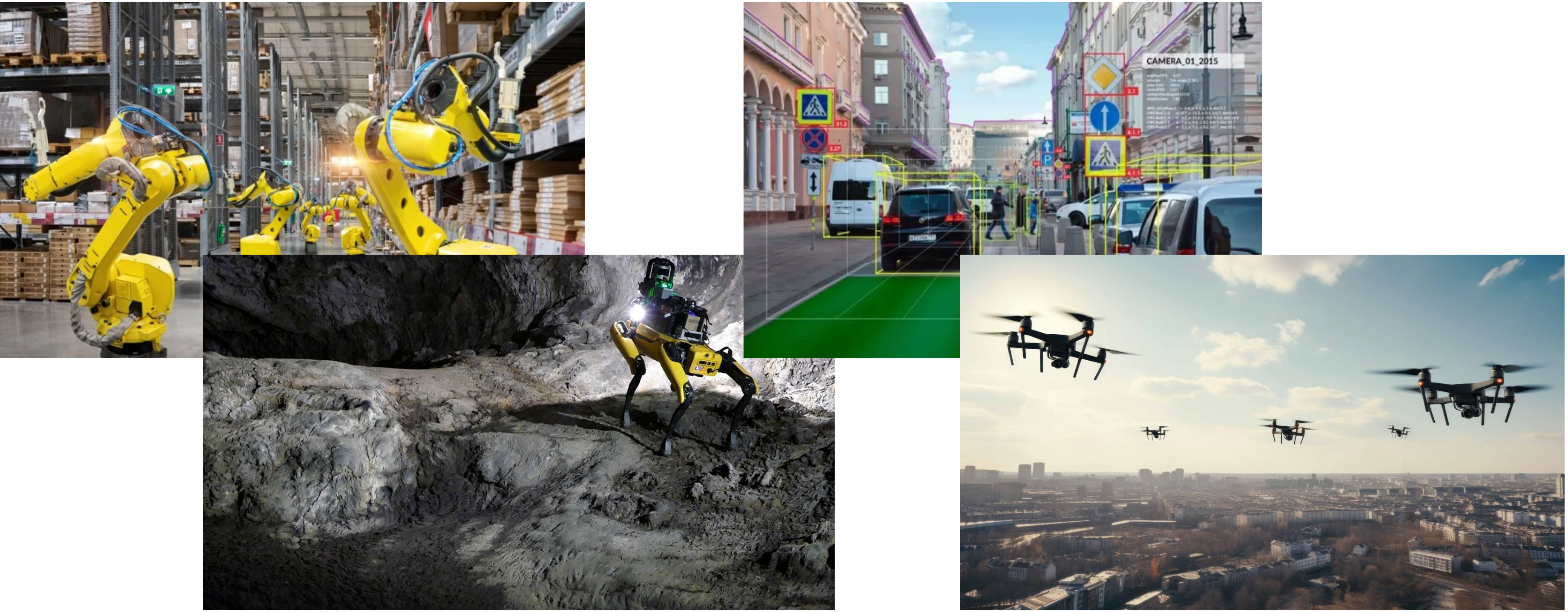
Scenic for ROS: A Probabilistic Programming System for World Modeling and Data Generation for Robotics

Hazem Torfah[1], Edward Kim[2], Daniel J. Fremont[3], Sanjit A. Seshia[2]

[1] Chalmers University of Technology and University of Gothenburg
[2] University of California, Berkeley
[3] University of California, Santa Cruz

@ROSCON Oct 23, 2024 Odense, Denmark

Applications and Complex Environments



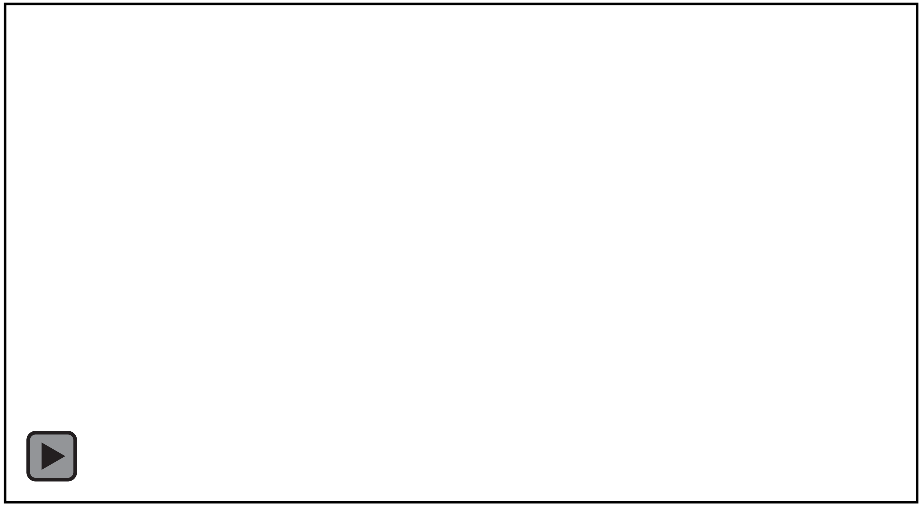
We need models that capture the diversity of environments and that allow us to guide data generation towards realistic and important scenarios

Scenic: A Probabilistic Language for World Modeling

A probabilistic programming language:

A Scenic program defines a *probabilistic scenario*,
a ***distribution*** over configurations of physical objects/agents and their behaviors over time

```
behavior EgoBehavior(target_speed=25, safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=target_speed)
    interrupt when withinDistanceToObjsInLane(self, safety_distance):
        take SetThrottleAction(0), SetBrakeAction(1)

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

ego = Car, with behavior EgoBehavior(safety_distance=10)

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
parkedCar = Car left of spot by 0.5,
                facing badAngle relative to roadDirection,
                with behavior PullIntoRoad()
```

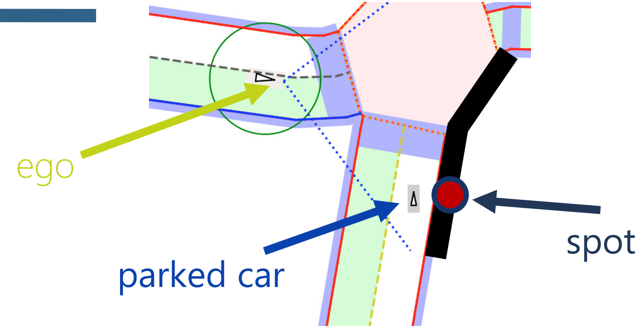


[1] Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, "Scenic: A Language for Scenario Specification and Scene Generation," PLDI, 2019
[2] Daniel J Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, "Scenic: A Language for Scenario Specification and Data Generation," Machine Learning Journal, 2022

