Scenic and VerifAI: Tools for Assured AI-Based Autonomy

Part II: Tutorial

Daniel J. Fremont

Edward Kim

Sanjit A. Seshia

UC Santa Cruz

UC Berkeley

Webinar
August 20, 2020

Scenic

High-Level, Probabilistic Programming Language for Modeling Environment Scenarios

VerifAI

Requirements Specification + Algorithms for Design, Verification, Testing, Debugging

GitHub **Open-Source Tools**

https://github.com/BerkeleyLearnVerify/Scenic
https://github.com/BerkeleyLearnVerify/VerifAI

for

Industry

Improve assurance of the systems you build

Academia

Use these tools in your research

Government/ Regulators

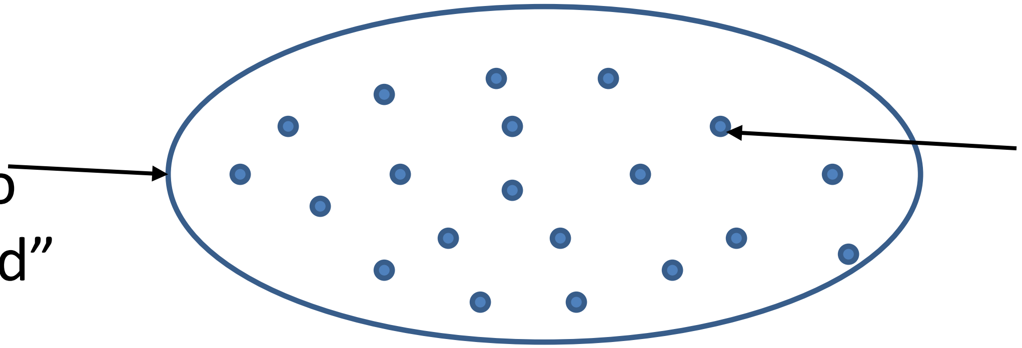
Evaluate the safety of AI-based autonomous systems

Share Scenarios and Metrics **Community** Develop Corpus of Tools and Data

Basic Terminology

- Scene: A configuration of objects and agents in physical space, with associated attributes and behaviors
- Concrete Spatial Scenario = Scene
- Concrete (Spatio-Temporal) Scenario: A sequence of scenes over time
- Abstract Scenario = Set of Concrete Scenarios
- Abstract Scenarios can be Probabilistic, i.e., the set can have an associated distribution

Abstract Scenario
“a car on the road”



Concrete Scenario
“a Tesla Model 3 at 1.2m x 4m”

SCENIC: A Language for Scenario Specification and Data Generation

- Scenic Program defines a probabilistic abstract scenario
  -- *distribution over scenes/concrete scenarios*
- First created in 2017-18
- Readable, concise syntax for common geometric and behavioral relationships
- Embedded DSL in Python
- Generative back-end implementing domain-specific sampling techniques
- Blends imperative and declarative programming

Bumper-to-Bumper Traffic
(~20 lines of Scenic)

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

Scenic enables modeling Three Types of Constraints

- **Use Case: Synthetic Data Generation**
  - "create traffic images to train this neural network"
- **Use Case: Synthesizing Test Stimuli**
  - "generate edge cases in rush hour traffic scenario"

- Objects should not intersect — **Hard Constraint**
- Usually, be similar to real-world traffic — **Soft Constraint**
- Generate a diverse image/test set from distribution — **Randomness Constraint**

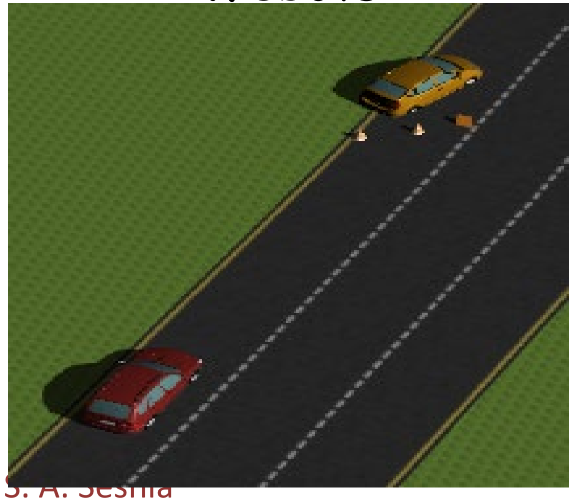
Scenic and VerifAI are Simulator-Agnostic

CARLA

LGSVL

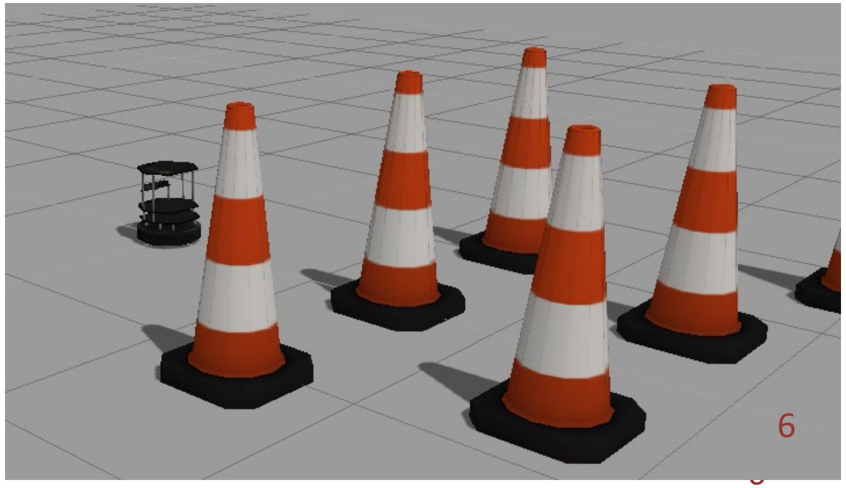
Grand Theft Auto V

Webots



X-Plane Flight Simulator

Gazebo Robotics Simulator



Outline for this Webinar

Part I: Overview

- Challenges for Assurance of Autonomous Driving Systems
- Overview of VerifAI and Scenic
- Case Study on Formal Scenario-Based Testing in Simulation and on the Road

Part II: Tutorial

- Spatial modeling, data generation, and debugging ML-based perception with Scenic
- Spatio-temporal scenario modeling, testing, falsification, debugging, retraining with Scenic and VerifAI
- Outlook

Tutorial on *Static* Environment Description

Overview of the Tutorial

- Modeling complex ***spatial*** relations among objects and agents
  - Starter Example : Badly Parked Car Scenario
  - Scaled Example : Bumper-to-Bumper Traffic Scenario
- Applications of Scenic
- Modelling ***temporal*** relations will be covered in the next part by Daniel Fremont

Example: a Badly-Parked Car Scenario

Example: a Badly-Parked Car

```
from carla_models import Car, curb, roadDirection
```

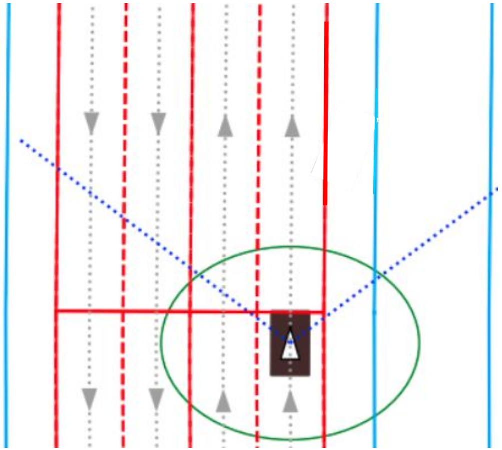
Example: a Badly-Parked Car

```
from carla_models import Car, curb, roadDirection

ego = Car
```

Definition of Car

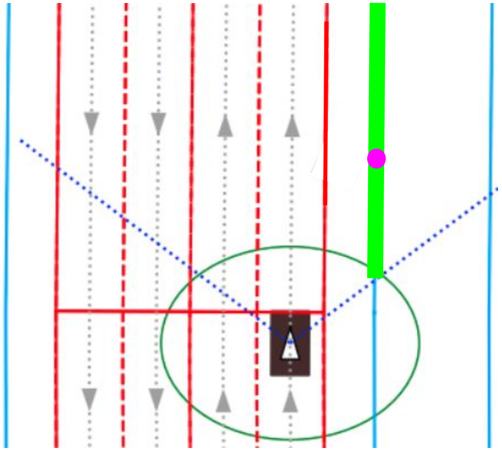
```
class Car():
    position: Point on road
    heading: roadDirection at self.position
    model: CarModel.defaultModel()
    color: Color.defaultCarColor()
    width: self.model.width
    height: self.model.height
    viewAngle: 120 deg
    visibleDistance: 30 # meters
```

Example: a Badly-Parked Car



```
from carla_models import Car, curb, roadDirection

ego = Car
```

