Assurance-based Learning-enabled Cyber-Physical Systems

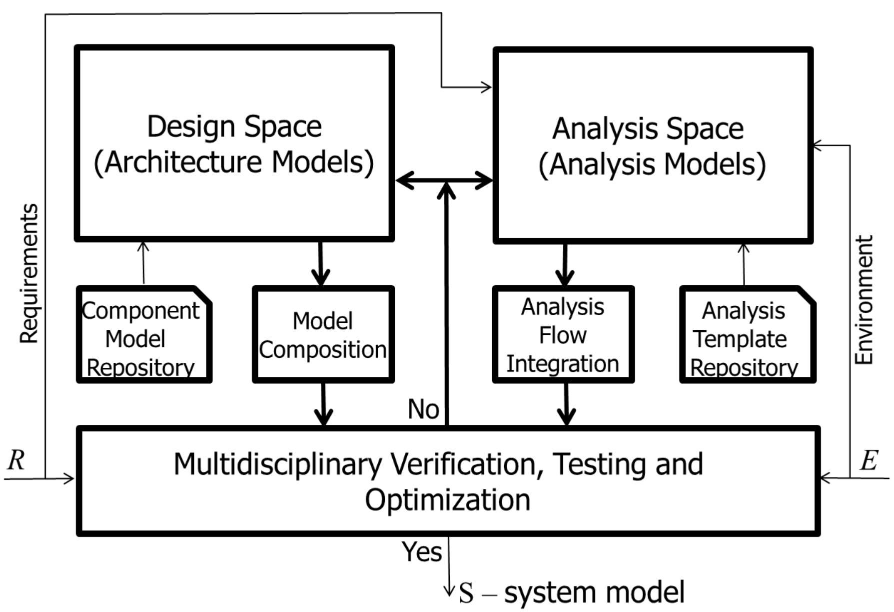
Gabor Karsai, Xenofon Koutsoukos, Taylor Johnson, Ted Bapty, Abhishek Dubey, Nag Mahadevan, and many others

Supported by DARPA AA/AFRL

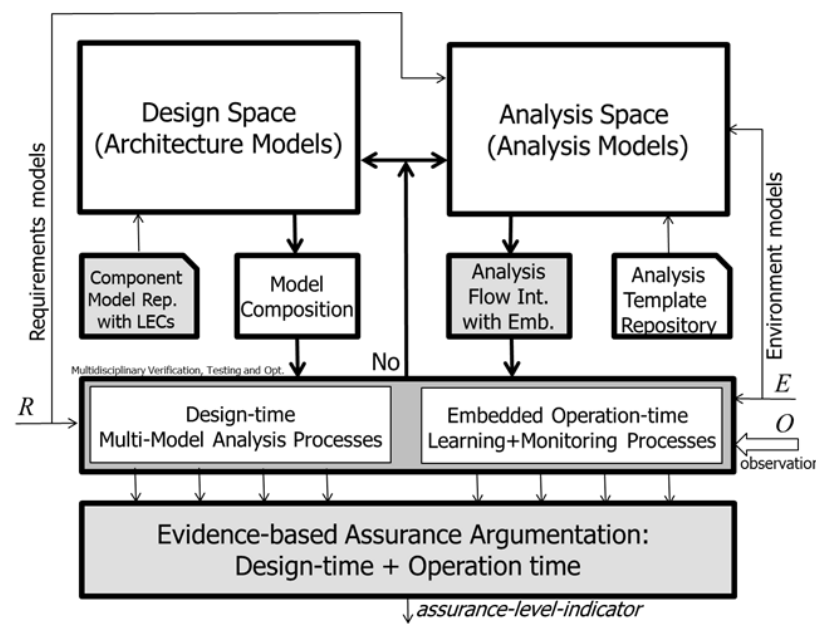
Project challenge
AI/ML in Cyber-Physical Systems

"Our vision is to … create a new design flow that extends from design-time to operation time, re-interprets the traditional assurance argumentation to become a dynamic, operational concept. Our ultimate goal is to establish a fusion of model- and component-based methods with data-driven methods."

