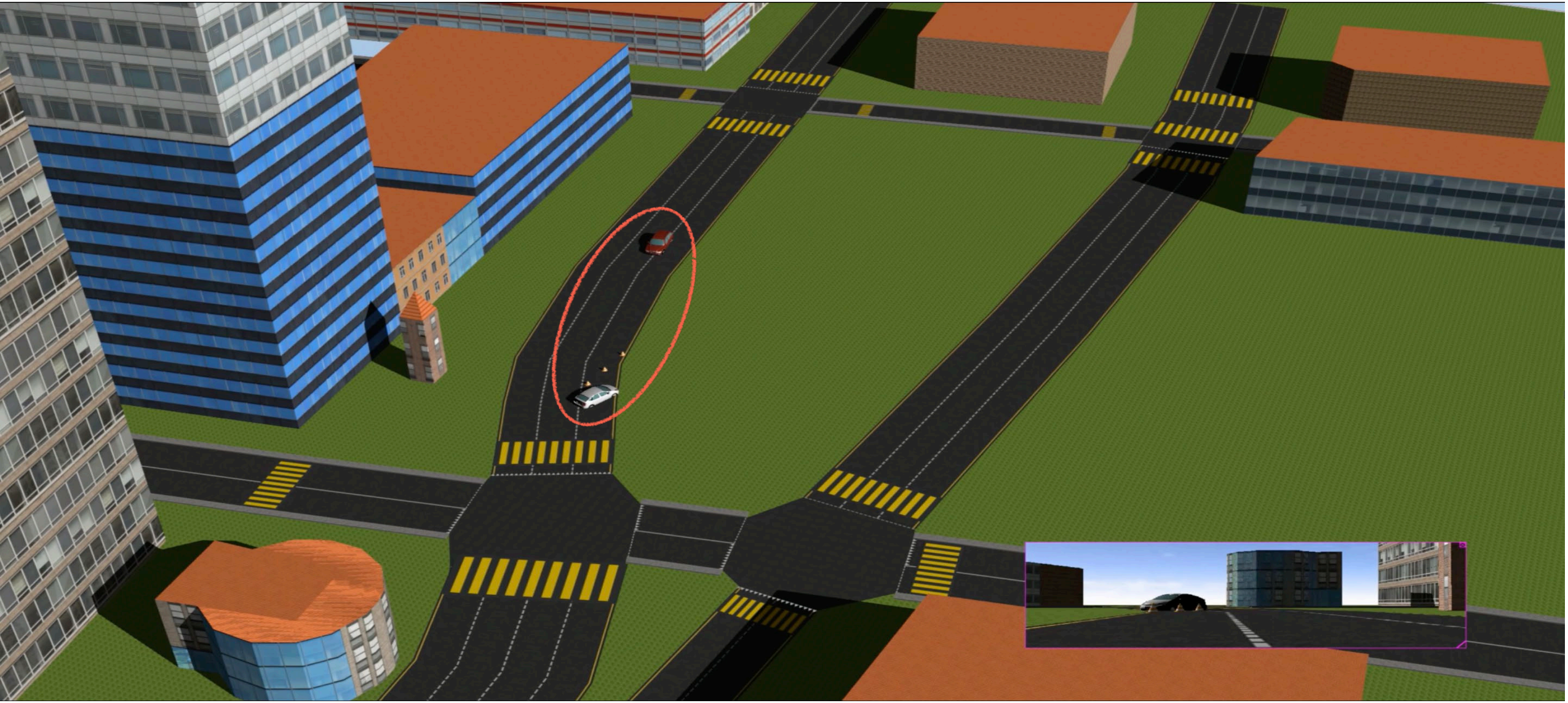


Modeling Case Study in the SCENIC Language

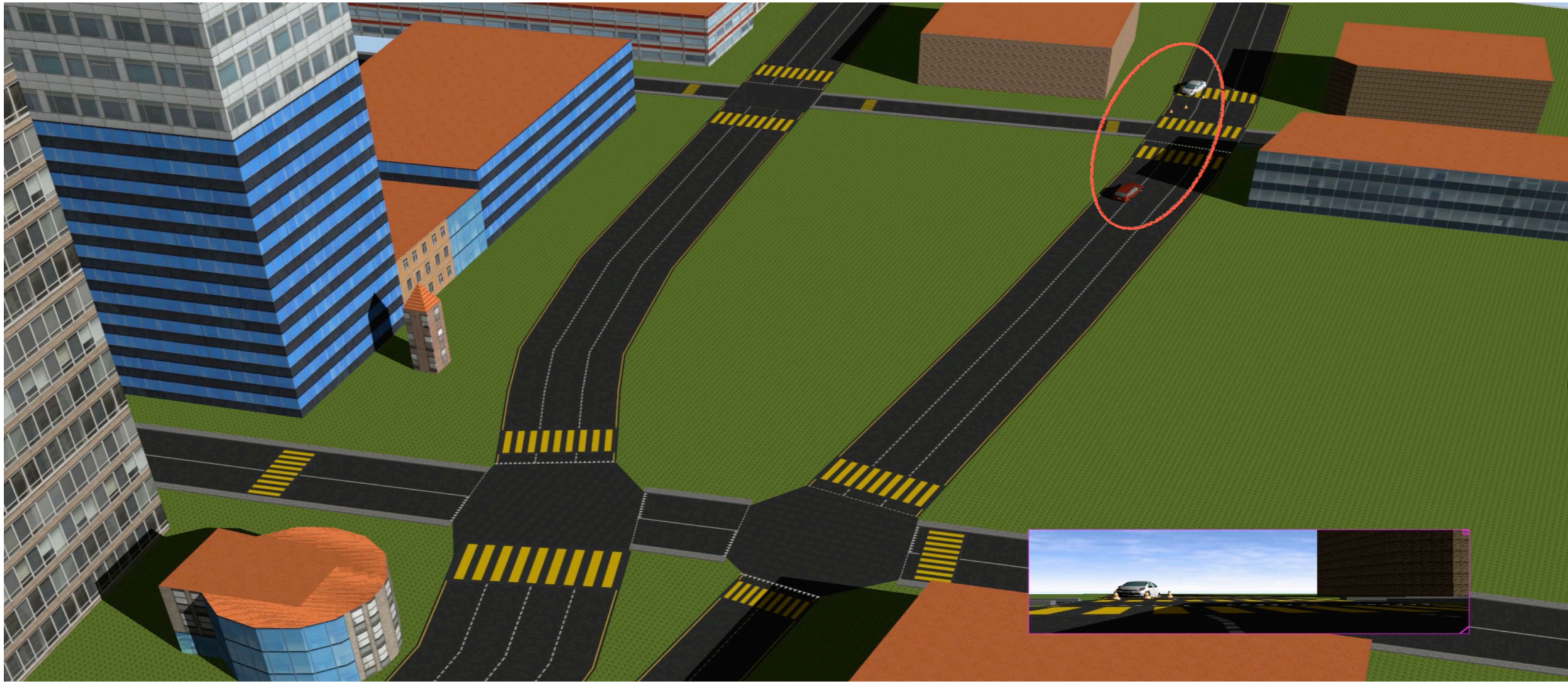
```
# Pick location for blockage randomly along curb
blockageSite = OrientedPoint on curb

# Place traffic cones
spot1 = OrientedPoint left of blockageSite by (0.3, 1)
cone1 = TrafficCone at spot1,
                     facing (0, 360) deg

...

# Place disabled car ahead of cones
SmallCar ahead of spot2 by (-1, 0.5) @ (4, 10),
          facing (0, 360) deg
```