Example: a Badly-Parked Car



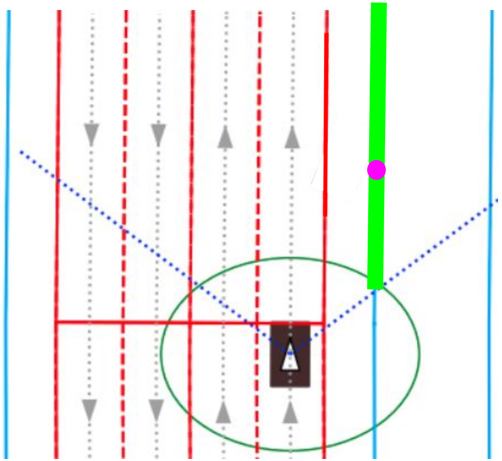
```
from carla_models import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
```

class

specifier

function

Example: a Badly-Parked Car

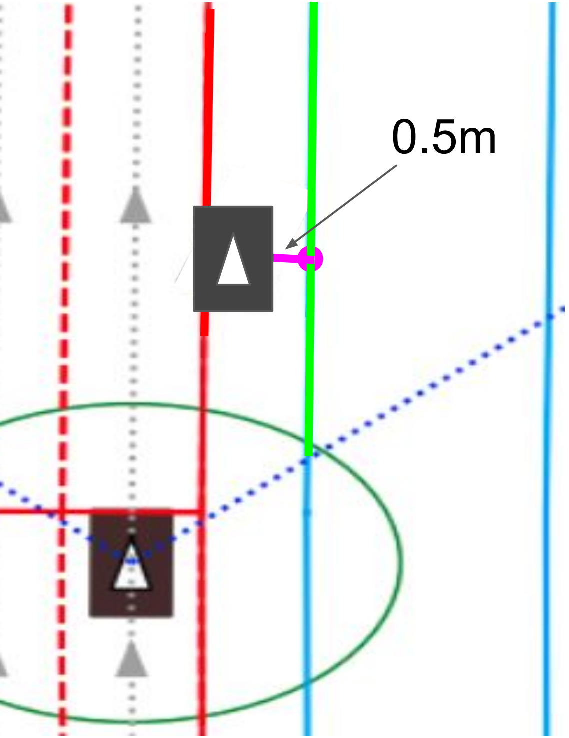


```
from carla_models import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(-1, 1) * Range(10, 20) deg
```

uniform distribution over these discrete choices

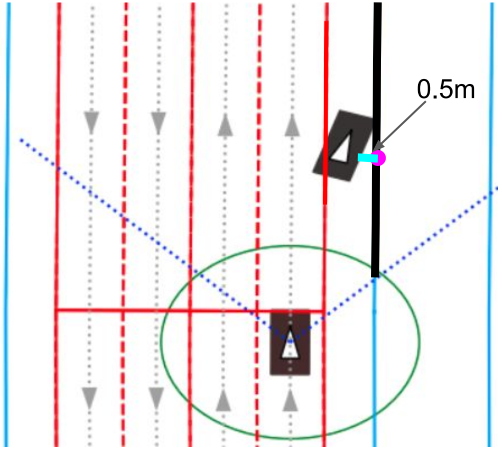
uniform distribution over this interval

Example: a Badly-Parked Car




```
from carla_models import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(-1, 1) * Range(10, 20) deg
parkedCar = Car left of spot by 0.5
```

Example: a Badly-Parked Car




```
from carla_models import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(-1, 1) * Range(10, 20) deg
parkedCar = Car left of spot by 0.5,
    facing badAngle relative to roadDirection
```

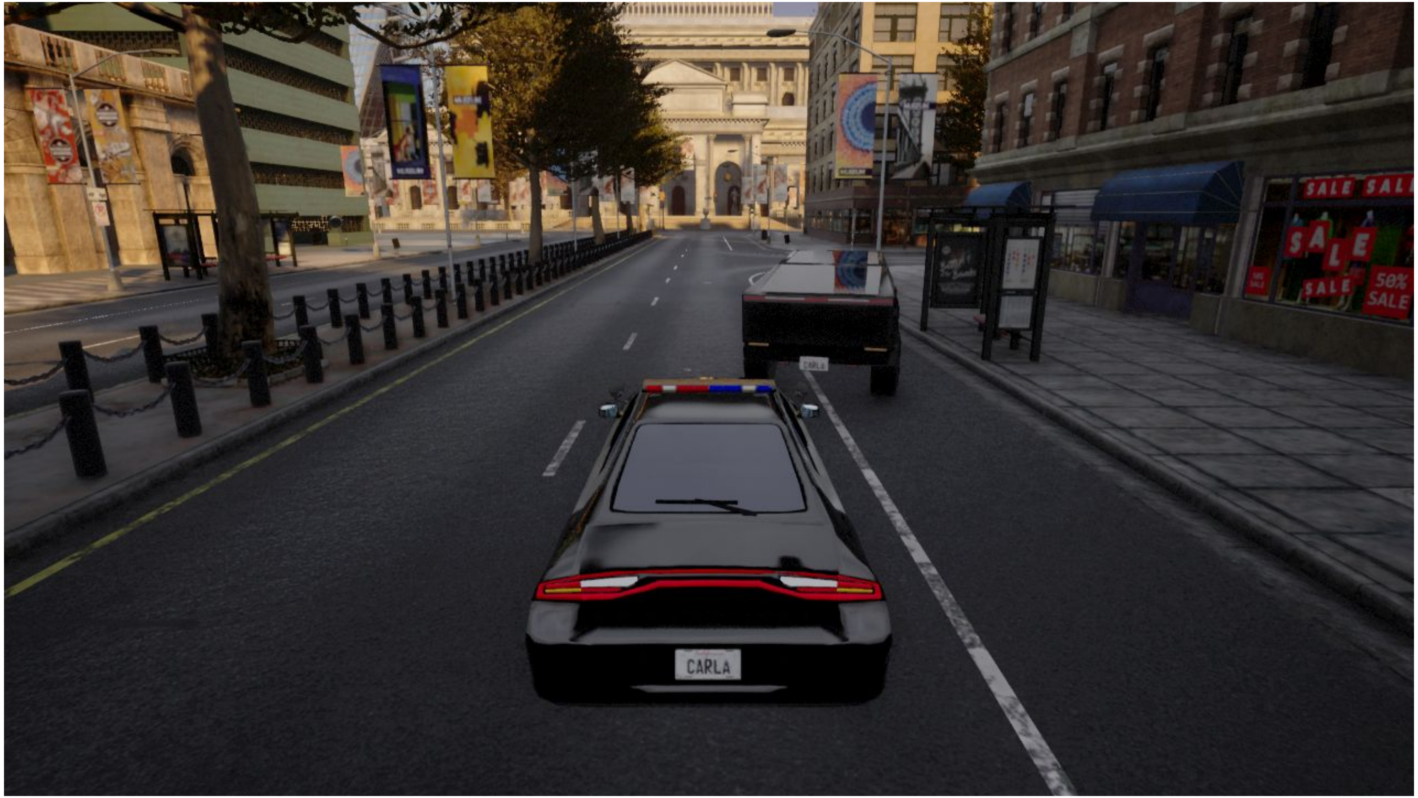
Example: a Badly-Parked Car

Question: How can we generate the badly parked car closer to ego?

Enforcing explicit constraints

```
from carla_models import Car, curb, roadDirection

ego = Car

spot = OrientedPoint on visible curb
badAngle = Uniform(-1, 1) * Range(10, 20) deg
parkedCar = Car left of spot by 0.5,
    facing badAngle relative to roadDirection

require (distance to parkedCar) < 20
```

Example: a (closer) Badly-Parked Car



Scaling up with multiple agents in Scenic

Bumper-to-bumper Traffic Scenes (from Grand Theft Auto V)

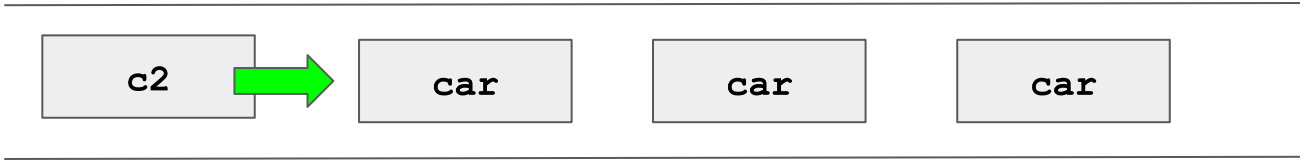
Written with less than <u>20 lines of code!</u>

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

Example : Bumper-to-bumper Traffic

```
def createPlatoonAt(car, numCars, dist=Range(2, 8), shift=Range(-0.5, 0.5), wiggle=Range(-5,5) deg):
    lastCar = car
    for i in range(numCars-1):
        center = follow roadDirection from (front of lastCar) for resample(dist)
        pos = OrientedPoint right of center by shift,
                  facing resample(wiggle) relative to roadDirection
        lastCar = Car ahead of pos
```

```
ego = Car
c2 = Car
platoon = createPlatoonAt(c2, 5, dist=Range(2, 8))
```