Model-driven design flow

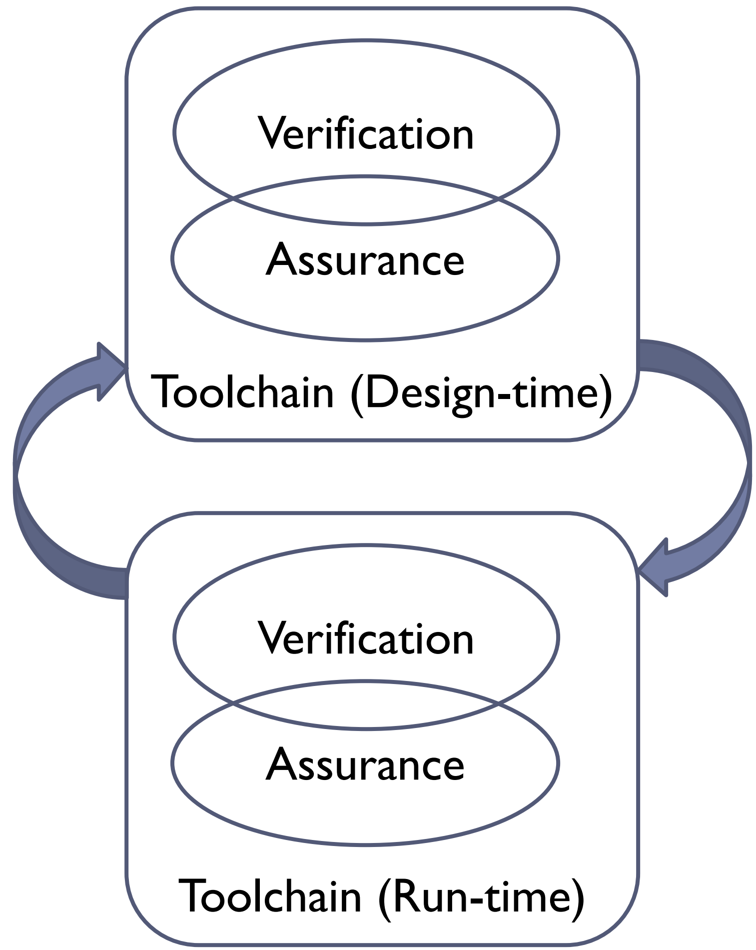


Model-driven design flow with LEC-s



Project activities

- Thrusts:
  - Verification: formal and/or coverage-driven verification of safety / robustness properties of components, subsystems, and systems, at design-time and at run-time, to provide evidence for assurance arguments
  - Assurance: construction and continuous monitoring of logical arguments that demonstrate the *truth* or *strength* of a safety claim based on available evidence
  - Toolchain: design-time and run-time software tools to implement and support the above, for real systems



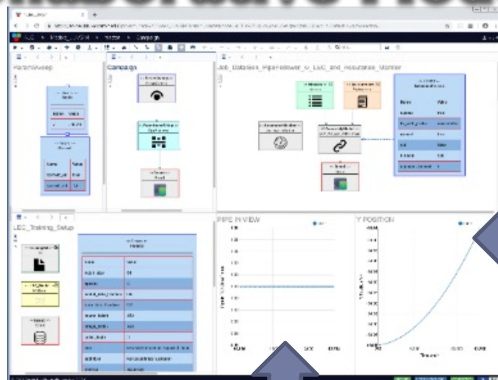
ALC Toolchain Approach

ALC Workflows

Collaborative Modeling

System Integrator

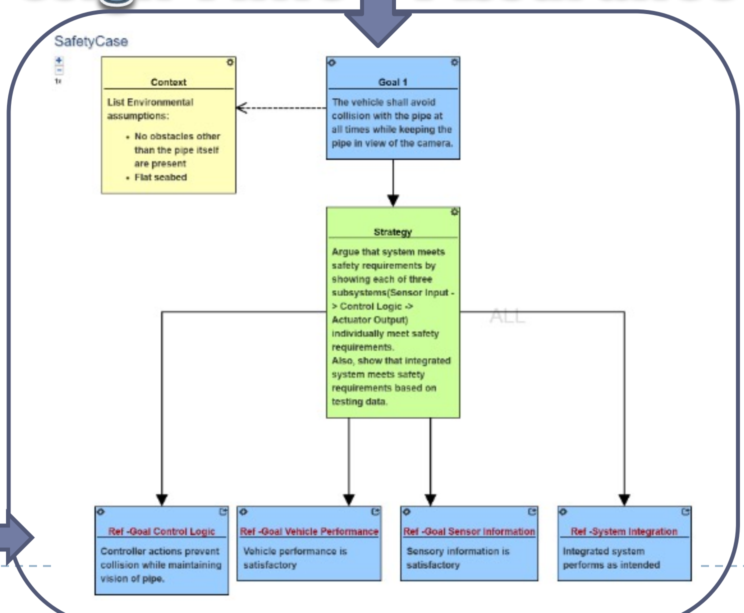
LEC Developer



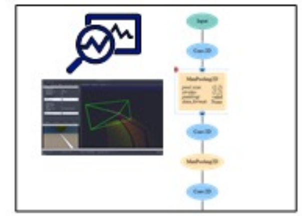
Design Time Assurance

- *The model driven toolchain supports training, verification and design-time assurance of learning enabled components.*
- *Toolchain helps with developing safety assurance cases for the system using collected evidence.*
- *Complete provenance tracking of experimental runs and data collection is supported.*