Fremont et al., *Scenic: A Language for Scenario Specification and Scene Generation*, PLDI 2019.

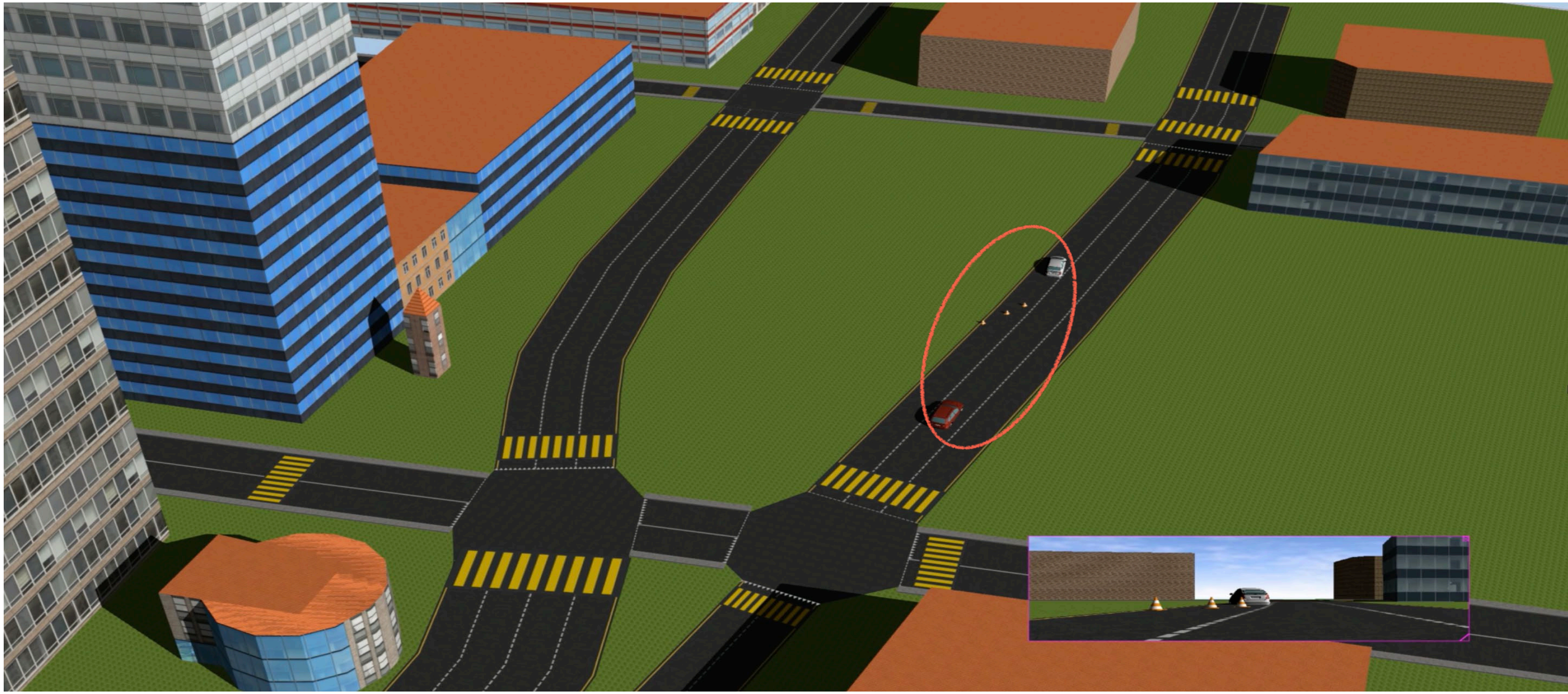
Using Scenic to Generate Initial Scenes



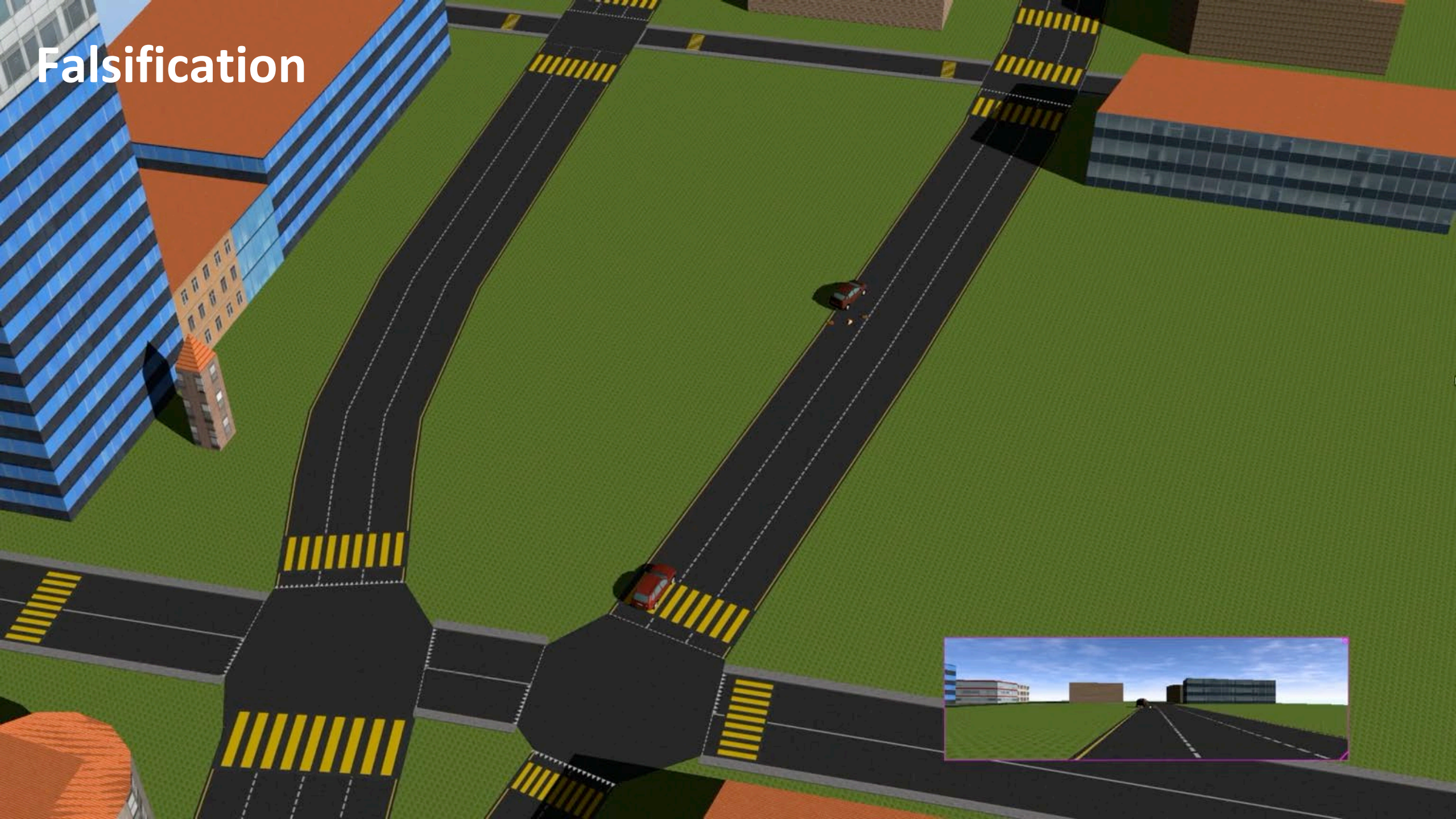
Using Scenic to Generate Initial Scenes



Using Scenic to Generate Initial Scenes



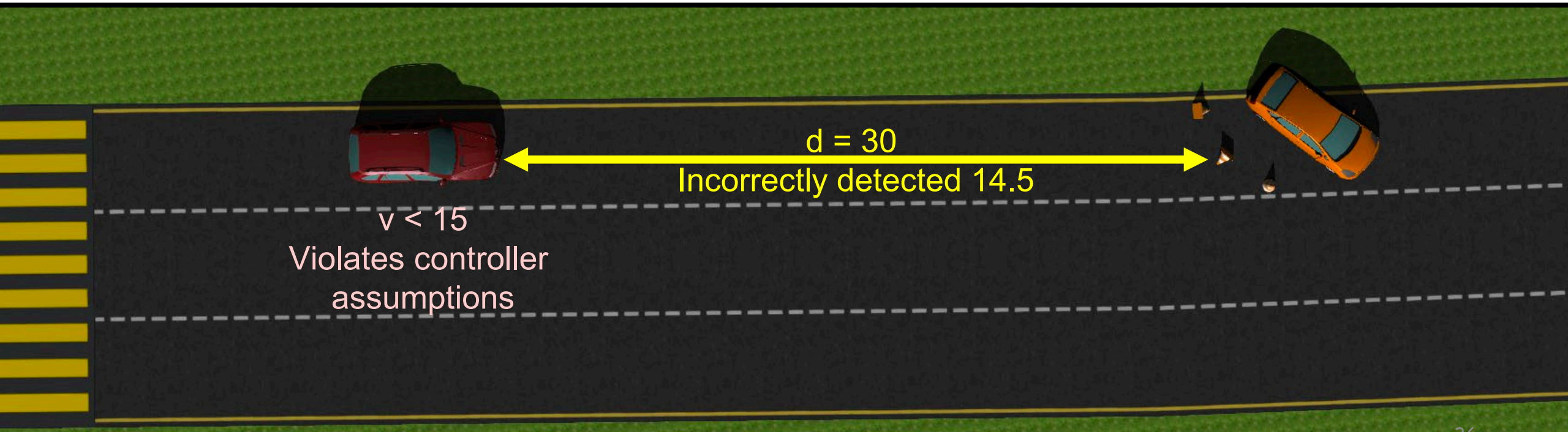
Falsification



Analyzing the failure

Fix the controller:
Update model assumptions
and re-design controller

Retrain the perception module:
Collect the counter-example images and
retrain the network [IJCAI'18]



Challenge: How to (Re)Synthesize Machine Learning Components

Principle: Use Oracle-Guided Inductive Synthesis (OGIS)

T. Dreossi et al. *Counterexample-Guided Data Augmentation*, IJCAI 2018.
D. Fremont et al. *Formal Analysis and Redesign of a Neural Network-Based Aircraft Taxiing System with VerifAI*, CAV 2020.