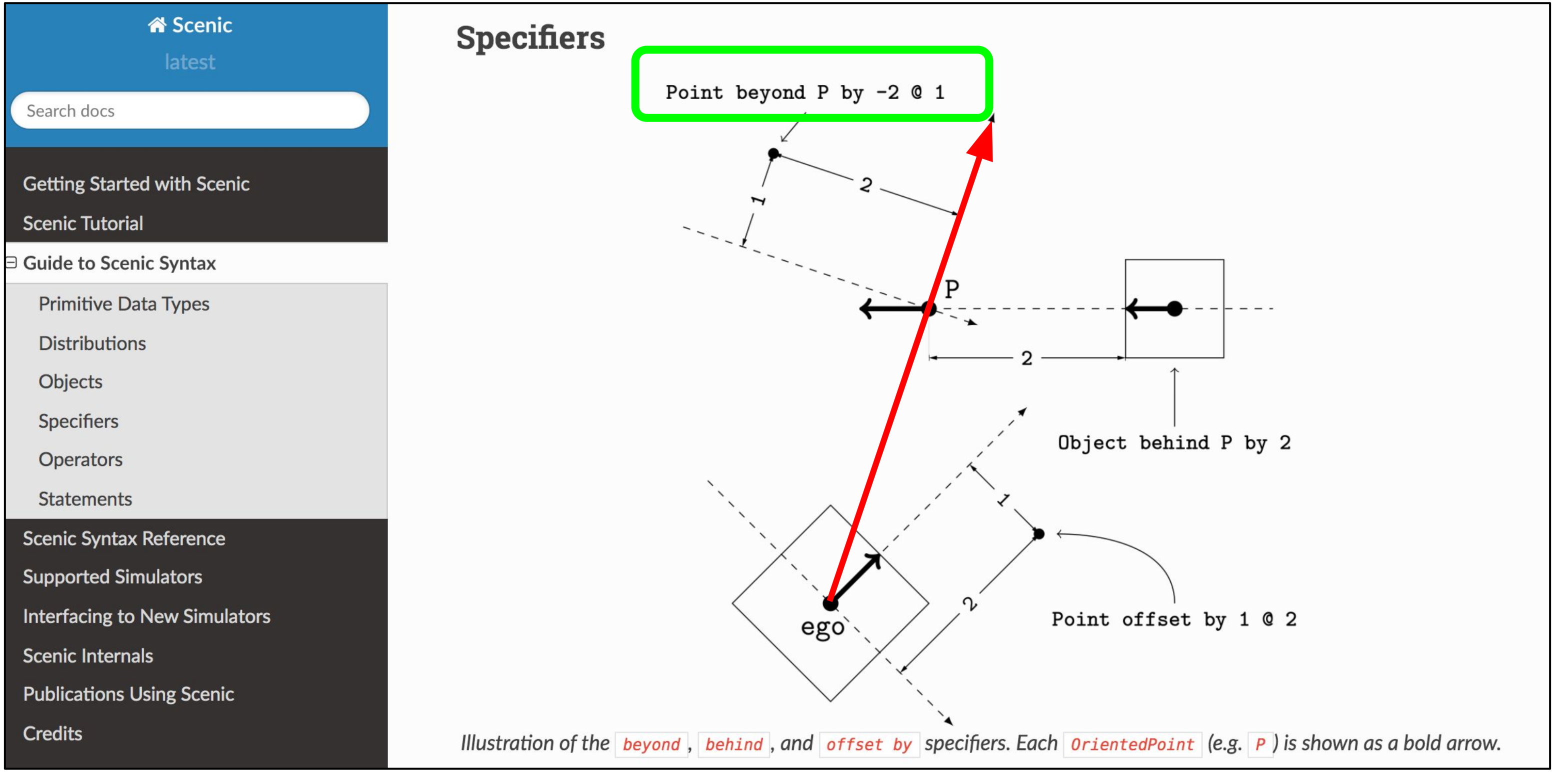


Positioning Objects in Front

Car ahead of ego by 20

Car following roadDirection from ego by 20

Position Specifiers



Source: https://scenic-lang.readthedocs.io/en/latest/syntax_guide.html

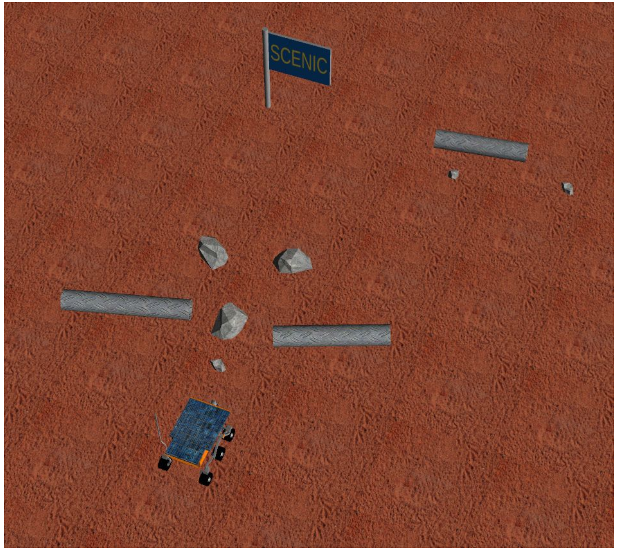
Domain-Specific Sampling Techniques

- Prune infeasible parts of the space given require statements

```
require distance to taxi <= 5

require 15 deg <= (relative
  heading of taxi) <= 45 deg
```

Applications of Scenic

- Exploring system performance
  - Generating specialized test sets
- Debugging a known failure
  - Identify the root cause by exploring semantic space near the scene
- Designing more effective training sets
  - Training on hard cases



Application: Training on Hard Cases

- For car detection, a hard case is one car partially occluding another:

Application: Training on Hard Cases

- Train on untargeted GTA data (“matrix”) [1], test on our overlapping cars scenario; then retrain on mixtures of the two [2]

	Precision on All Testset	Precision on Testset with Occlusion scenes
Trainset (5k images)	72.9 %	62.8%
95% Trainset, 5% Occlusion	73.1 %	68.9%

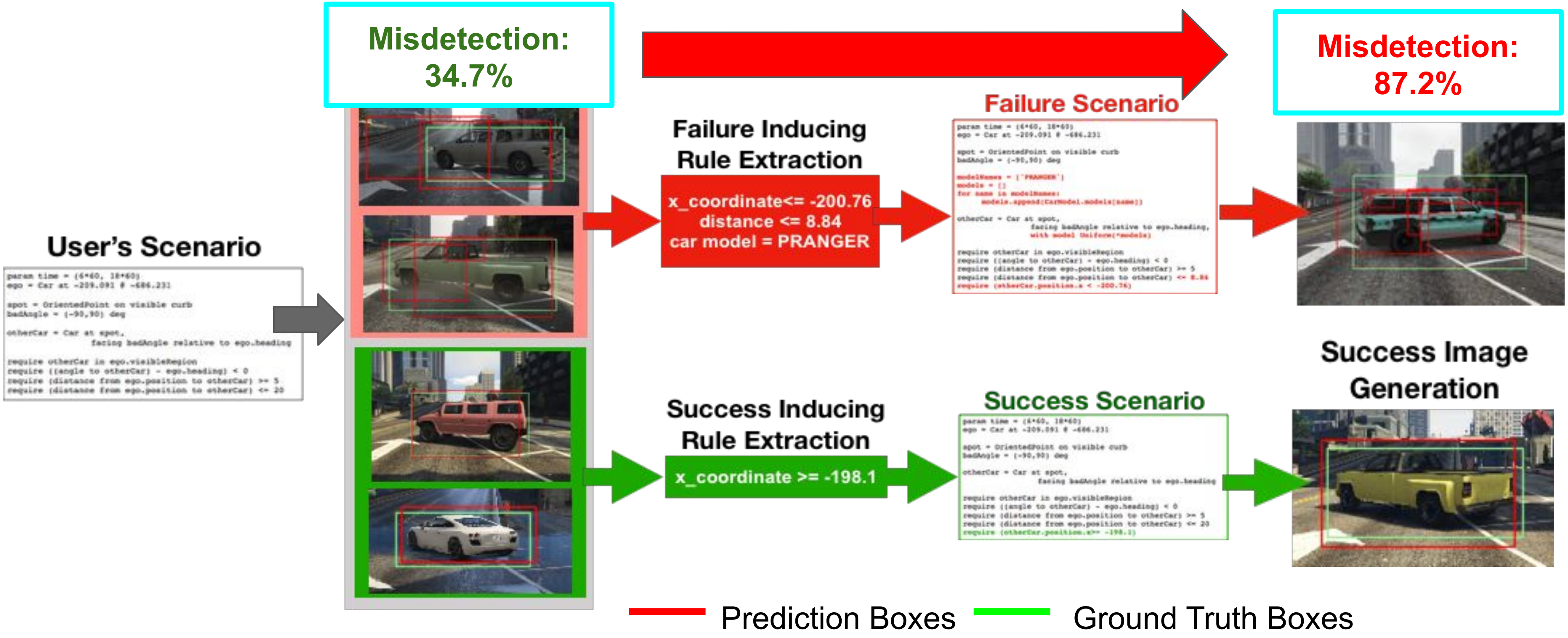
- **<u>Performance in the hard case improves, without hurting the typical case</u>**

[1] Johnson-Roberson et al., *Driving in the Matrix*, ICRA 2017
[2] Daniel Fremont, et al. “Scenic: A Language for Scenario Specification and Scene Generation,” PLDI 2019

Application: Why did the neural network misdetect?

Citation: Edward Kim, et al. “A programmatic and semantic approach to explaining and debugging neural network-based object detectors,” CVPR 2020 [Oral Presentation]

Application: Why did the neural network misdetect?



Citation: Edward Kim, et al. "A programmatic and semantic approach to explaining and debugging neural network-based object detectors," CVPR 2020 [Oral Presentation]

Q&A Session

Dynamic Scenarios in Scenic

Outline

Scenic can describe *dynamic scenarios* which evolve over time.

- Specifying initial conditions and parameters for simulations
  - Falsification of cyber-physical systems (collision-avoidance case study)
  - Analysis & retraining for ML-based systems (runway tracking case study)

- Specifying behaviors of *dynamic agents* which can react to their environment
- *Composing* scenarios in space and time
  - Falsification in dynamic environments (pedestrian scenario case study)

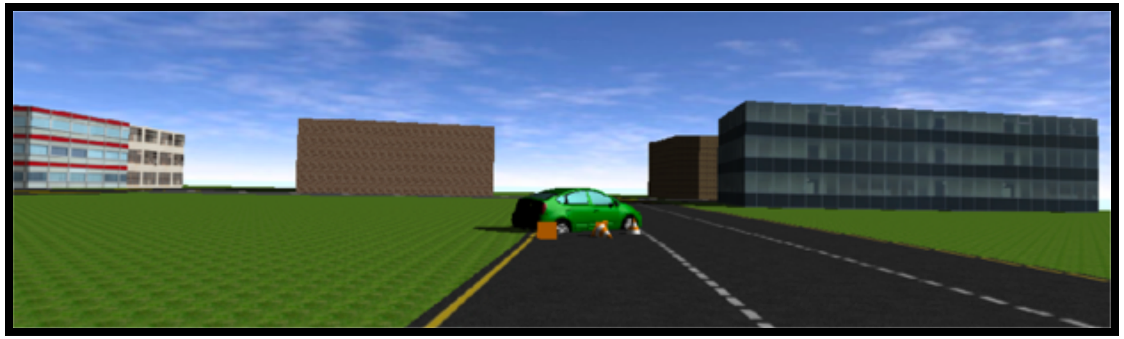
Outline

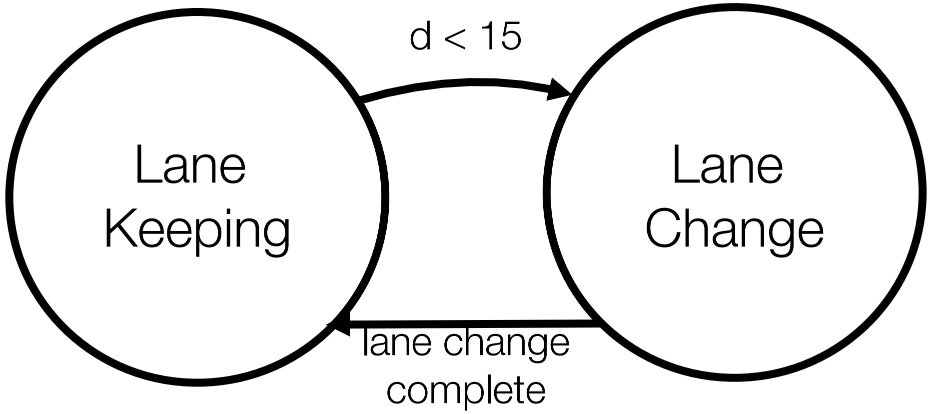
Scenic can describe *dynamic scenarios* which evolve over time.