Scenic: Program Structure

```
behavior EgoBehavior(target_speed=25, safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=target_speed)
    interrupt when withinDistanceToObjsInLane(self, safety_distance):
        take SetThrottleAction(0), SetBrakeAction(1)

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

Agent behaviors




```
ego = Car, with behavior EgoBehavior(safety_distance=10)

spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * Range(10, 20) deg
parkedCar = Car left of spot by 0.5,
                facing badAngle relative to roadDirection,
                with behavior PullIntoRoad()
```

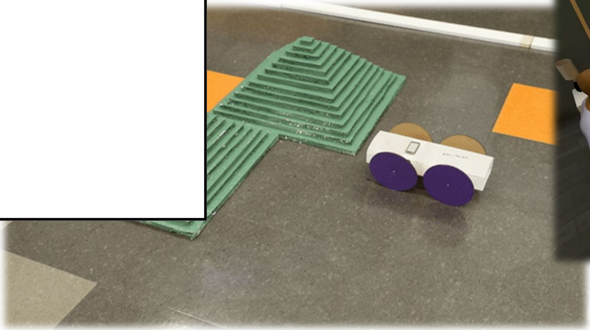
Initial states

Application Domains

Autonomous Driving

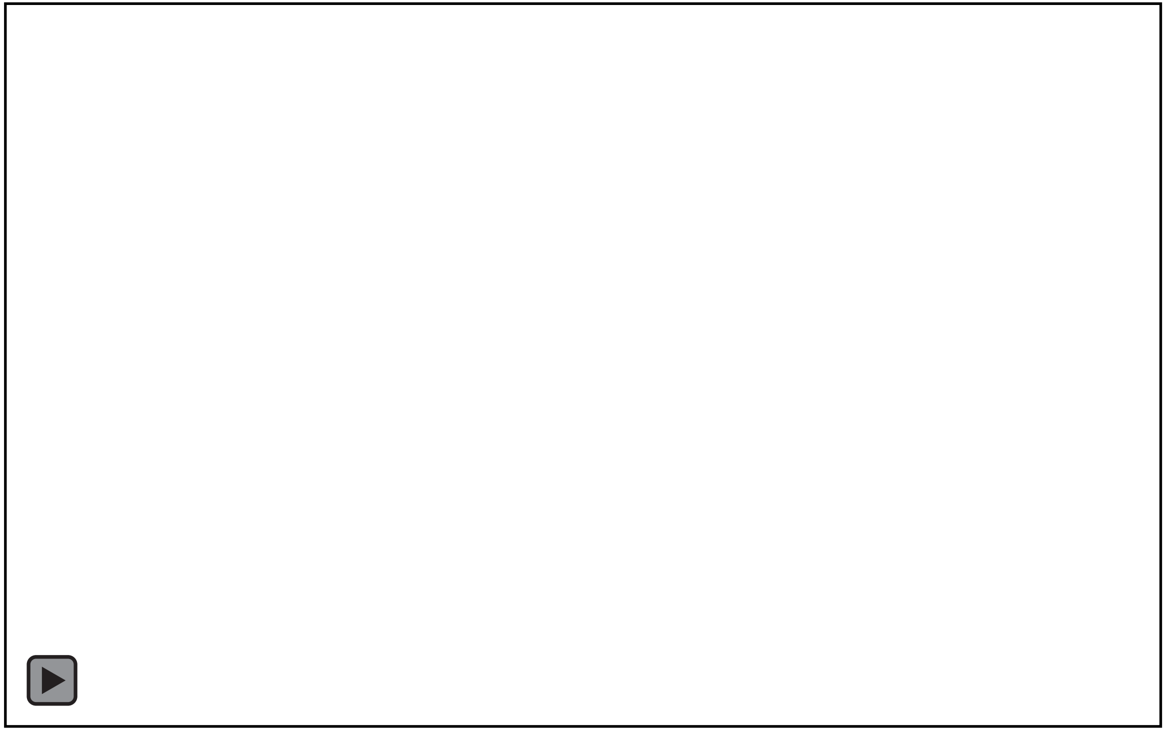
Aviation

Robotics



Testing and Falsification: Toyota Human Support Robot (HSR)

Automatically finding "edge cases" that lead to failures / unsafe states



Task: Fetch a bottle of water, while avoiding obstacles

Environment Model:
Variations of corridor layout with chairs

Simulator: Gazebo + ROS

Fail cases:
- HSR failing to avoid chairs when moving backwards
- HSR struggles to plan path

Testing and Falsification: Toyota Human Support Robot (HSR)

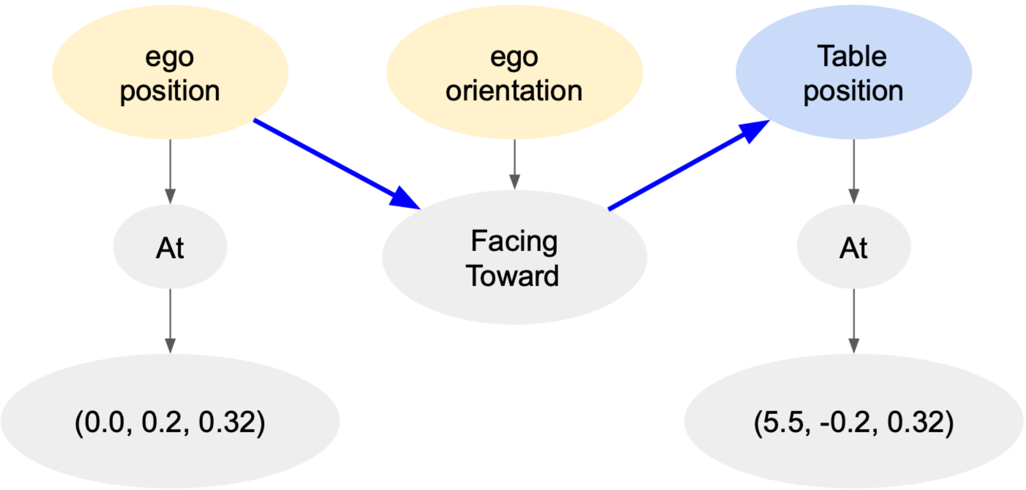
```
w RectangularRegion((2,0,0), 1.57, 2.3, 9)

itchenTable at (5.5, -0.2, 0.32)