Standard Machine Learning

LEARNER

labeled/unlabeled data

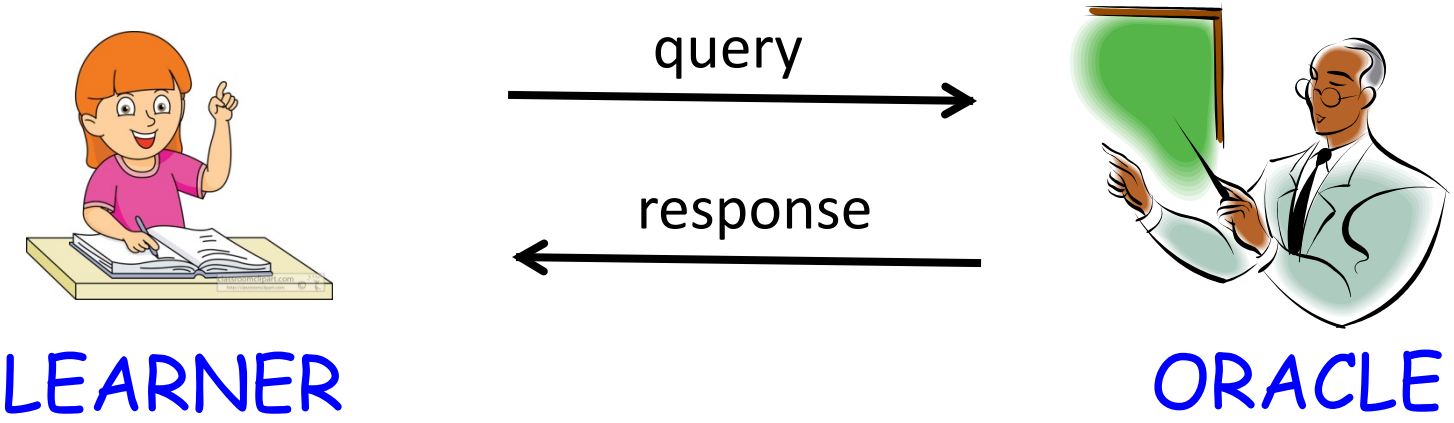
SOURCE OF DATA

Learned Model only as good as the Data!

Correct-by-Construction Design with Oracle-Guided Inductive Synthesis

Key Idea: **Oracle-Guided Learning**

Combine Learner with Oracle (e.g., Verifier) that answers Learner's Queries

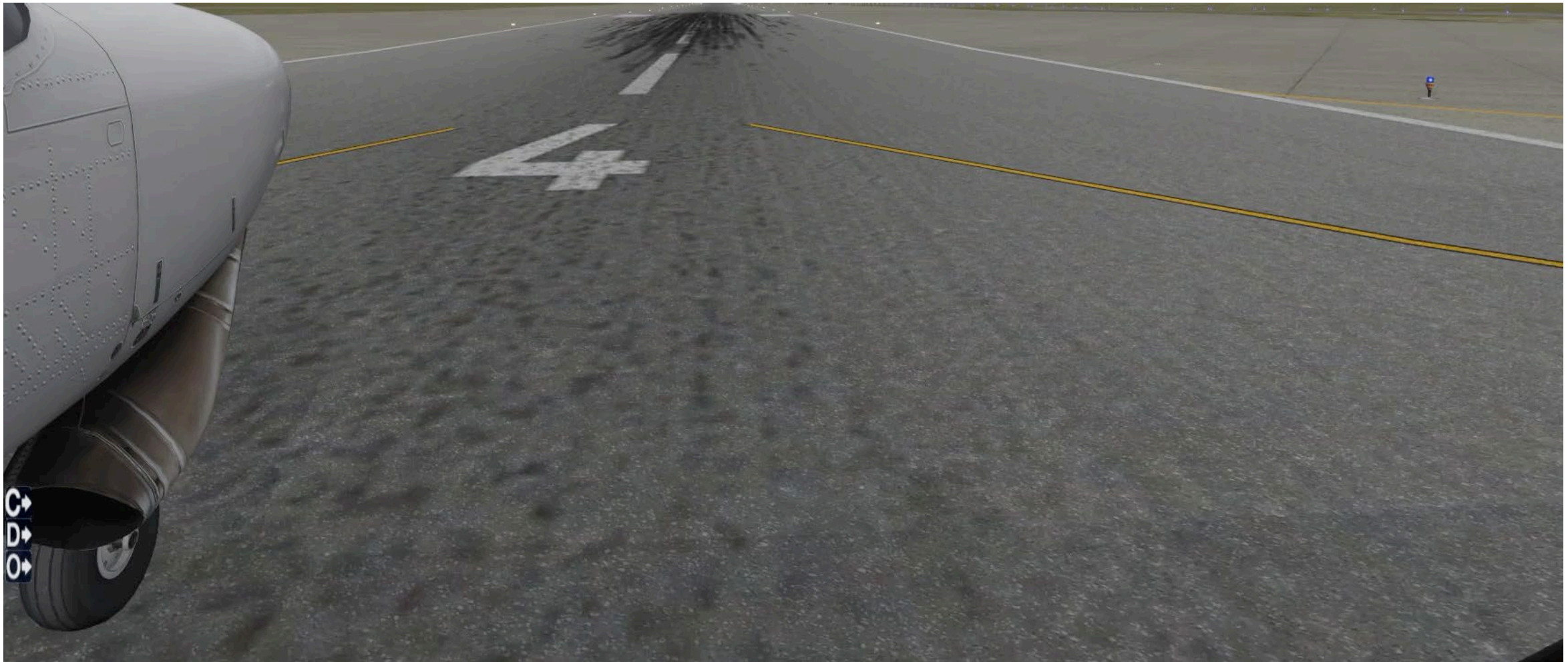


Common Instance: *Counterexample-Guided Inductive Synthesis*

[Jha & Seshia, "A Theory of Formal Synthesis via Inductive Learning", 2015, Acta Informatica 2017.]