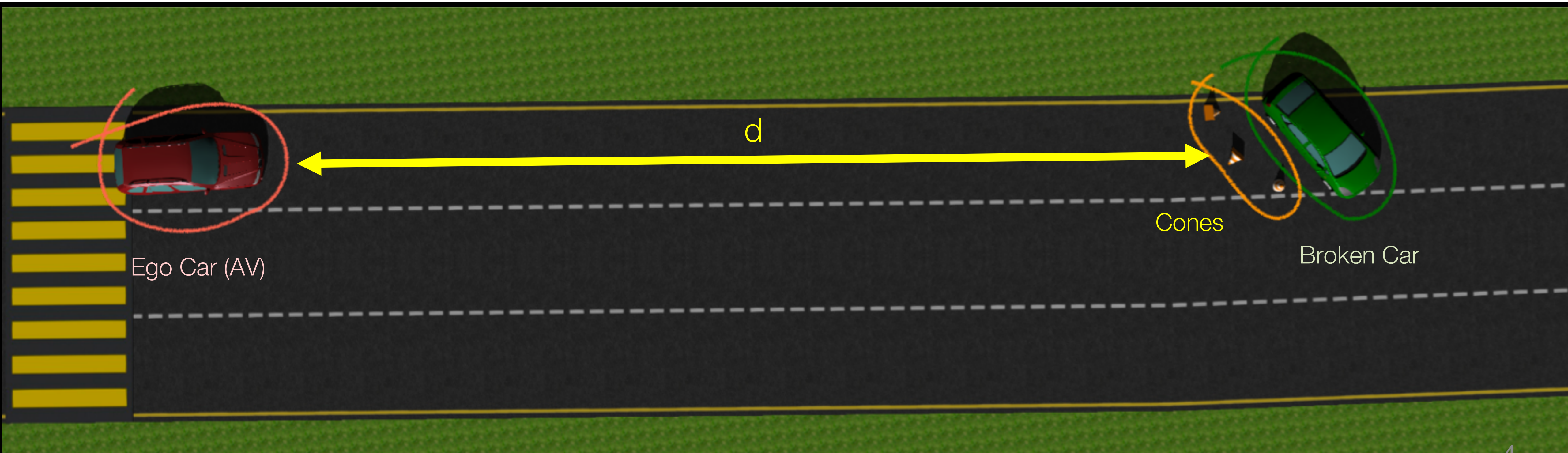
- Specifying initial conditions and parameters for simulations
  - **Falsification of cyber-physical systems (collision-avoidance case study)**
  - Analysis & retraining for ML-based systems (runway tracking case study)

- Specifying behaviors of *dynamic agents* which can react to their environment
- *Composing* scenarios in space and time
  - Falsification in dynamic environments (pedestrian scenario case study)

Case Study: Falsifying a Collision-Avoidance System

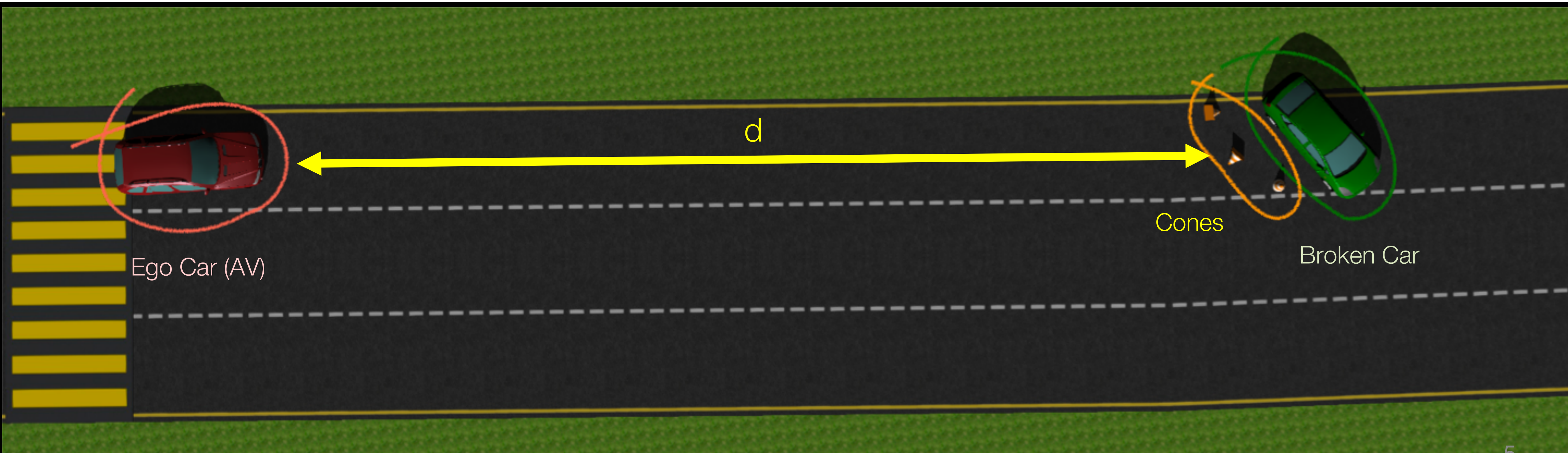






Case Study: Falsifying a Collision-Avoidance System

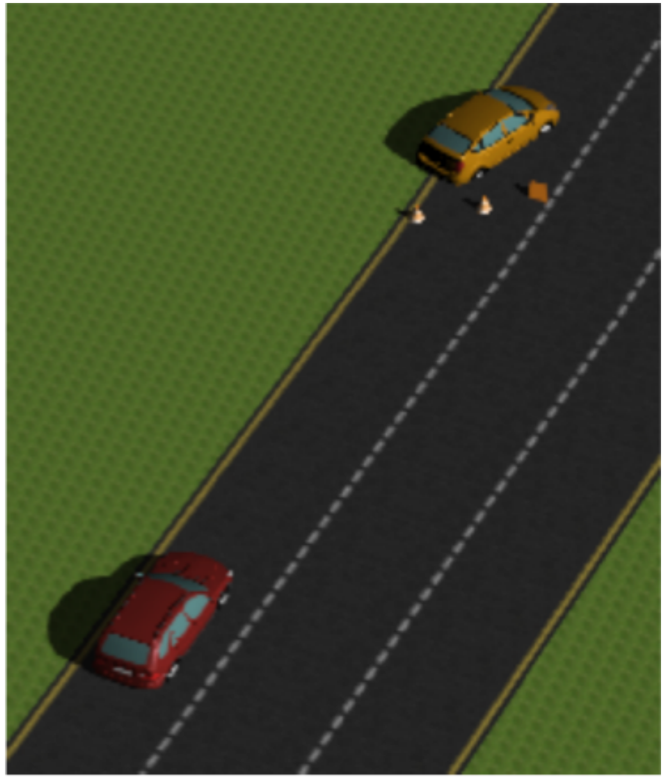
- Question: does the system always avoid a collision?
- *Falsification:* automated search for inputs causing the system to violate its specification.