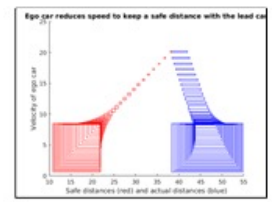
Assurance Engineer



Execution, Training, Data Collection, Verification



(d) Run System with LEC



(c) Verify Closed-Loop System With LEC

(b) Train LECs

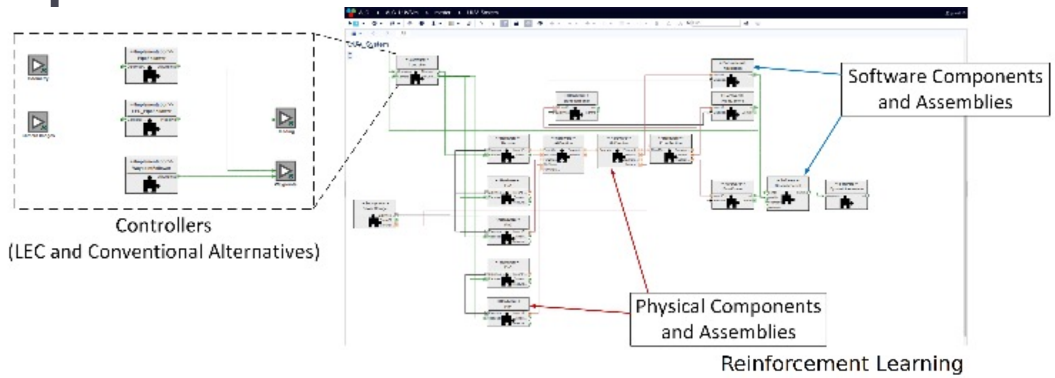
(a) Run Scenarios To Collect Data

Typical Workflow Sequence

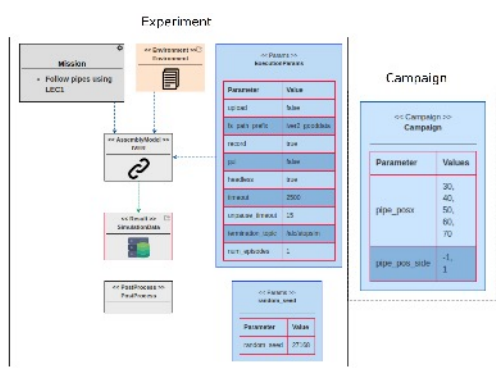
ALC Toolchain

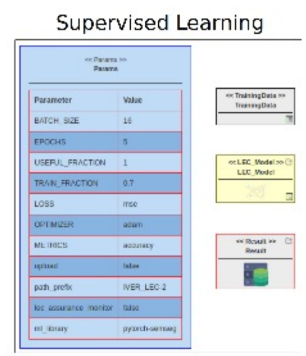
ALC Toolchain Concepts

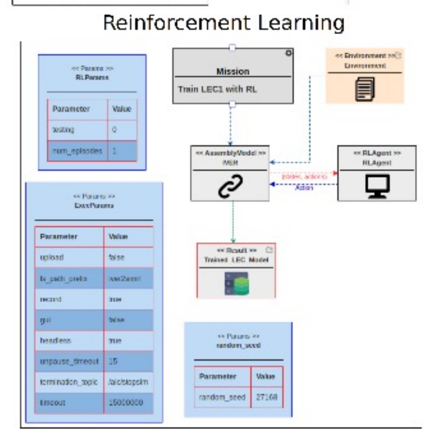
- Modeling
  - System Architecture / SysML



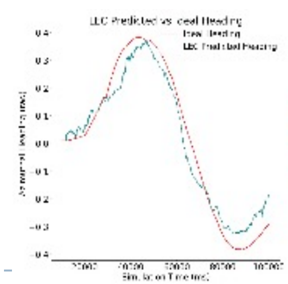
- LEC Construction
  - Data collection
  - Training
  - Evaluation