Bottle on table

Chair on corridor,
ith yaw Range(0, 2 * math.pi)

Chair on corridor,
ith yaw Range(0, 2 * math.pi)

_Robot at (0, 0, 0.32),
acing toward table
ith behavior RetrieveObject(bottle)

require (distance from chair1 to chair2) > 1.5 # meters
```

Testing and Falsification: Toyota Human Support Robot (HSR)

```
model scenic.simulators.hsr.model
from scenic.simulators.hsr.behaviors import *
from scenic.simulators.hsr.actions import *
import math
import time

behavior RetrieveObject(obj):
    ego_initial_position = self.position

    try:
        do MoveToTarget(obj)
        do PickUp(obj)
        do MoveToTarget(ego_initial_position)
        terminate

    interrupt when (self.holdingObject and self.distanceToClosest('chair')) < 1.0
        take Stop()
        do GripperSetAngle(1.0) # open the gripper to drop the object
```

```
corridor = new RectangularRegion((2,0,0), 1.57, 2.3, 9)

 = new KitchenTable at (5.5, -0.2, 0.32)

e = new Bottle on table

1 = new Chair on corridor,
        with yaw Range(0, 2 * math.pi)

2 = new Chair on corridor,
        with yaw Range(0, 2 * math.pi)

new HSR_Robot at (0, 0, 0.32),
        facing toward table
        with behavior RetrieveObject(bottle)

require (distance from chair1 to chair2) > 1.5 # meters
```

Sampling Concrete Scenarios from a Scenic Program

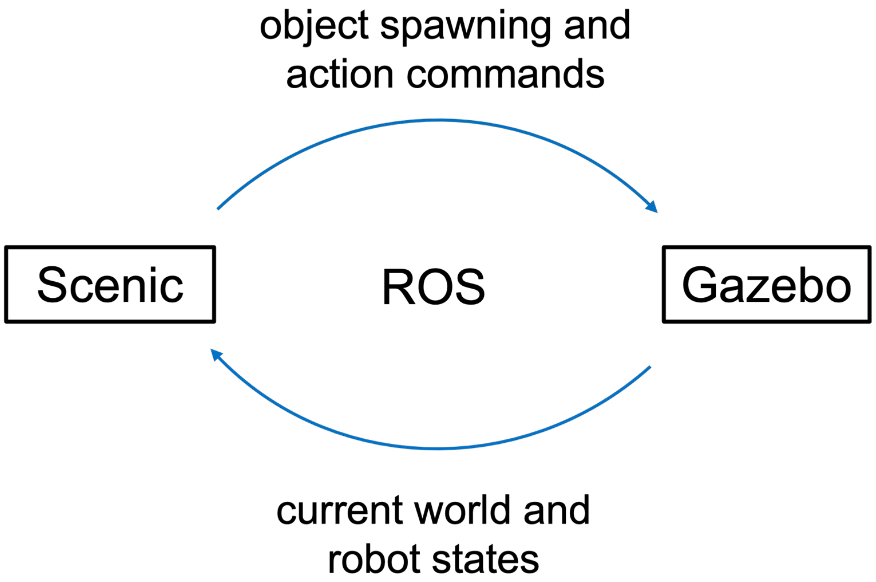



```
corridor = new RectangularRegion((2,0,0), 1.57, 2.3, 9)

table = new KitchenTable at (5.5, -0.2, 0.32)

bottle = new Bottle on table

chair1 = new Chair on corridor,
            with yaw Range(0, 2 * math.pi)

chair2 = new Chair on corridor,
            with yaw Range(0, 2 * math.pi)

ego = new HSR_Robot at (0, 0, 0.32),
            facing toward table
            with behavior RetrieveObject(bottle)

require (distance from chair1 to chair2) > 1.5 # meters
```

Scenic Interface Overview

- Scenic sends over commands through ROS

- This includes where to spawn objects at the beginning of the scene and what actions the robot should take



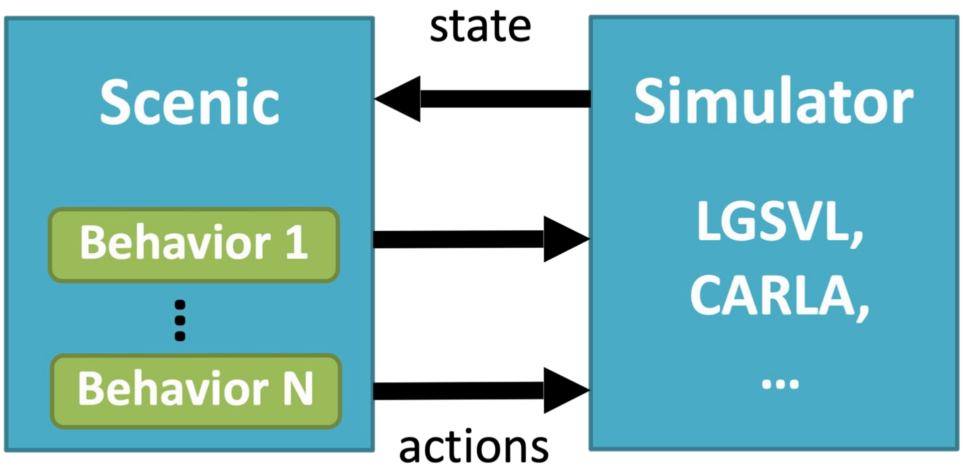
- Gazebo receives the commands, executes, and relays the world state info back to Scenic through ROS

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 stateof the simulation and makechoices accordingly

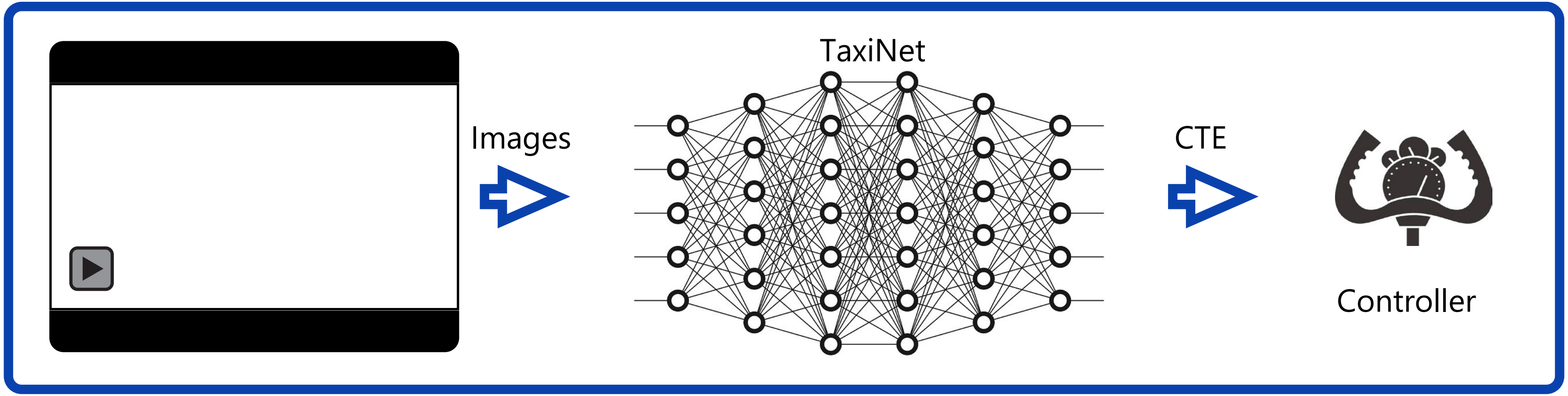