DARPA Assured Autonomy Case Study: Automated Taxiing

- TaxiNet: NN-based research prototype from Boeing
- Specification: track centerline within 1.5 m



Scenic Scenario for Falsification of TaxiNet in X-Plane

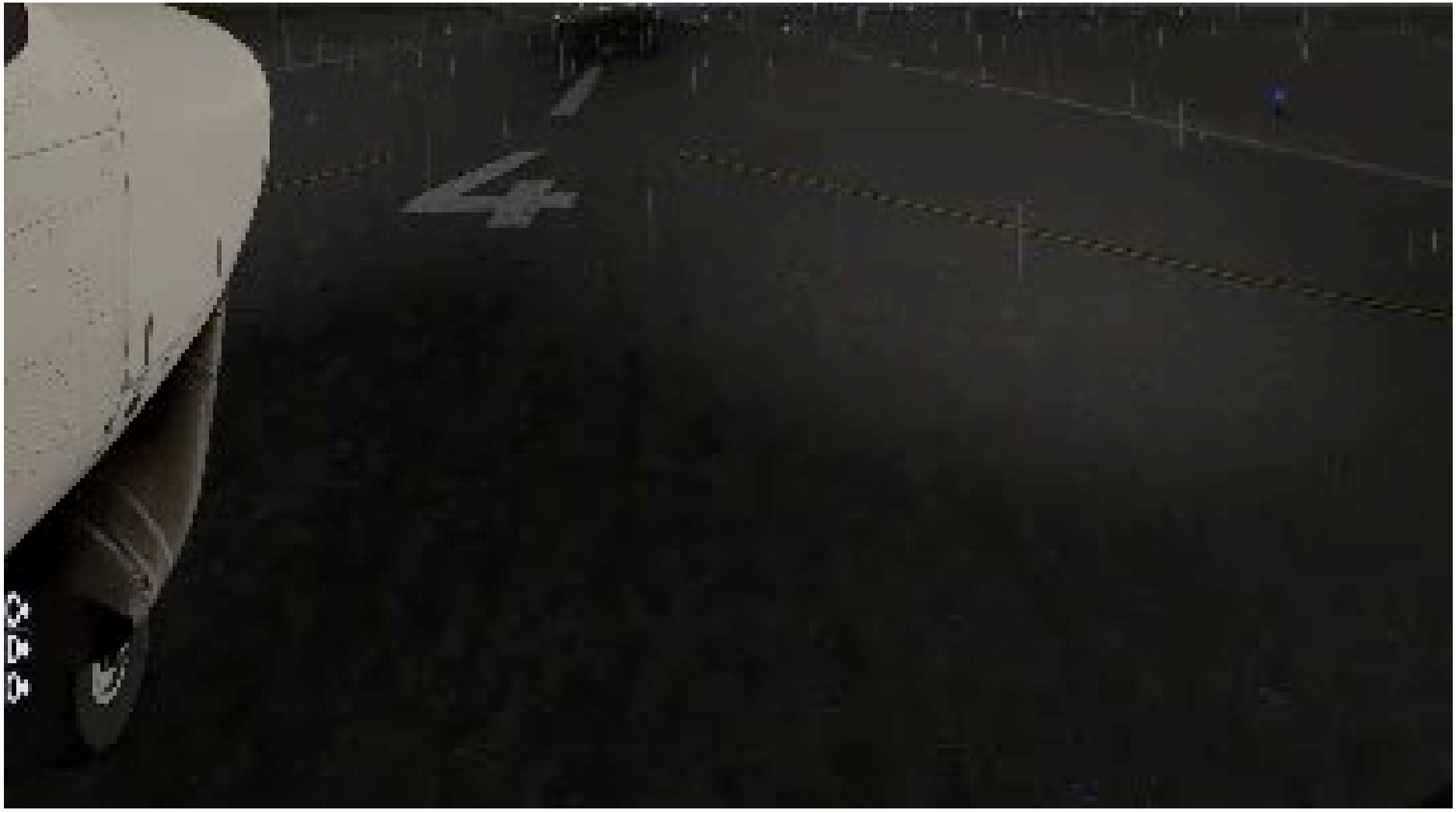
Semantic features:

- Time of day
- Type of clouds
  - none, cirrus, overcast, etc.
- Percentage of rain
- Position & size of tire mark

Property: **G** [CTE <= 1.5 m]

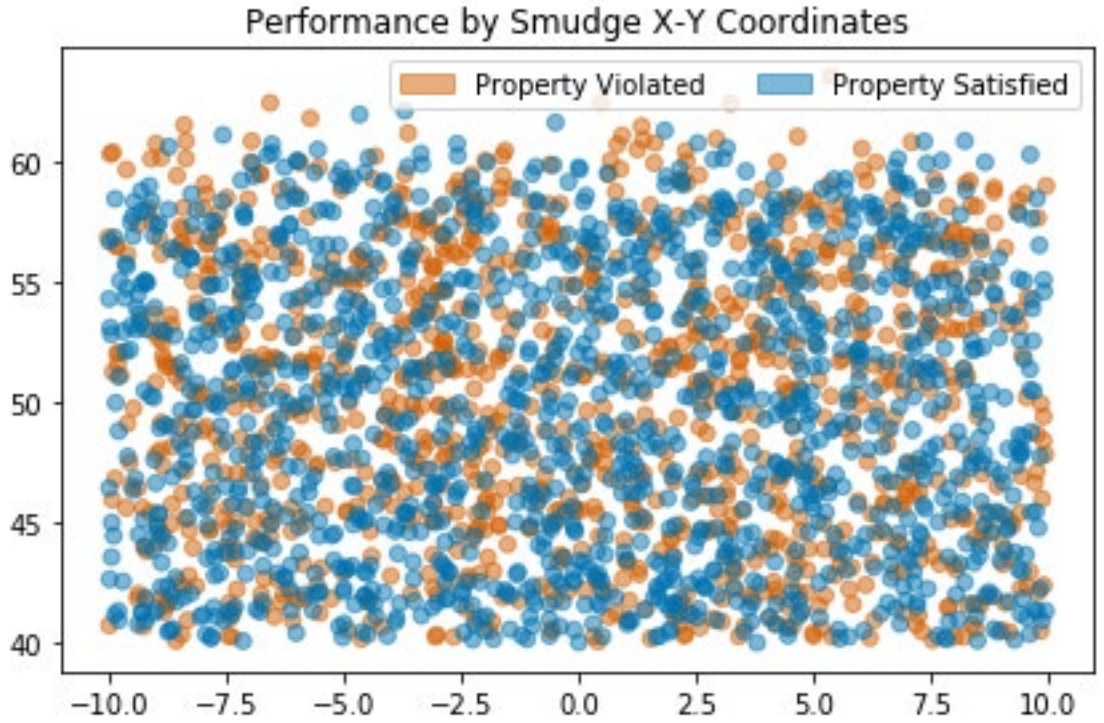
```
# Time of day: from 6 am to 6 pm
param zulu_time = ((6, 18) + 8) * 60 * 60

# Raining 1/3 of the time.
# 2/3: any cloud type 0-5; zero rain
# 1/3: cloud types 3-5; rain from 25% to 100%
clouds_and_rain = Options({
    tuple([Uniform(0, 1, 2, 3, 4, 5), 0]): 2,
    tuple([Uniform(3, 4, 5), (0.25, 1)]): 1
})
param cloud_type = clouds_and_rain[0]
param rain_percent = clouds_and_rain[1]

# Plane
ego = Plane

# Smudge (tire mark) on the runway
Smudge at (-10, 10) @ (40, 80),
    facing Normal(0, 2) deg,
    with height Normal(20, 3),
    with alpha (0.4, 1)
```