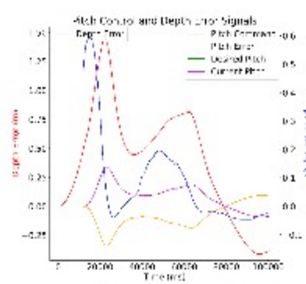


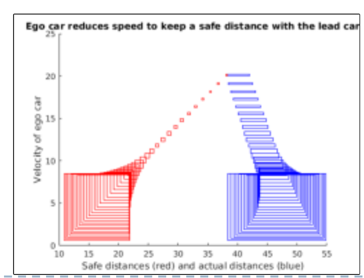


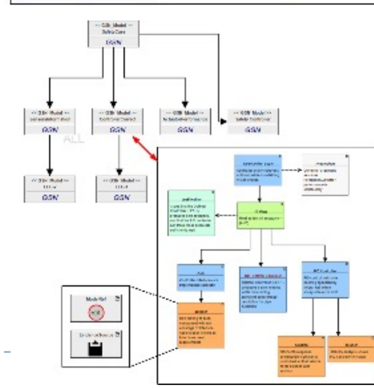


- Testing -- Verification/Validation/Assurance



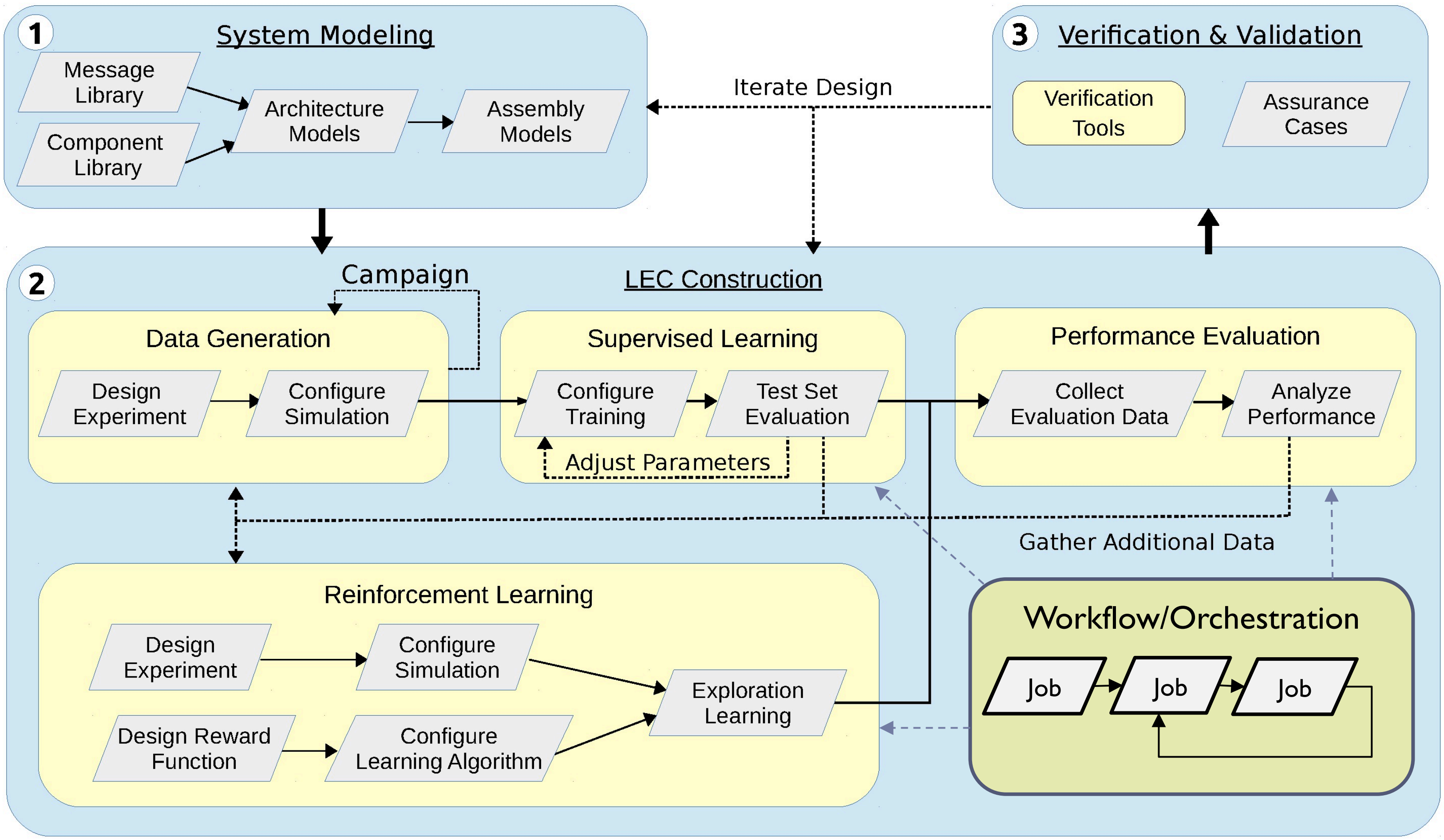




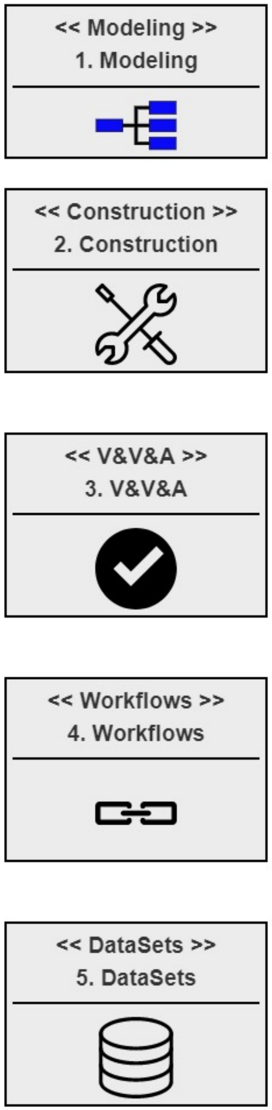


ALC Design Workflow

- Specialized for LEC development



Modeling Blocks, Systems, Training, & Execution



Model Systems
- Block Library
- Messages/Datatypes for Software
- System Structure