```
behavior FollowLaneBehavior(lane):
    while True:
        throttle, steering = ...
        take (SetThrottleAction(throttle),
              SetSteerAction(steering))
```

Use Cases

Synthetic Data Generation and Retraining: TaxiNet

TaxiNet: A neural network developed by Boeing that uses camera images to estimate the cross track error (CTE), i.e., the distance to the centerline of the runway.



Plane must track centerline within 1.5 meters

Synthetic Data Generation and Retraining: TaxiNet

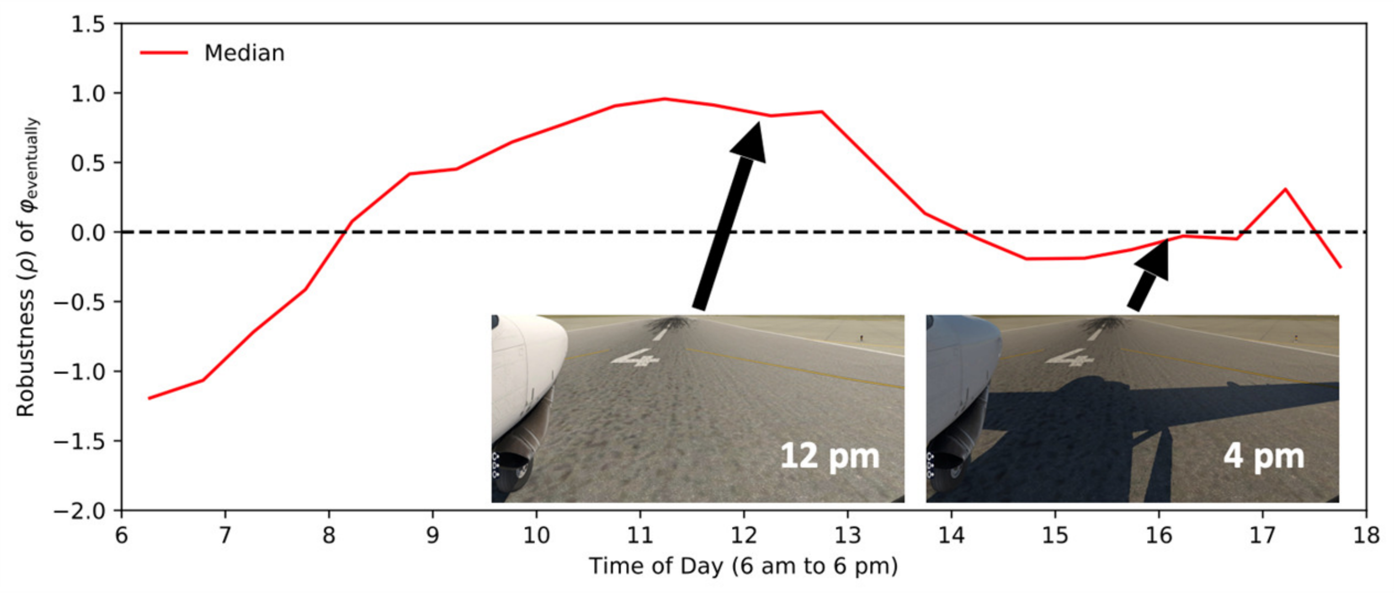
- Semantic features: time, clouds, rain, position/orientation of plane

```
# Time of day: from 6 am to 6 pm. (+8 to get GMT, as used by X-Plane)
param zulu_time = ((6, 18) + 8) * 60 * 60

# Rain: 1/3 of the time. Clouds: rain requires types 3-5; otherwise 0-5.
clouds_and_rain = Options({
    tuple([Uniform(0, 1, 2, 3, 4, 5), 0]): 2,  # no rain
    tuple([Uniform(3, 4, 5), (0.25, 1)]): 1    # 25% to 100% rain
})
param cloud_type = clouds_and_rain[0], rain_percent = clouds_and_rain[1]