Falsifying Run Found with VerifAI/Scenic

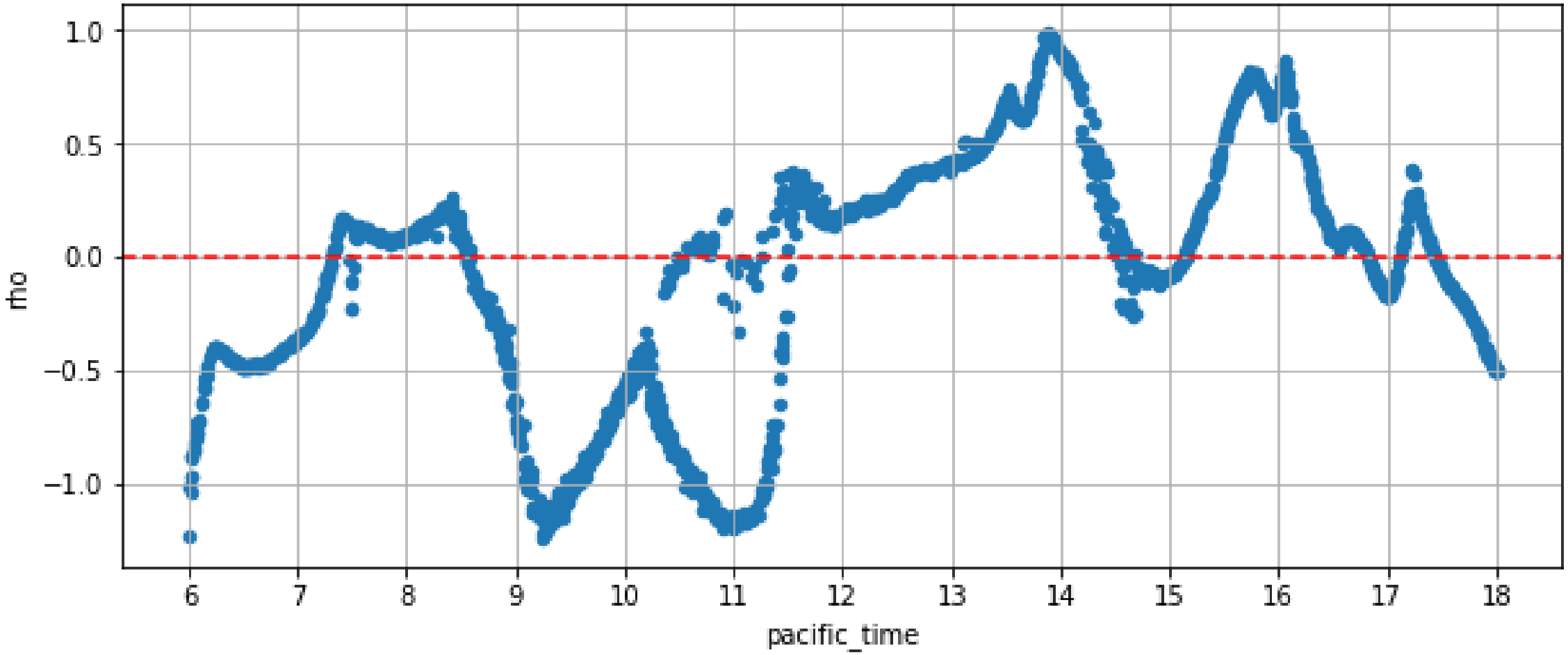


Error Analysis and Debugging

- Initial experiment: 2405 runs
  - 45% violated the CTE property
- Tire mark had no effect (neither did rain)
- Time of day most important
- Clouds also significant



Impact of Time of Day (with no clouds)