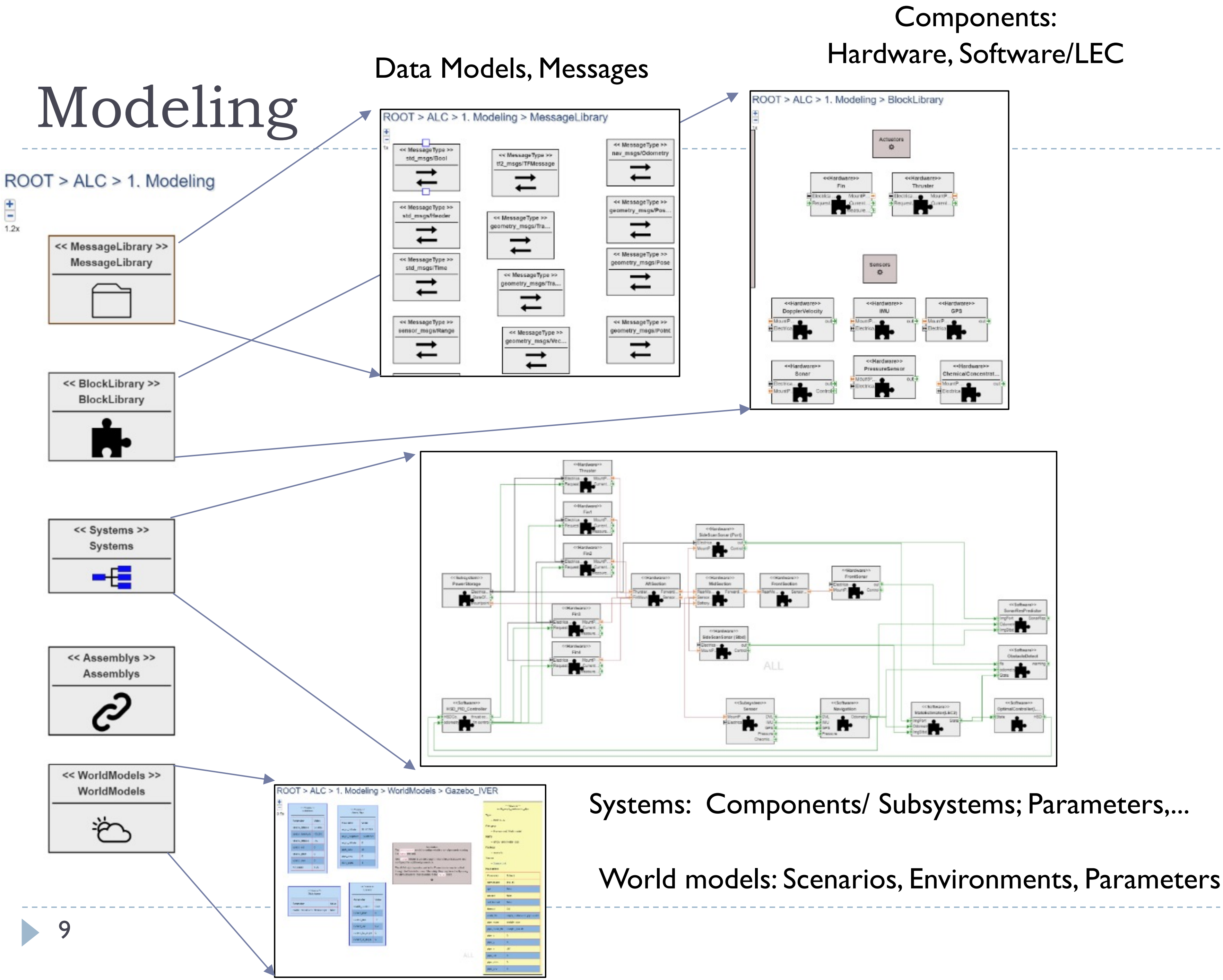
Construct Experiments
- Data Collection
- LEC Training
- Assurance

Verification, Validation, and Assurance
- Formal System Verification
- LEC Validation
- Assurance Argument Modeling

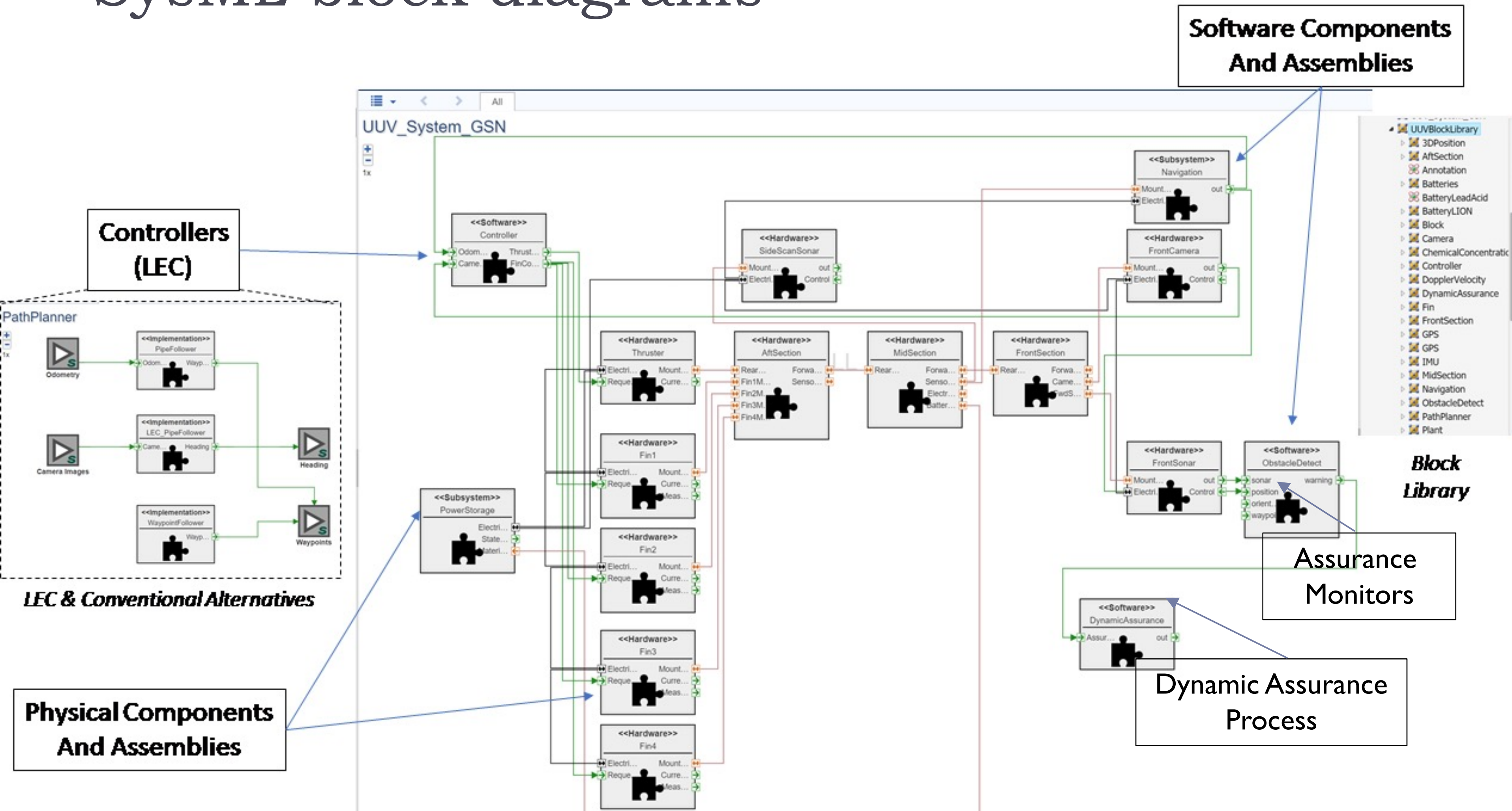
Workflows:
- Create/Execute Sequences of Operations