# Plane: up to 8 m left/right, 2000 m down the runway, 30° left/right.
ego = Plane at (-8, 8) @ (0, 2000),
            facing (-30, 30) deg
```

- Falsification: out of ~4,000 simulations,
  - **45%** violated
  - **9%** left runway entirely

What went wrong? → Debugging the Root Cause

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



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

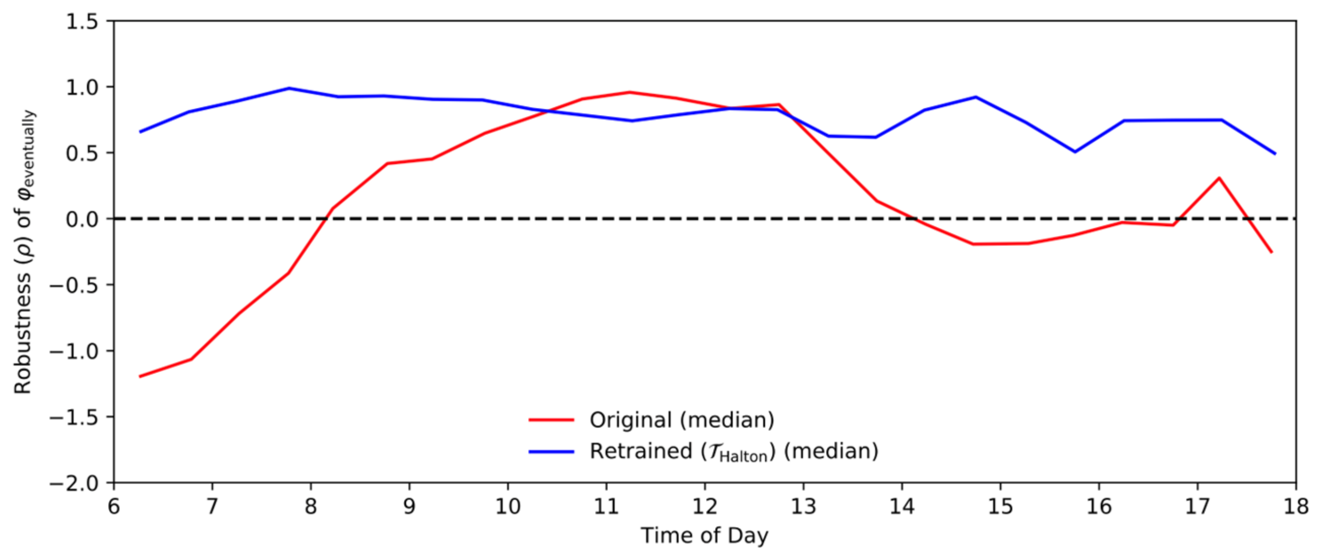
Retraining

- Generatea new training set (same size as original)

- Obtained much better performance
- 17% violated (vs. 45%)
- 0.6% left runway entirely(vs. 9%)

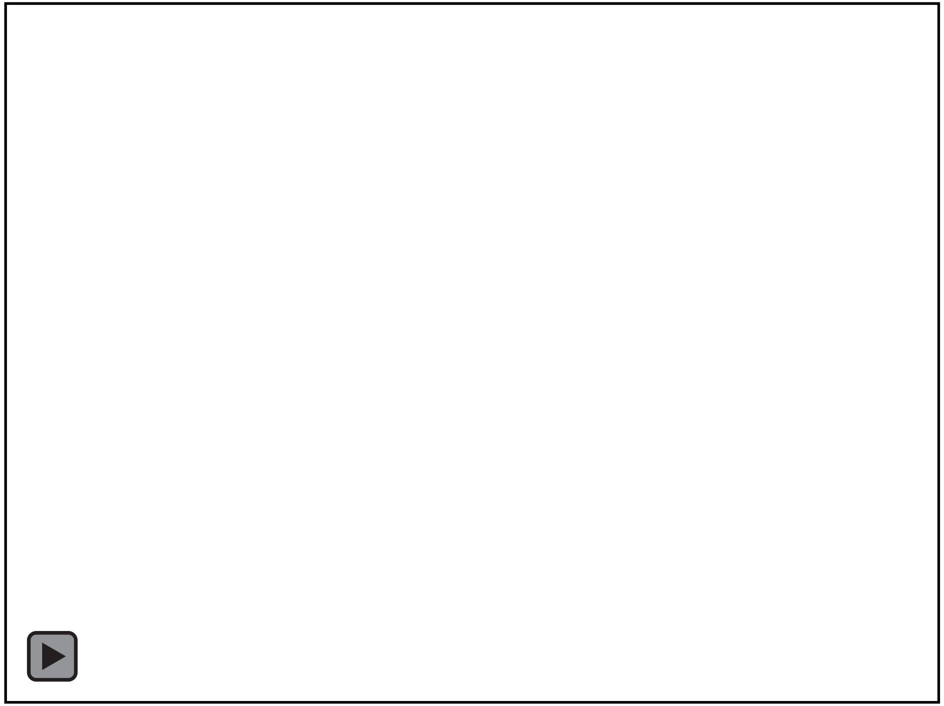
Retraining

- Eliminated dependence on time of day



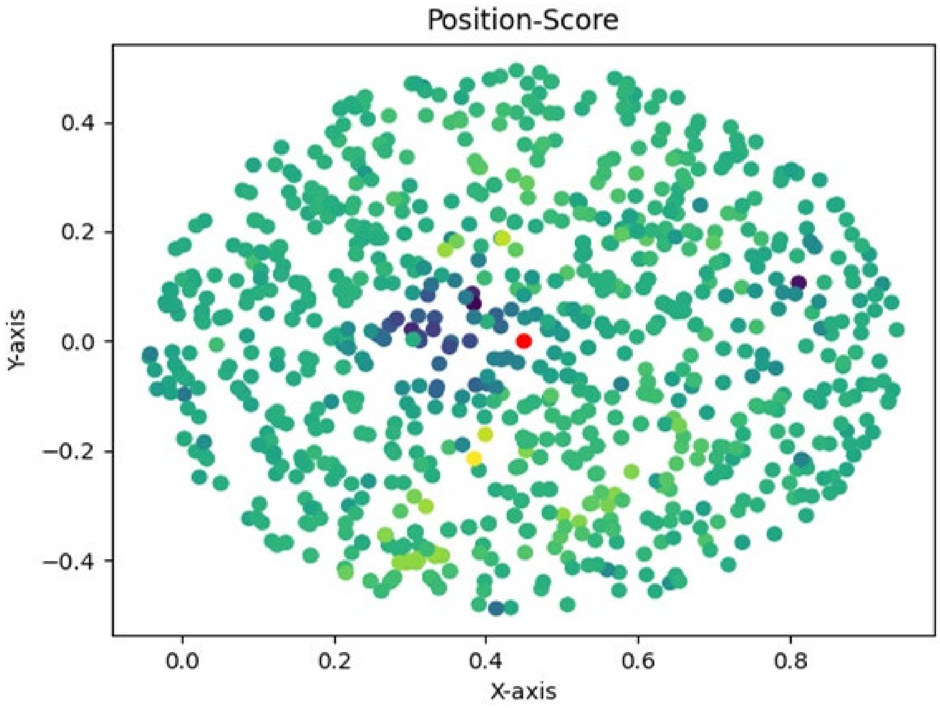
Testing/Training Reinforcement Learning Agents

- Franka Emika Panda arm simulated in Webots
- Deep RL controller trained using Scenic
- Task: reach target while avoiding obstacle
- Falsification shows goal positions just beyond the obstacle (red dot) are hardest for the controller

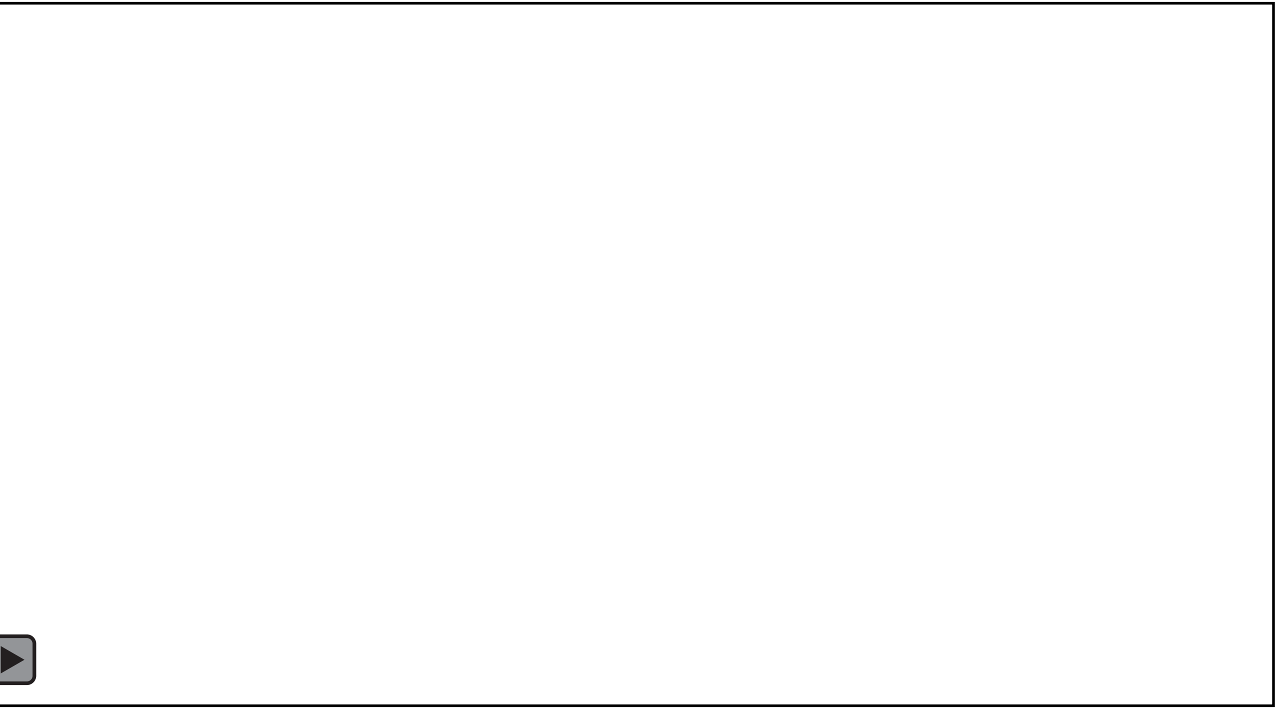