Using Scenic to Generate Initial Scenes

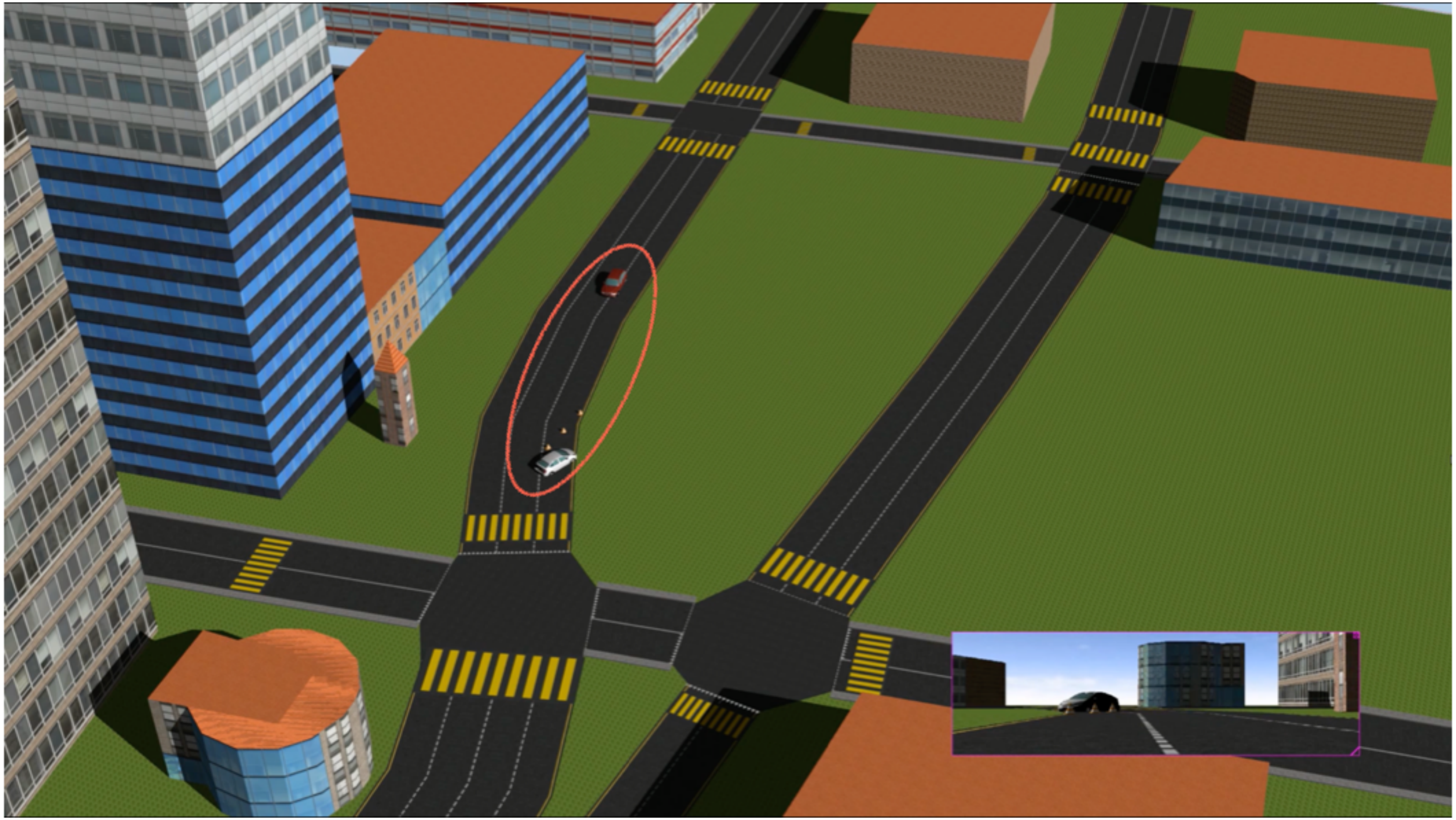
- A scene can be the initial condition for a simulation

```
# 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
```

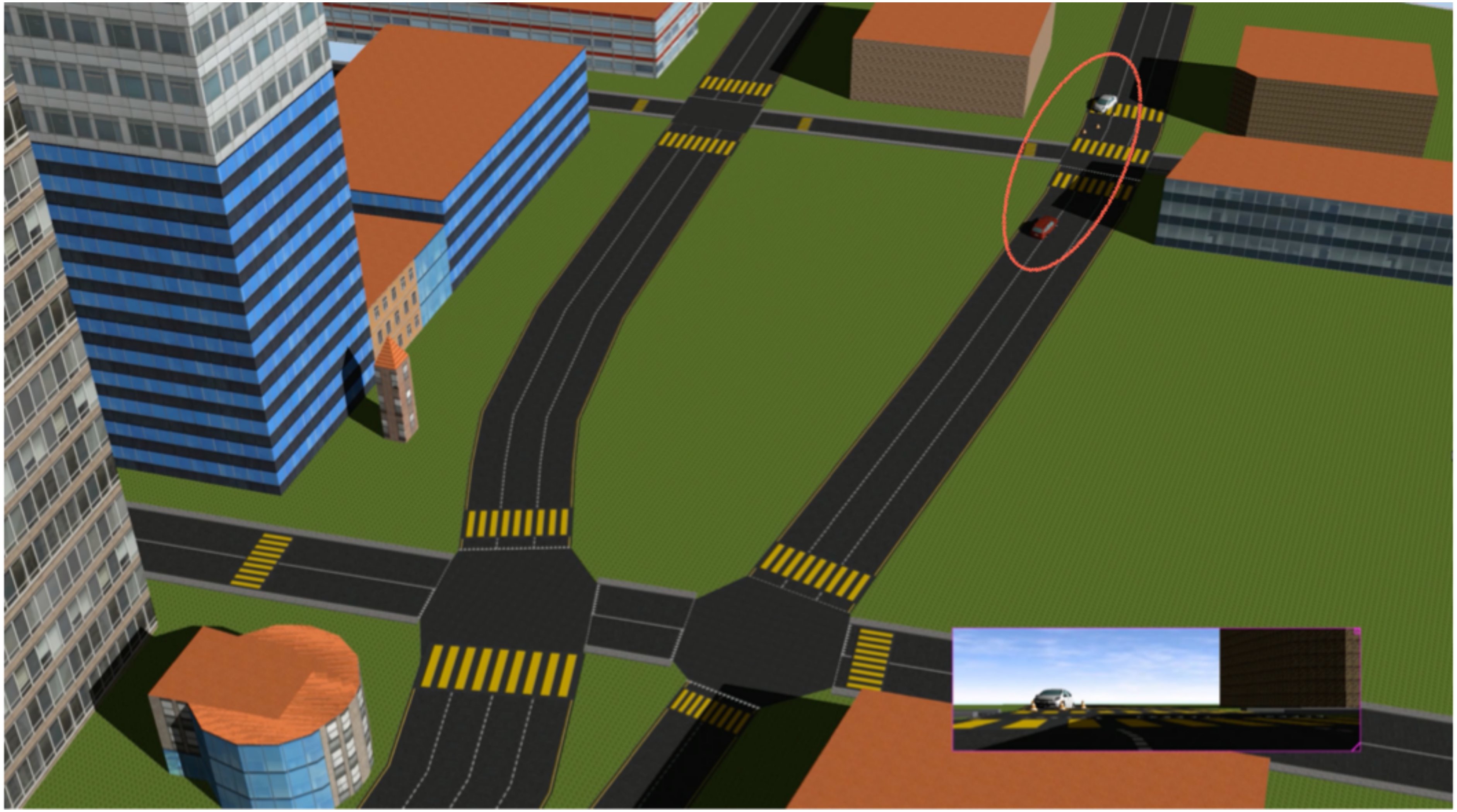


- Can also include parameters for controllers (e.g. reaction time, how quickly to swerve)