Data Sets:
- Maintain Data Created via Construction Workflows
- Track Data Provenance
- Launch analysis of data

Modeling


Data Models, Messages
Components:
Hardware, Software/LEC
ROOT > ALC > 1. Modeling
<< MessageLibrary >>
MessageLibrary
<< BlockLibrary >>
BlockLibrary
<< Systems >>
Systems
<< Assemblys >>
Assemblys
<< WorldModels >>
WorldModels
ROOT > ALC > 1. Modeling > MessageLibrary
ROOT > ALC > 1. Modeling > BlockLibrary
ROOT > ALC > 1. Modeling > WorldModels > Gazebo_IVER
Systems: Components/ Subsystems; Parameters,...
World models: Scenarios, Environments, Parameters

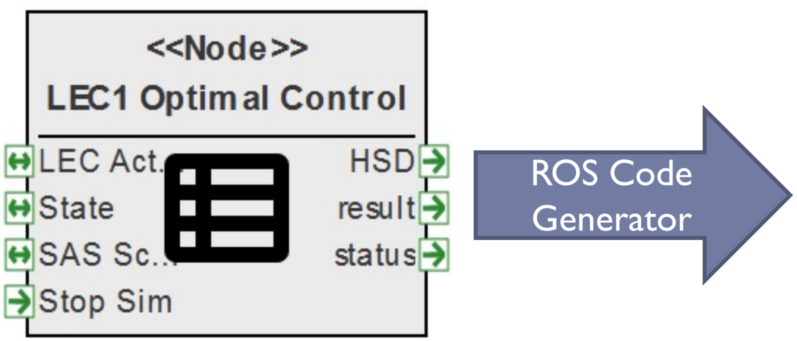
System architecture models: SysML block diagrams



Subset of SysML Blocks, IBD to model all blocks, implementation alternatives for flexibility

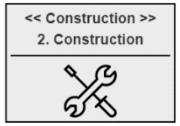
Code generation: ROS Skeleton Code

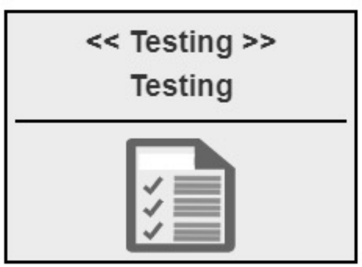
- Generate implementation source code (skeleton) and launch files for the components from architectural models
  - Preserve custom code ('business logic') when re-generating
  - Boilerplate code for interfacing with LECs
  - Launch files generated for individual components and composed system
- Automatically deploy & build ROS Packages
  - ROS source code and launch files