Testing/Training Reinforcement Learning Agents

- Franka Emika Panda arm simulated in Webots
- Deep RL controller trained using Scenic
- Task: reach target while avoiding obstacle
- Falsification shows goal positions just beyond the obstacle (red dot) are hardest for the controller



Reinforcement Learning in Multi-Agent Setting: Soccer Scenario

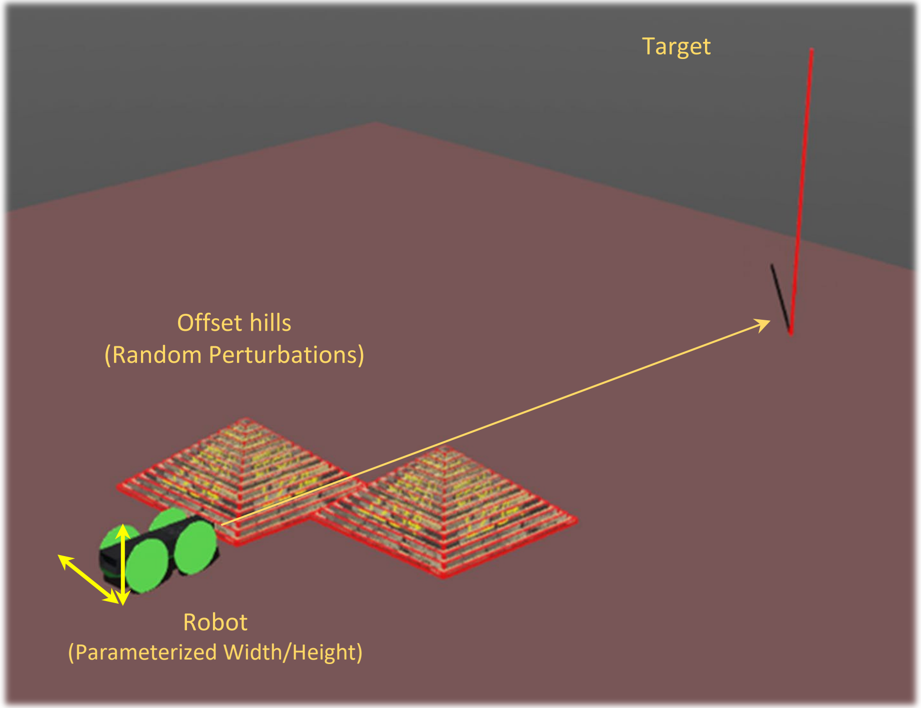


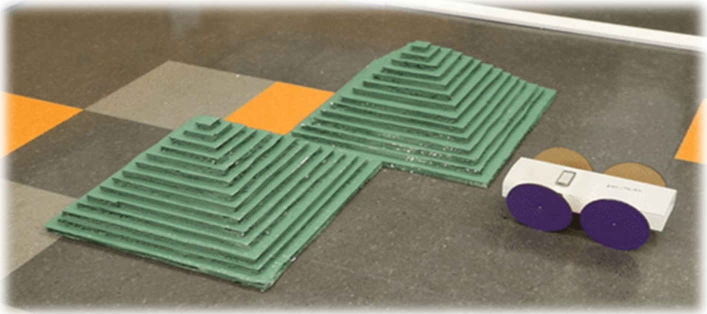
Multi-agent,
Reactive,
Stochastic Scenarios
Generated using Scenic

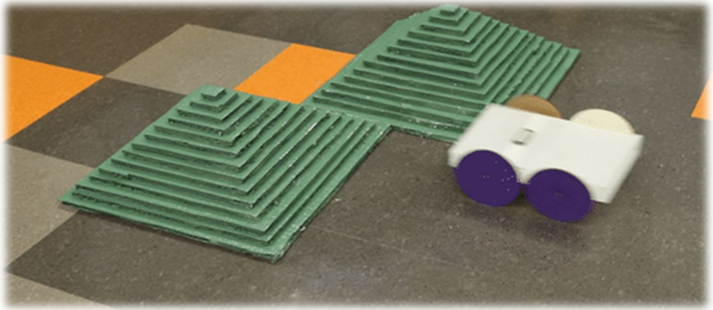
To train, test, or data generation for offline training

A. Salam, E. Kim, K. Lee, Q. Wu, I. Stoica, P. Abbeel, S. Seshia, “Programmatic Modeling and Generation of Real-time Strategic Soccer Environments for Reinforcement Learning,” AAAI 2022

Design Space Exploration and Evaluation (Sim2Real)







Additional Scenic features: Compositional Scenarios

Scenic allows for scenarios to be defined modularly using parallel, sequential and more complex forms of compositions

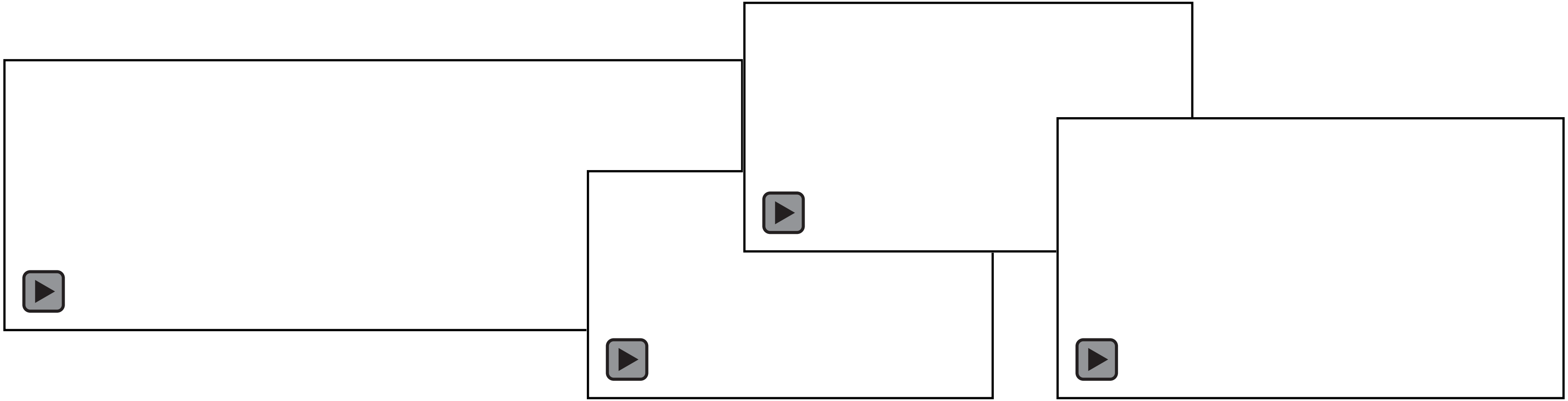
```
import StopAndStart, BadlyParkedCar