Retraining using Scenic

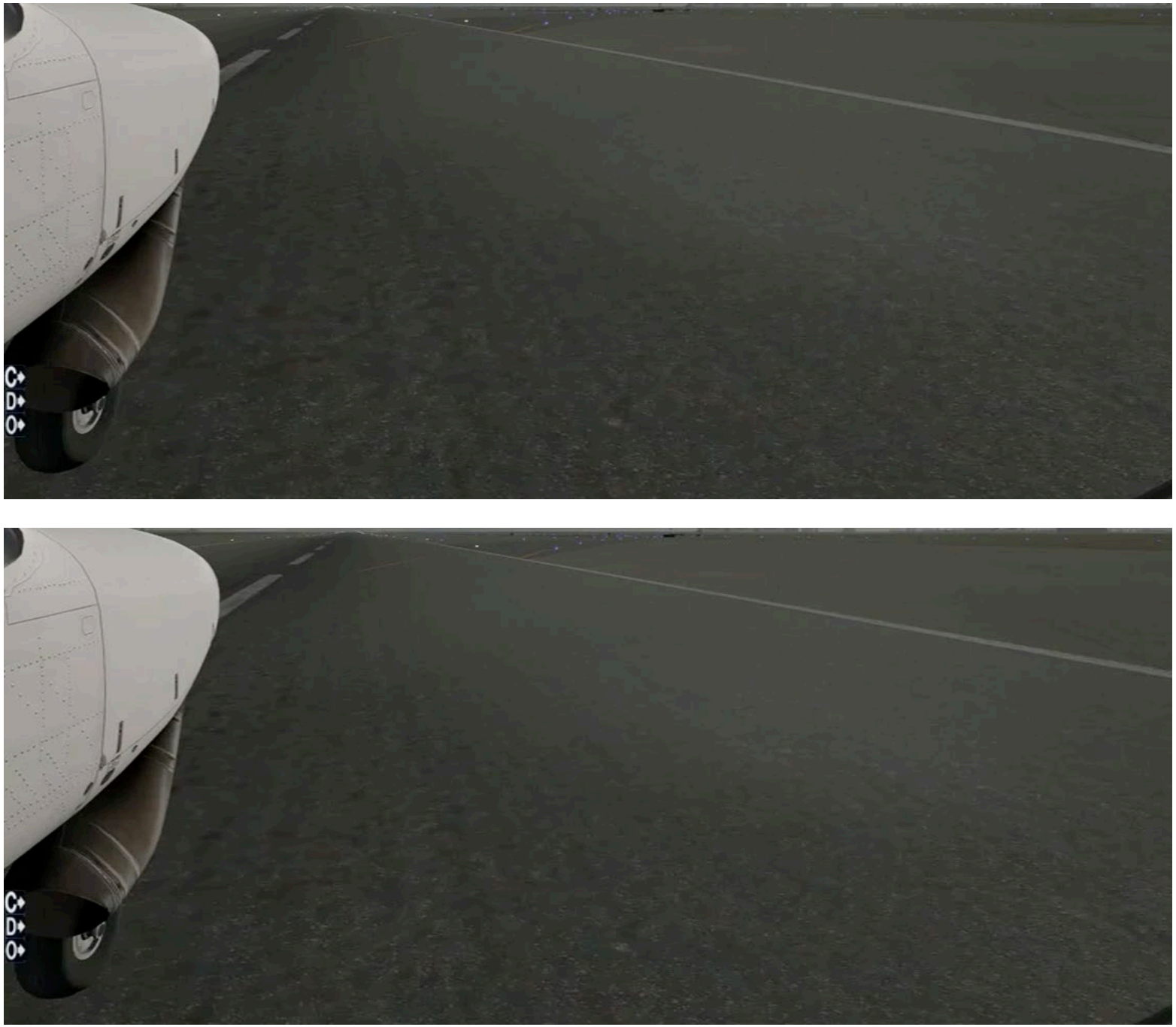
- Used Scenic to generate a diverse training set including counterexamples
- Initial plane position up to 8 meters and 30° off of centerline

```
# Time of day: from 6 am to 6 pm
local_time = (6, 18)
param zulu_time = (local_time + 8) * 60 * 60

# Raining 1/3 of the time.
# Rain requires cloud types 3-5.
rain_frac = (0.25, 1)
clouds_and_rain = Options({
    tuple([Uniform(0, 1, 2, 3, 4, 5), 0]): 2,
    tuple([Uniform(3, 4, 5), rain_frac]): 1
})
param cloud_type = clouds_and_rain[0]
param rain_percent = clouds_and_rain[1]

# Plane
ego = Plane at (-8, 8) @ (0, 2000),
            facing (-30, 30) deg
```