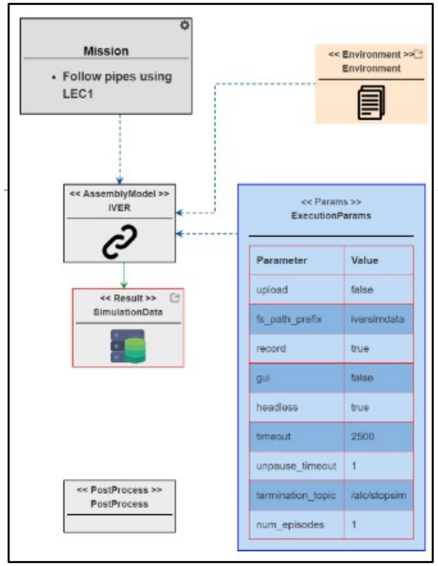



```python
class LEC1_Optimal_ControlImplementation(object):
    """
    Class to contain Developer implementation.
    """
    def __init__(self):
        """
        Definition and initialization of class attributes
        """
        #parameters
        self.agent_json = rospy.get_param("~agent_json","$(arg agent_json)")
        self.fls_clustering_neighborhood = rospy.get_param("~fls_clustering_neighborhood","$(
        self.lat_ref = rospy.get_param("~lat_ref","$(arg vehicle_latitude)")
        self.lon_ref = rospy.get_param("~lon_ref","$(arg vehicle_longitude)")
        self.min_fls_samples = rospy.get_param("~min_fls_samples","$(arg min_fls_samples)")
        self.models_dir = rospy.get_param("~models_dir","$(arg rl_model_dir)")
        self.network_json = rospy.get_param("~network_json","$(arg network_json)")
        self.num_agents = rospy.get_param("~num_agents","$(arg num_agents)")
        self.testing = rospy.get_param("~testing","$(arg testing)")
        self.use_lec2 = rospy.get_param("~use_lec2","$(arg use_lec2)")
        self.weights_dir = rospy.get_param("~weights_dir","$(arg weights_dir)")
        self.world_lat = rospy.get_param("~world_lat","$(arg origin_latitude)")
        self.world_lon = rospy.get_param("~world_lon","$(arg origin_longitude)")
        self.deployment_folder = rospy.get_param("~rl_model_dir")
        self.network_interface = None
        self.assurance_monitor_paths=[];
        self.ams = ''
```