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


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

Additional Scenic features

- Easy to interface to other simulators
- Constructs for easily recording data from simulations (sensors, ground truth, etc.)
- Declaratively imposing requirements on scenes and simulations
- Saving and replaying scenes and simulations
- Integration with *VerifAI* toolkit for falsification, optimization, etc. https://verifai.readthedocs.io/

Scenic: Probabilistic Scenario Description Language

- **A probabilistic programming language** for modeling multi-agent, reactive, stochastic scenarios
- Many applications, including testing & verification, synthetic data generation, design space exploration, debugging & triage, …
- Interfaced with various simulators and can be interfaced to any simulator of your choice:



- Scenic Website: scenic-lang.org
- Scenic Forum: https://forum.scenic-lang.org/
- Github Repo: https://github.com/BerkeleyLearnVerify/Scenic/

Scenic Team members

We thank the many people who have contributed to Scenic over the years:

Johnathan Chiu,

Tommaso Dreossi,

Shromona Ghosh,

Francis Indaheng,

Sebastian Junges,

Kevin Li,

Yash Pant,

Hadi Ravanbakhsh,

Steve Lemke,

Shalin Mehta

Jay Shenoy,

Kesav Viswanadha,

Xiangyu Yue,

Kurt Keutzer,

Alberto Sangiovanni-Vincentelli,

Pravin Varaiya,

Alex Kurzhanskiy,

Ellan Kalvan,

Qiang Lu,

Sridhar Duggirala,

Shun Kashiwa,

Matthew Rhea,

Necmiye Ozay,

Eric Vin,

Dragos Margineatu,

Denis Osipychev,

Atul Acharya,

Xantha Bruso,

Paul Wells,

and several others…

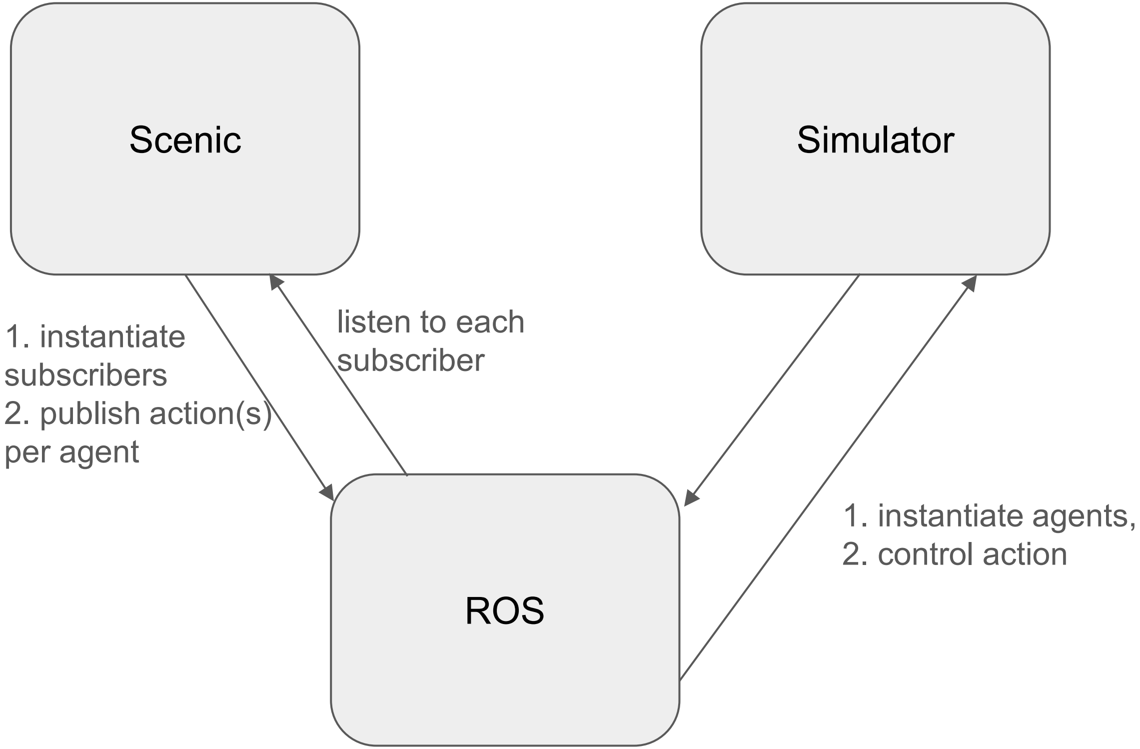
Thank you!