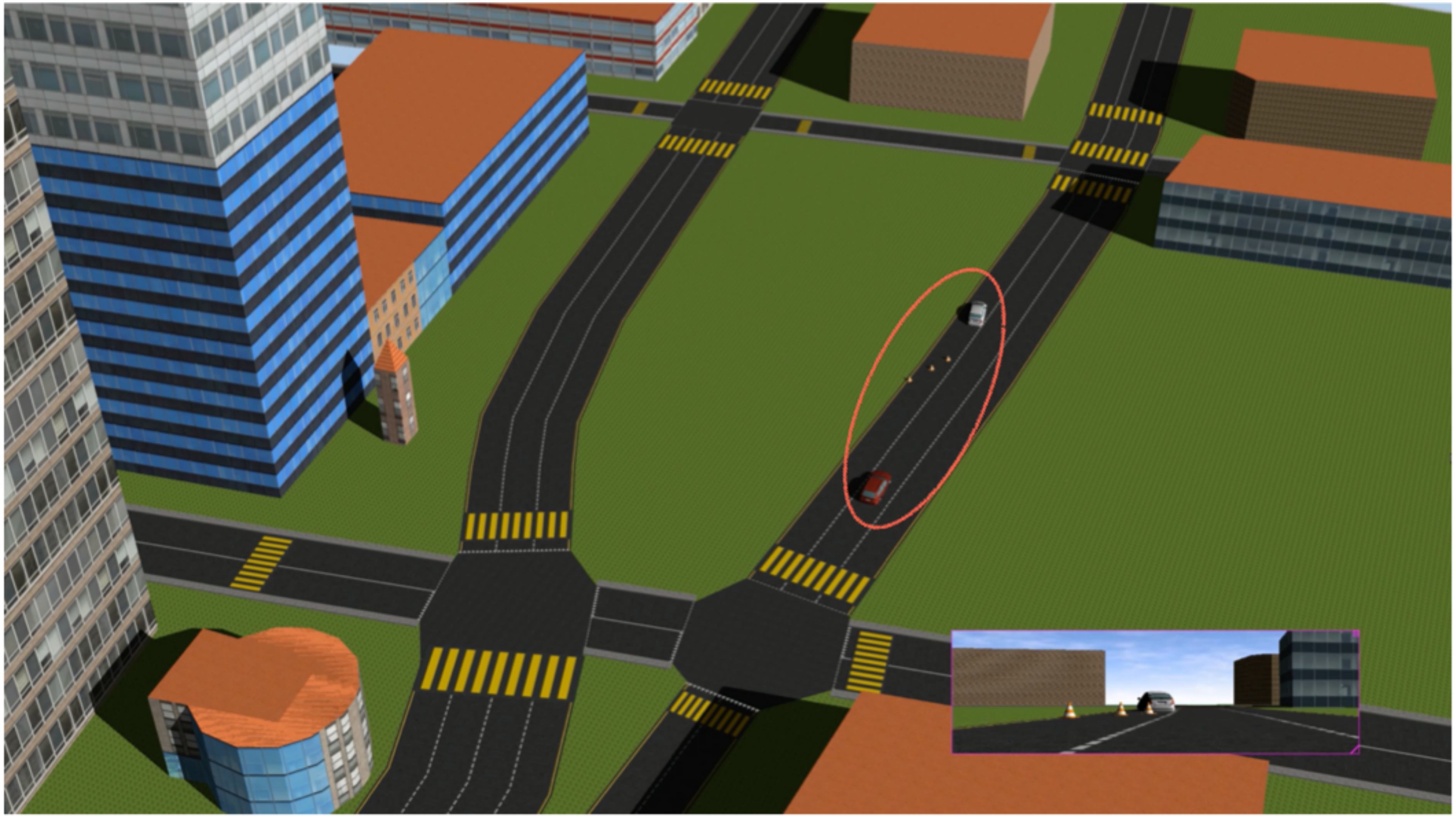
Using Scenic to Generate Initial Scenes



Using Scenic to Generate Initial Scenes



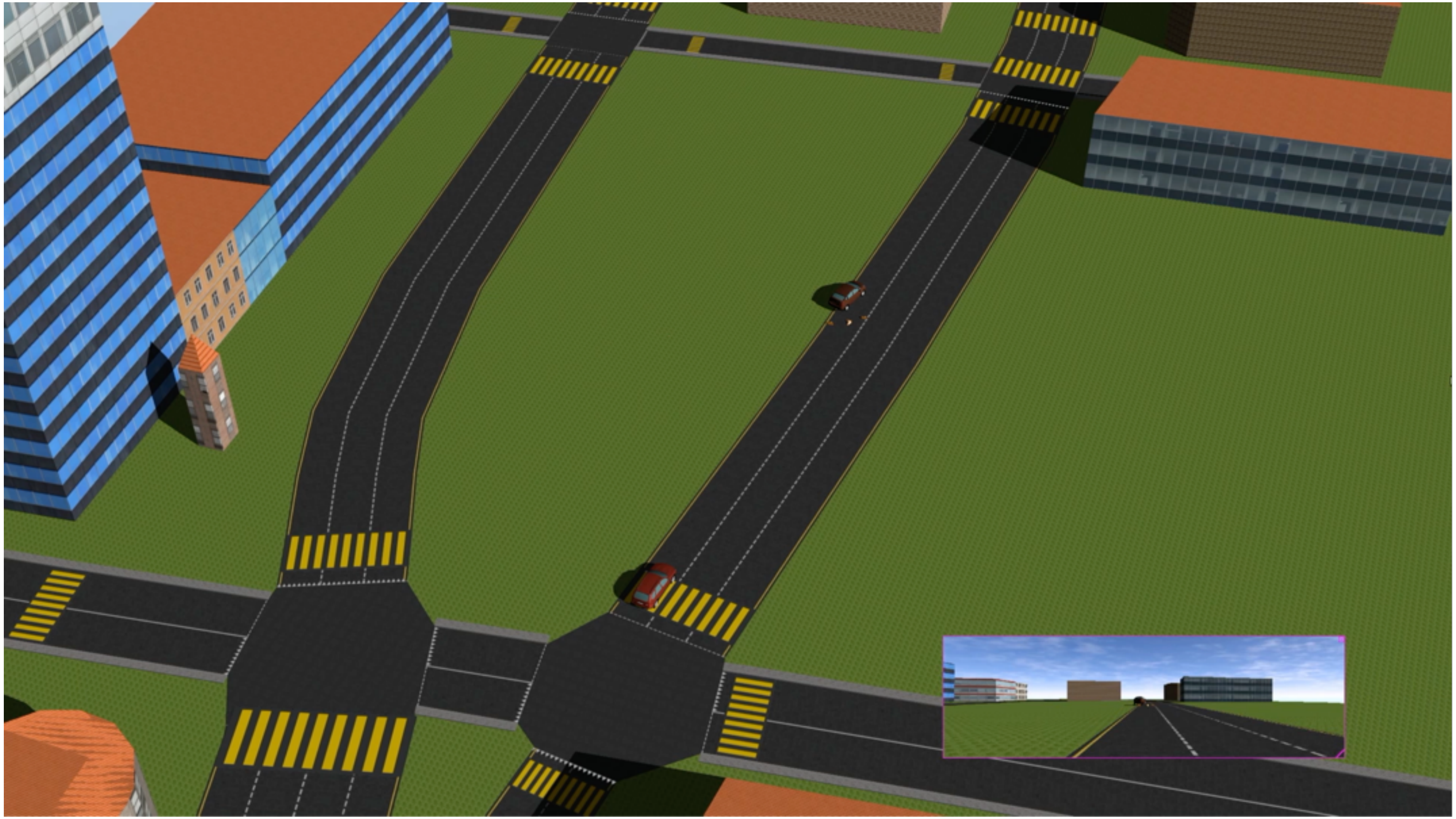
Using Scenic to Generate Initial Scenes



Setting up Falsification in VerifAI

```
# Define semantic feature space
control_params = Struct({
        'x_init': Box([-0.05, 0.05]),
        'cruising_speed': Box([10.0, 20.0]),
        'reaction_time': Box([0.7, 1.00])
    })
env_params = Struct({
        'broken_car_color': Box([0.5, 1], [0.25, 0.75], [0, 0.5]),
        'broken_car_rotation': Box([5.70, 6.28])
    })
sample_space = {'control_params':control_params, 'env_params':env_params}

# Use sampler based on cross-entropy optimization
sampler_type = 'ce'

# System-level specification to falsify, as a Metric Temporal Logic formula
specification = ["G(collisioncone0 & collisioncone1 & collisioncone2)"]

# Create and run falsifier
falsifier = mtl_falsifier(sample_space=sample_space, sampler_type=sampler_type,
                          specification=specification)
falsifier.run_falsifier()
```

Falsification



Outline

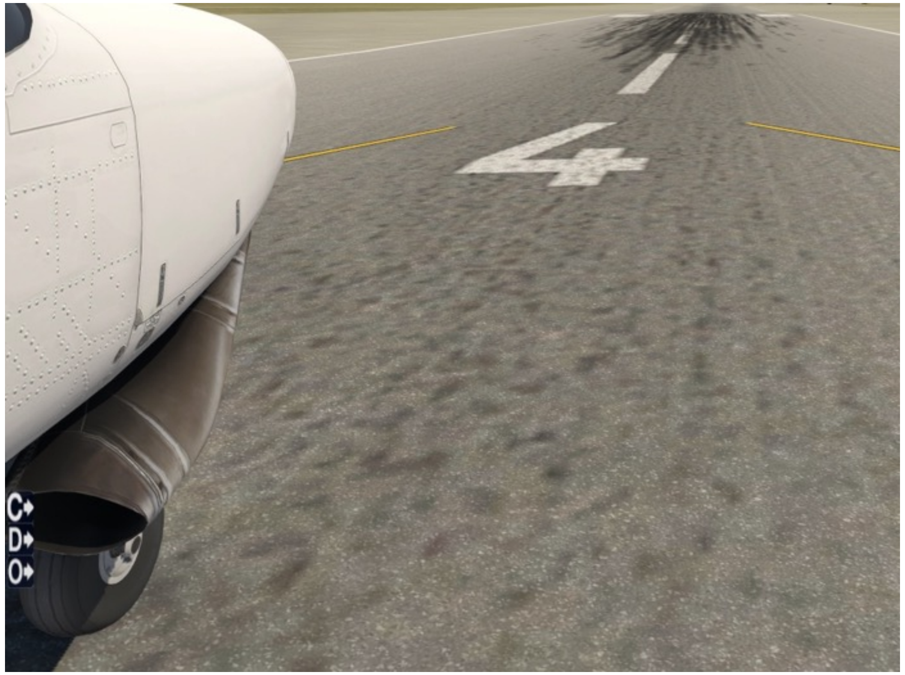
Scenic can describe *dynamic scenarios* which evolve over time.

- Specifying initial conditions and parameters for simulations
  - Falsification of cyber-physical systems (collision-avoidance case study)
  - **Analysis & retraining for ML-based systems (runway tracking case study)**
- Specifying behaviors of *dynamic agents* which can react to their environment
- *Composing* scenarios in space and time
  - Falsification in dynamic environments (pedestrian scenario case study)

A Full Design Iteration using Scenic & VerifAI

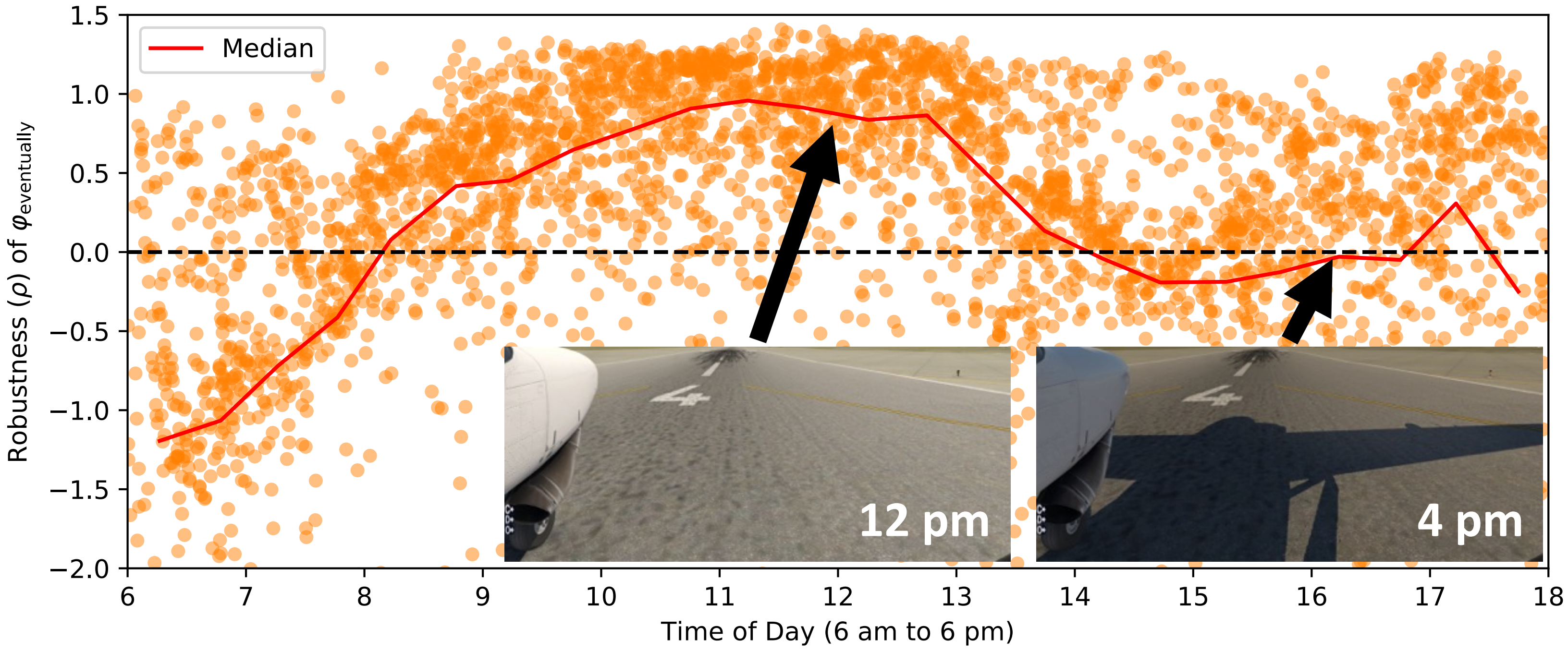
- In addition to discovering failures, VerifAI can help debug and fix them