Results of Retraining

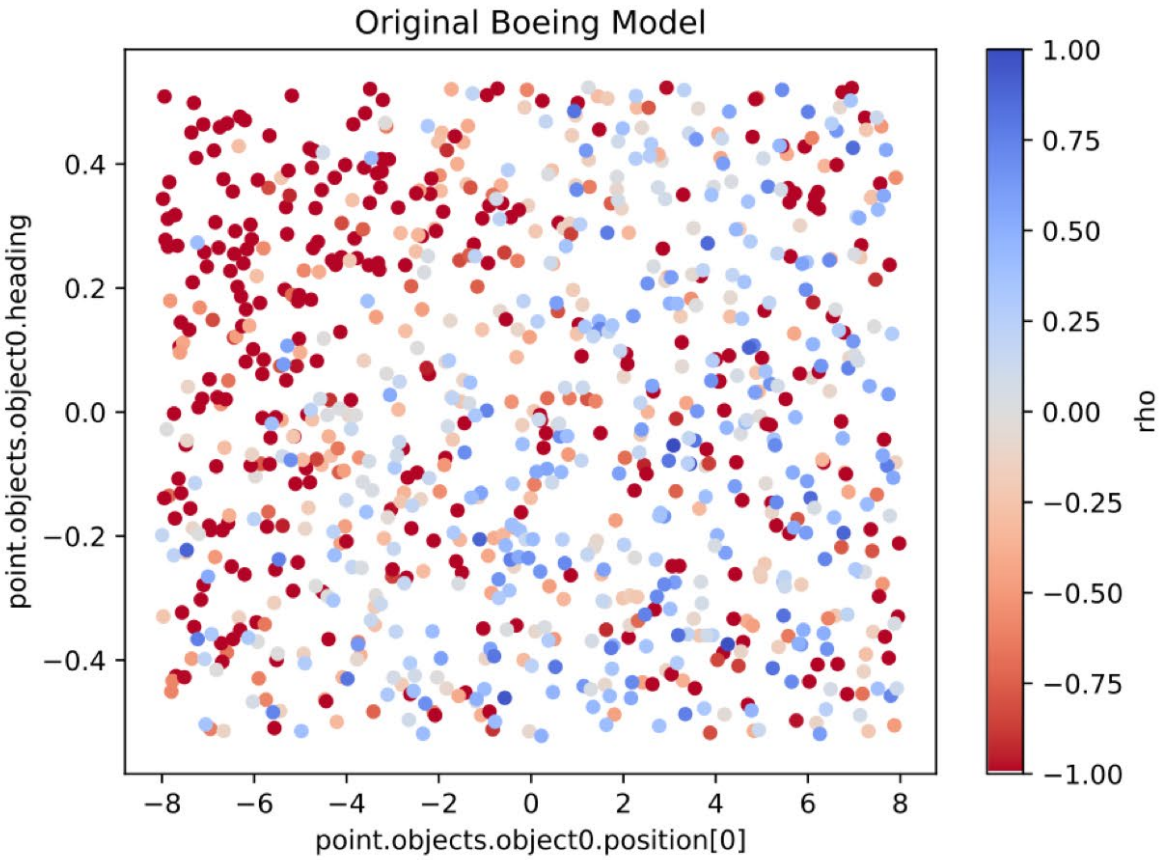
- Above: original
- Below: retrained

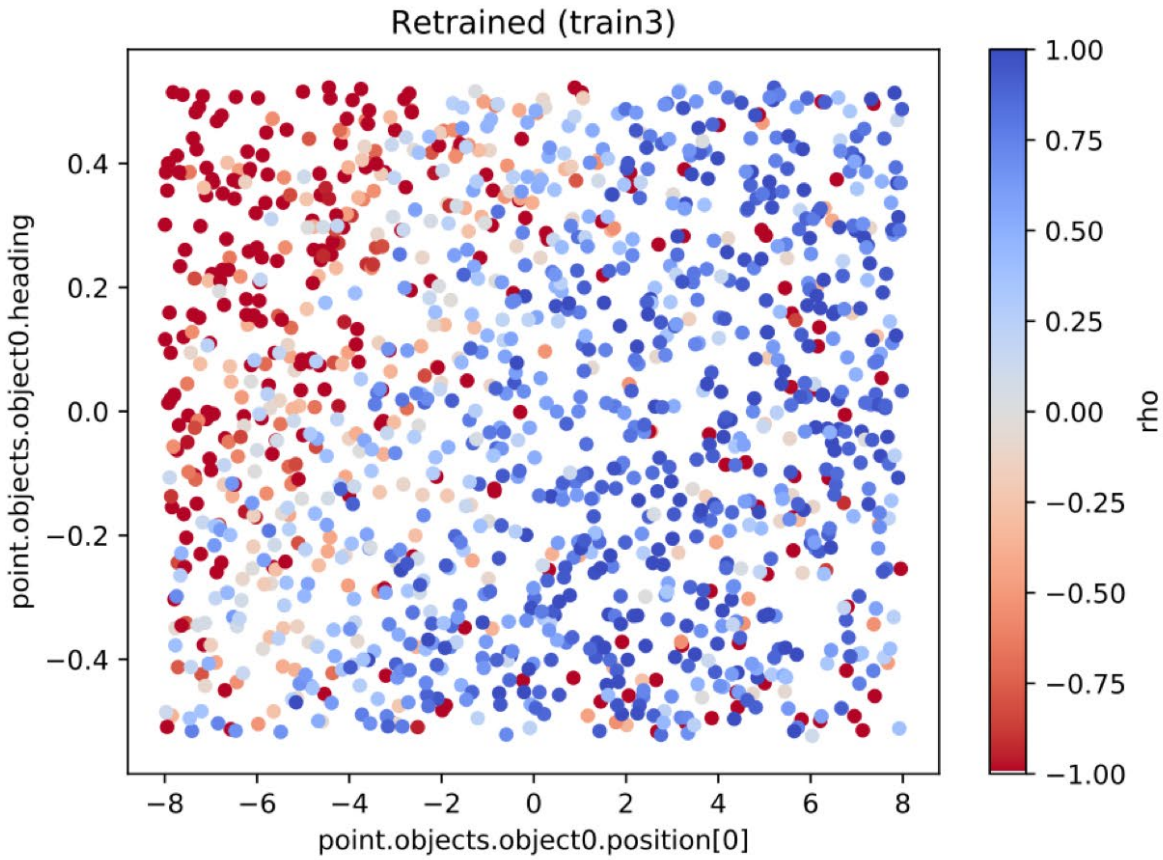
(Both simulations use exactly the same initial condition and env parameters)



Results of Retraining

- Significantly improved performance, but bugs remain
  - Original model satisfies property 37% of time; retrained 62% of time

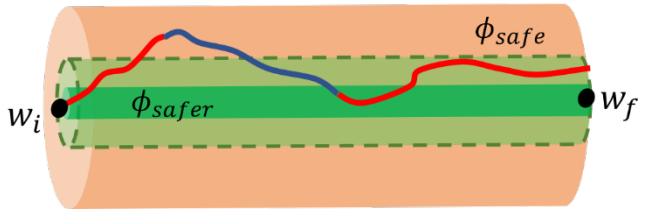




Ongoing/Future Directions

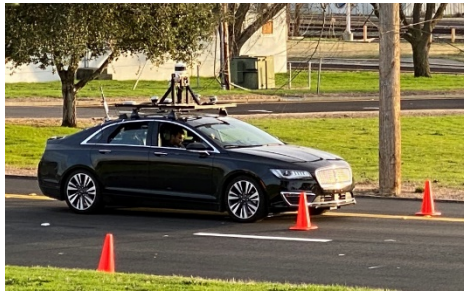
Open-Source Verified AI Toolkit
(VerifAI & Scenic, on github)

Run-Time Assurance
SOTER framework based on Simplex architecture [DSN 2019]



Verified Human-Robot Collaboration
Learning Specifications from Demonstrations, Interaction-Aware Control, etc. [IROS 2016, NeurIPS 2018, CAV 2020]

Explaining Success/Failures of Deep Learning
Automated approach using Scenic [CVPR 2020]



Bridging Simulation & Real World
Metrics to compare simulated vs real behaviors [HSCC 2019]
Using falsification to design real world tests [ITSC 2020]

Safety in Simulation → Safety on the Road? [Fremont et al., ITSC 2020]

Unsafe in simulation → unsafe on the road: **62.5% (incl. collision)**
Safe in simulation → safe on the road: **93.5% (no collisions)**

[joint work with American Automobile Association and LG Electronics]

Conclusion: Towards Verified AI/ML based Autonomy

Challenges		Core Principles
1. Environment (incl. Human) Modeling	→	Data-Driven, Introspective, Probabilistic Modeling
2. Specification	→	Start with System-Level Specification, then Component Spec (robustness, …)
3. Learning Systems Complexity	→	Abstraction, Semantic Representation, and Explanations
4. Efficient Training, Testing, Verification	→	Compositional Analysis and Semantics-directed Search/Training
5. Design for Correctness	→	Oracle-Guided Inductive Synthesis; Run-Time Assurance

Exciting Times Ahead!!! Thank you!

S. A. Seshia, D. Sadigh, S. S. Sastry. *Towards Verified Artificial Intelligence*. July 2016. https://arxiv.org/abs/1606.08514.