Please provide your feedback in this QR Code / Link

- Which domains of robotics is Scenic useful for?
- Which simulator should we interface Scenic to?

https://shorturl.at/1bHcU

Scenic Interface to Simulator with ROS



def setup():
- instantiate ROS nodes for all Scenic agents

def getProperties():
- listen each Scenic agent's ROS node to update properties

def step():
- publish to each Scenic agent's ROS node to execute action(s)

Testing and Falsification

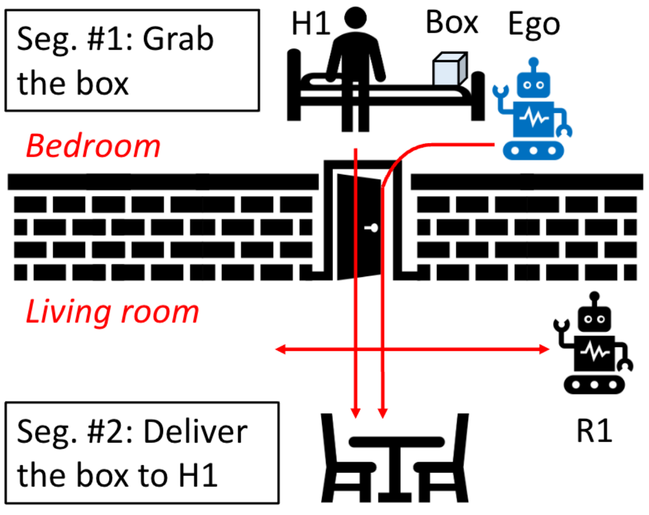
Multi-agent scenario involving

- 2 Robots: Spot, Fetch
- Human

Simulator: Meta Habitat

Habatat Example

```
behavior GrabAndDeliver():
    do SpotPickUp(box)
    try:
        while True:
            wait

    interrupt when human._in_position:
        do GoToHuman()
        do SpotDeliver()
        terminate

behavior ReachHandAndWalk(walk_position, reach_position):
    try:
        do HumanReach(reach_position)
        while True:
            wait
    interrupt when spot._holding_object:
        do HumanGoTo(walk_position)
        while True:
            wait

behavior Traverse():
    while True:
        do GoForward(1.0)
        do TurnAround()
```




```
humanoid = new Female_0 at (human_spawn_x, -4.8, 0),
            with yaw -90 deg,
            with behavior ReachHandAndWalk(human_dest,
                                            human_reach_rel_pose)

box = new GelatinBox on (box_spawn_x, box_spawn_y, bed_height)
spot = new SpotRobot at spot_pos, with behavior GrabAndDeliver()
fetch = new FetchRobot at (-3.7, fetch_spawn_y 0),
            with yaw 90 deg, with behavior Traverse()
```

Habitat Scenario Overview

1. ego (Spot), will pick up a box spawned on the bed.
2. Meanwhile, a human (controlled by Scenic), will reach her hand around the area of the box.
3. ego has to pick up the box, without getting close to hitting the human. (end of segment 1)
4. Once ego finishes picking up the box, the human will walk out into the living room. ego has to deliver the box to the human.
5. At the same time, a second robot is walking back and forth outside the living room. Spot should deliver the box while avoiding hitting the other robot.
6. The video shows some instances where the robot wouldn’t walk out to the living room. That is because its navigation unit is unable to path plan.

Advanced Scenic Constructs

Interrupts allow adding special cases to behaviors without modifying their code

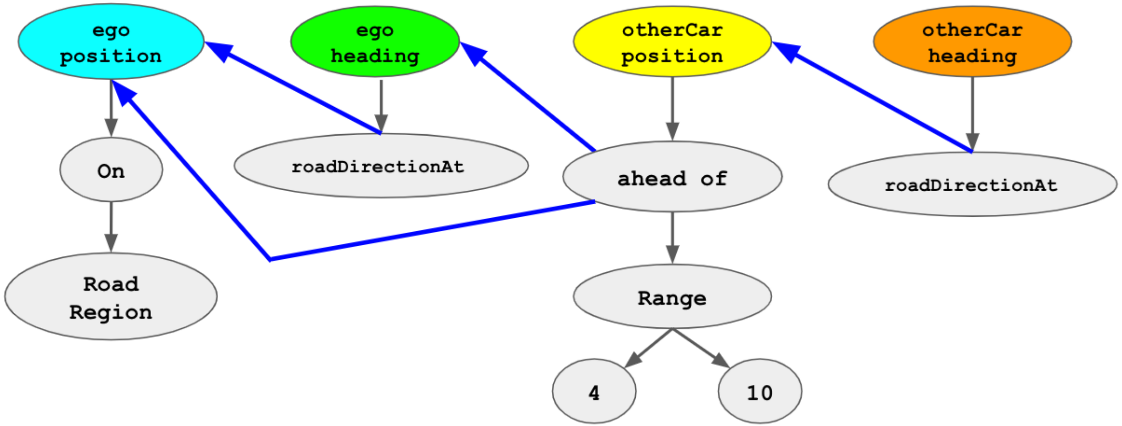
```
behavior FollowLeadCar(safety_distance=10):
    try:
        do FollowLaneBehavior(target_speed=25)
    interrupt when (distance to other) < safety_distance:
        do CollisionAvoidance()
```

Temporal requirements and monitors allow for enforcing constraints during simulations

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

Sampling Concrete Scenarios from a Scenic Program

```
param weather = Uniform('sunny','rainy')
param time = Range(10, 12) # pm

ego = Car on road, facing roadDirection
otherCar = Car ahead of ego by Range(4,10) #meters
require not (otherCar in intersection)
```



Scenic samples while abiding by the specified dependencies in the program