- Industrial case study on **TaxiNet**, a NN-based taxiing system [CAV 2020]
  - **Modeling** runway scenarios in SCENIC
  - **Falsifying** the system, finding scenarios when it violates its specification
  - **Debugging** to find distinct failures and their root causes
  - **Retraining** the system to eliminate failures and improve performance



Counterexample Analysis

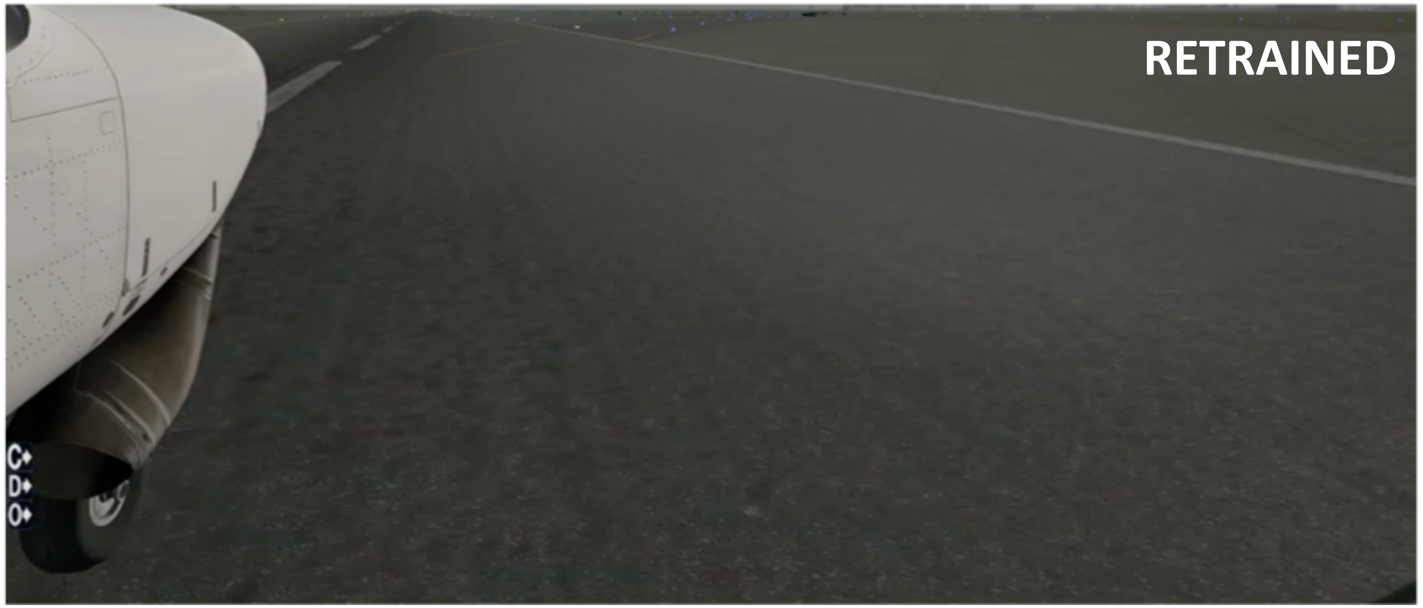
- Falsification found several types of failures, e.g. sensitivity to time



- Follow-up experiments confirmed root cause is the plane's shadow

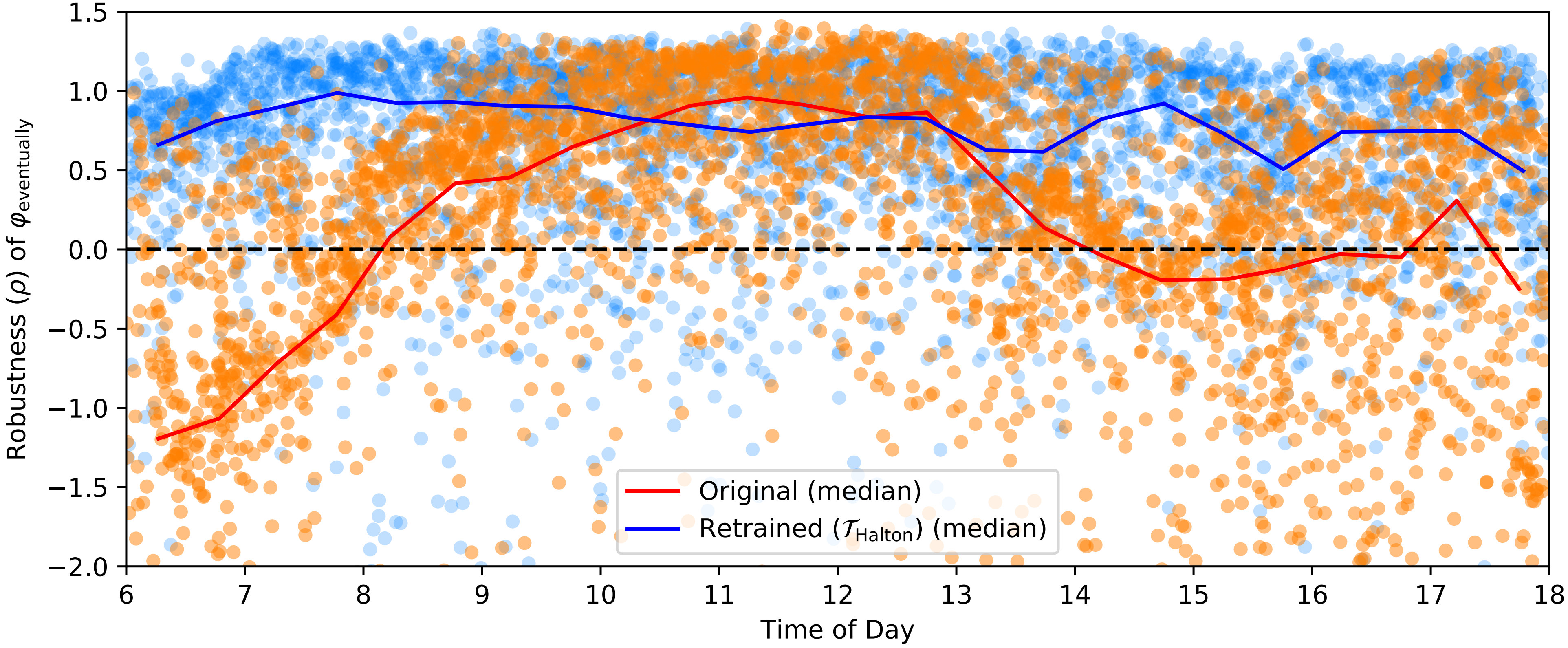
Retraining

- Use VERIFAI to generate a new training set (same size as original)
- Obtained much better performance



Retraining

- Eliminated dependence on time of day



- Used cross-entropy method to *learn* good training distributions

Outline

Scenic can describe *dynamic scenarios* which evolve over time.

- Specifying initial conditions and parameters for simulations
  - Falsification of cyber-physical systems (collision-avoidance case study)
  - Analysis & retraining for ML-based systems (runway tracking case study)
- **Specifying behaviors of *dynamic agents* which can react to their environment**
- *Composing* scenarios in space and time
  - Falsification in dynamic environments (pedestrian scenario case study)

Going Beyond Initial Conditions

- Scenic can also describe *dynamic agents* which take actions over time, reacting to a changing environment

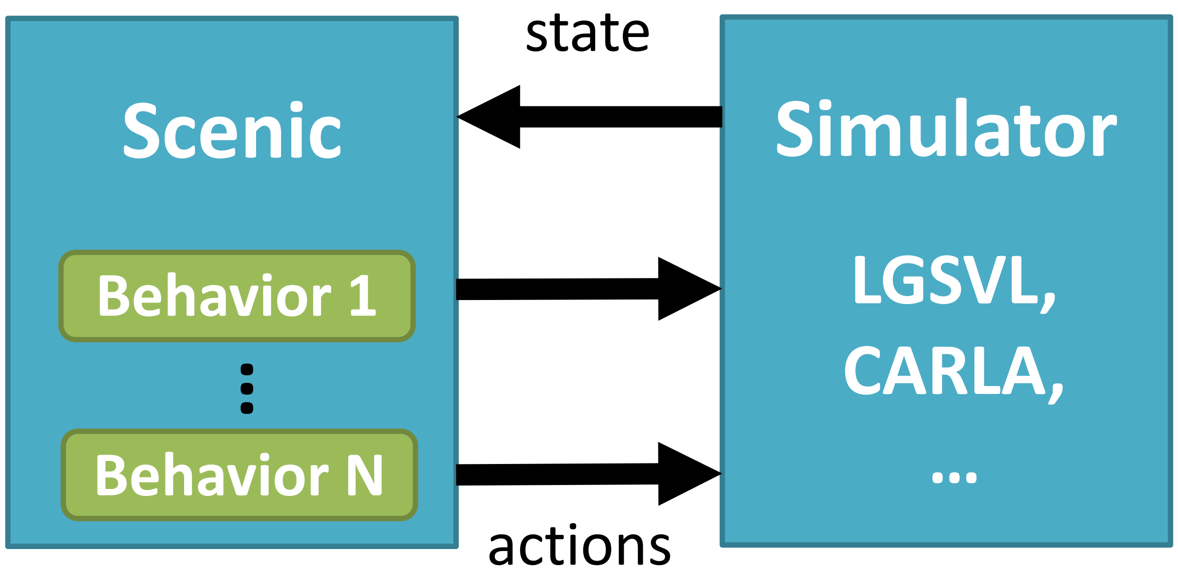
- Example: ”a badly-parked car, which suddenly pulls into the road as the ego car approaches”

- The dynamic actions of the car are specified by giving it a *behavior*

```
parkedCar = Car left of spot by 0.5,
                facing badAngle relative to roadDirection,
                with behavior PullIntoRoad
```

Behaviors and Actions

- Behaviors are functions running in parallel with the simulation, issuing *actions* at each time step
  - e.g. for AVs: set throttle, set steering angle, turn on turn signal
  - Provided by a Scenic library for the driving domain
  - Abstract away details of simulator interface
- Behaviors can access the state of the simulation and make choices accordingly




```
behavior FollowLaneBehavior(lane):
    while True:
        throttle, steering = ...
        take (SetThrottleAction(throttle),
              SetSteerAction(steering))
```

Behaviors and Actions

- Behaviors can call each other
  - Can define libraries of behaviors
- Scenic's driving domain library
  - Classes for cars, pedestrians, etc.
  - Controllers for simple maneuvers
  - API for accessing road network geometry

```
behavior FollowLaneBehavior(lane):
    while True:
        throttle, steering = ...
        take (SetThrottleAction(throttle),
              SetSteerAction(steering))
```

```
behavior PullIntoRoad():
    while (distance from self to ego) > 15:
        wait
    do FollowLaneBehavior(ego.lane)
```

Automatically rejects simulation if ego is not in a lane

A Simple Scenario: CARLA Challenge Scenario #2

- Based on NHTSA pre-crash typology scenario 25
  - Lead car decelerates because of obstacle; ego car must brake

```
behavior LeadCarBehavior():
    brake = Range(0.9, 1)
    throttle = Range(0.5, 1)
    brakingDistance = Range(5, 10)
    while True:
        if (distance from self to trash) <= brakingDistance:
            take SetBrakeAction(brake), SetThrottleAction(0)
        else:
            take SetBrakeAction(0), SetThrottleAction(throttle)
```

A Simple Scenario: CARLA Challenge Scenario #2

- Based on NHTSA pre-crash typology scenario 25
  - Lead car decelerates because of obstacle; ego car must brake

```
behavior LeadCarBehavior():
    brake = Range(0.9, 1)
    throttle = Range(0.5, 1)
    brakingDistance = Range(5, 10)
    while True:
        if (distance from self to trash) <= brakingDistance:
            take SetBrakeAction(brake), SetThrottleAction(0)
        else:
            take SetBrakeAction(0), SetThrottleAction(throttle)

lane = Uniform(*network.lanes)

trash = Trash on lane.centerline

leadCar = Car following roadDirection from trash for Range(-15, -30)

ego = Car following roadDirection from leadCar for Range(-10, -30)
```

A Simple Scenario: CARLA Challenge Scenario #2

More Advanced Temporal Constructs

- *Interrupts* allow adding special cases to behaviors without modifying their code

```
behavior FollowLeadCar(safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when (distance to other) < safety_distance:
        do CollisionAvoidance()
```

- *Temporal requirements* and *monitors* allow enforcing constraints during simulation

```
require always taxi in lane
require eventually ego can see pedestrian
```

A Worked Example

- OAS Voyage Scenario 2-2-XX-CF-STR-CAR:02
- Lead car periodically stops and starts; ego car must brake to avoid collision
- Cross-platform scenario works in CARLA and LGSVL

```
behavior FollowLeadCar(safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when (distance to other) < safety_distance:
        do CollisionAvoidance()

behavior StopsAndStarts():
    stop_delay = Range(3, 6) seconds
    last_stop = 0
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when simulation.currentTime - last_stop > stop_delay:
        do FullBraking() for 5 seconds
        last_stop = simulation.currentTime

ego = Car with behavior FollowLeadCar(safety_distance=10)
other = Car ahead of ego by 10,
            with behavior StopsAndStarts

require (Point ahead of ego by 100) in road

terminate when ego.lane is None
```

A Worked Example: CARLA

A Worked Example: LGSVL

Outline

Scenic can describe *dynamic scenarios* which evolve over time.

- Specifying initial conditions and parameters for simulations
  - Falsification of cyber-physical systems (collision-avoidance case study)
  - Analysis & retraining for ML-based systems (runway tracking case study)

- Specifying behaviors of *dynamic agents* which can react to their environment
- ***Composing* scenarios in space and time**
  - Falsification in dynamic environments (pedestrian scenario case study)

Composing Scenarios

- Scenic allows scenarios to be defined modularly and combined into more complex scenarios

- Parallel, sequential, and more complex forms of composition

```
import StopAndStart, BadlyParkedCar

scenario StopStartWithParkedCar():
    compose:
        do StopAndStart(), BadlyParkedCar()


scenario StopStartThenParkedCar():
    compose:
        do StopAndStart()
        do BadlyParkedCar()


scenario StopStartThenParkedCar():
    compose:
        try:
            do StopAndStart()
        override when ...:
            do BadlyParkedCar()
```

Outline

Scenic can describe *dynamic scenarios* which evolve over time.

- Specifying initial conditions and parameters for simulations
  - Falsification of cyber-physical systems (collision-avoidance case study)
  - Analysis & retraining for ML-based systems (runway tracking case study)

- Specifying behaviors of *dynamic agents* which can react to their environment
- *Composing* scenarios in space and time
  - **Falsification in dynamic environments (pedestrian scenario case study)**

ITSC Case Study

- Pedestrian which crosses the road, hesitating for some amount of time

- Describe trajectory with 3 parameters:
  - Threshold distance
  - Time until hesitation
  - Length of hesitation

```
behavior Hesitate():
    while ((distance from ego to self)
            > self.thresholdDistance):
        wait
    do WalkForward() for self.walkTime
    do Stop() for self.hesitateTime
    do WalkForward()

ped = Pedestrian at 14.9@208.2,
          facing 80 deg relative to ego,
          with behavior Hesitate,
          with thresholdDistance Range(10, 20),
          with walkTime Range(1.5, 3.5),
          with hesitateTime Range(1, 3)
```

Scenic and VerifAI: Summary of Features and Use Cases

- Classes, Objects, Geometry, and Distributions
- Local Coordinate Systems
- Readable, Flexible Specifiers
- Declarative Hard & Soft Constraints
- Externally-Controllable Parameters
- Agent Actions and Behaviors, Interrupts, Termination
- Monitors, Temporal Constraints
- Scenario Composition

…

- Synthetic Data Generation
- Test Generation, Fuzz Testing
- Specifying (Safety) Requirements and Metrics
- Falsification (directed search for bugs, edge cases, etc.)
- Debugging and Error Explanation
- Data Augmentation
- Goal-Directed Parameter Synthesis

…

Documentation on Scenic and VerifAI – linked from GitHub

Scenic
latest

Search docs

- Getting Started with Scenic
- Scenic Tutorial
- Guide to Scenic Syntax
- Scenic Syntax Reference
- Supported Simulators
- Interfacing to New Simulators
- Scenic Internals
- Publications Using Scenic
- Credits

Welcome to Scenic's documentation!

Scenic is a domain-specific probabilistic programming language for modeling the environments of cyber-physical systems like robots and autonomous cars. A Scenic program defines a distribution over *scenes*, configurations of physical objects and agents; sampling from this distribution yields concrete scenes which can be simulated to produce training or testing data.

Scenic was designed and implemented by Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. For a description of the language and some of its applications, see our PLDI 2019 paper; a more in-depth discussion is in Chapters 5 and 8 of this thesis. Our publications page lists additional papers using Scenic.

VerifAI
latest

Search docs

- Getting Started with VerifAI
- Basic Usage
- Tutorial / Case Studies
- Feature APIs in VerifAI
- Search Techniques
- Publications Using VerifAI

Docs » Welcome to VerifAI's documentation!   Edit on GitHub

Welcome to VerifAI's documentation!

VerifAI is a software toolkit for the formal design and analysis of systems that include artificial intelligence (AI) and machine learning (ML) components. VerifAI particularly seeks to address challenges with applying formal methods to perception and ML components, including those based on neural networks, and to model and analyze system behavior in the presence of environment uncertainty. The current version of the toolkit performs intelligent simulation guided by formal models and specifications, enabling a variety of use cases including temporal-logic falsification (bug-finding), model-based systematic fuzz testing, parameter synthesis, counterexample analysis,

Thank you!

- Scenic and VerifAI are available open source with documentation at:

  https://github.com/BerkeleyLearnVerify/Scenic
  https://github.com/BerkeleyLearnVerify/VerifAI

- Please complete our (short!) post-webinar survey at

  https://forms.gle/79sMYFH8NrTLB9V97

  You can also join our mailing list by completing this survey.

- Send us your feedback!

QUESTIONS?

Acknowledgments: Contributors, Co-authors, Collaborators

UC Berkeley
- Johnathan Chiu
- Tommaso Dreossi
- Shromona Ghosh
- Francis Indaheng
- Sebastian Junges
- Kevin Li
- Yash Vardhan Pant
- Hadi Ravanbakhsh
- Jay Shenoy
- Hazem Torfah
- Marcell Vazquez-Chanlatte
- Kesav Viswanadha
- Xiangyu Yue

UC Berkeley
- Kurt Keutzer
- Alberto Sangiovanni-Vincentelli
- Pravin Varaiya
- Alex Kurzhanskiy

UC Santa Cruz
- Ellen Kalvan

Boeing
- Dragos Margineantu
- Denis Osipychev

AAA NCNU
- Xantha Bruso
- Paul Wells

LG Electronics
- Steve Lemke
- Shalin Mehta
- Qiang Lu

NASA Ames
- Corina Pasareanu
- Divya Gopinath

Thank you!