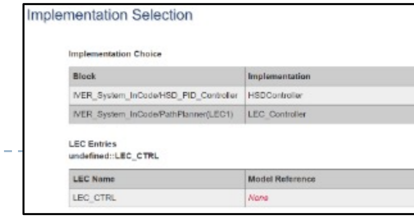
LEC Construction



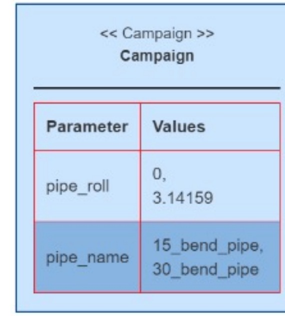


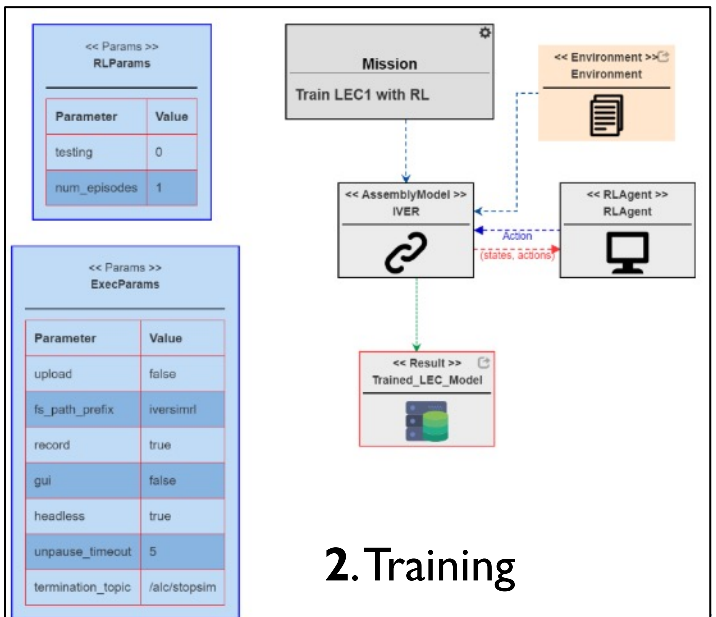


1. Data Collection

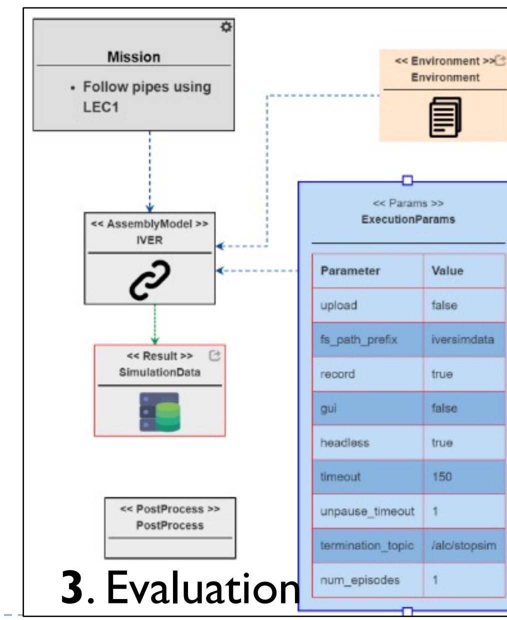


Select Configuration



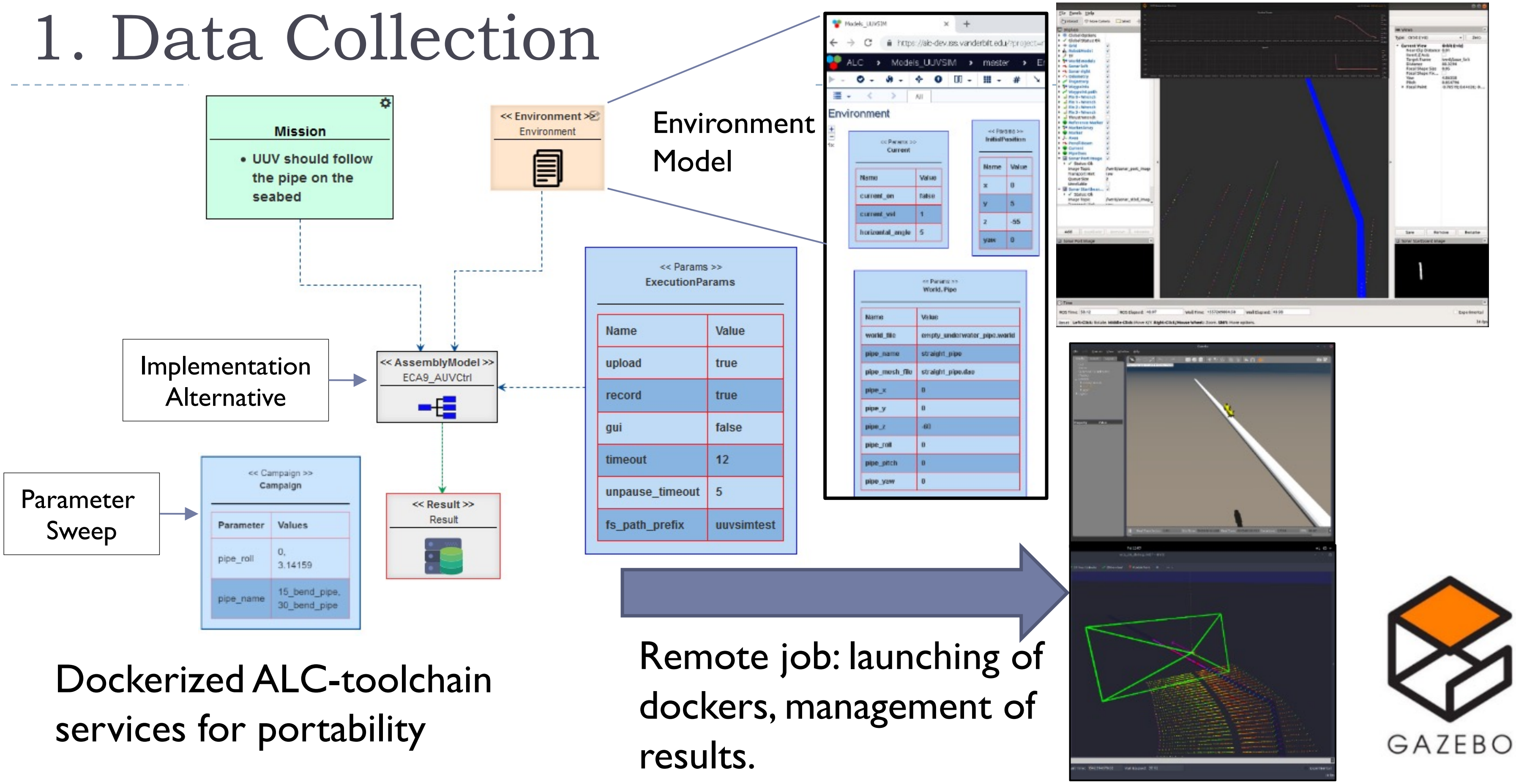


2. Training



3. Evaluation

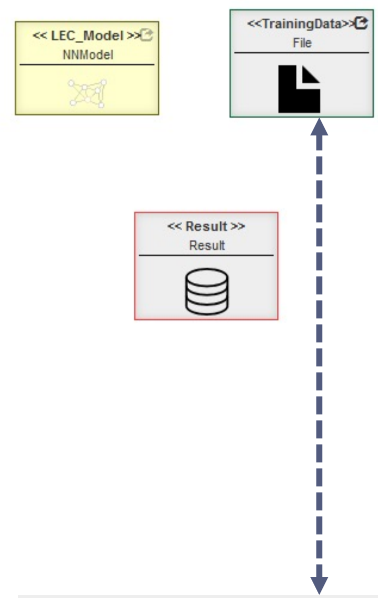
LEC Construction: 1. Data Collection



- *Assembly model* selects a specific implementation variant of a system architecture
- *Mission*, *Environment*, and *Execution parameters* set up the experiment scenario
- *Campaigns* across parameters a configurations related to system and environments
- Tool generates configuration file for running the simulation, captures results + meta data for all trials

LEC Construction: 2. Training

- Neural Net model and parameters specified in "LEC Model"
- "Training Data" links to data generated from previous experiments
- Training job is dispatched to worker machines (typically with GPUs)
- Results and metadata are saved from the training sessions

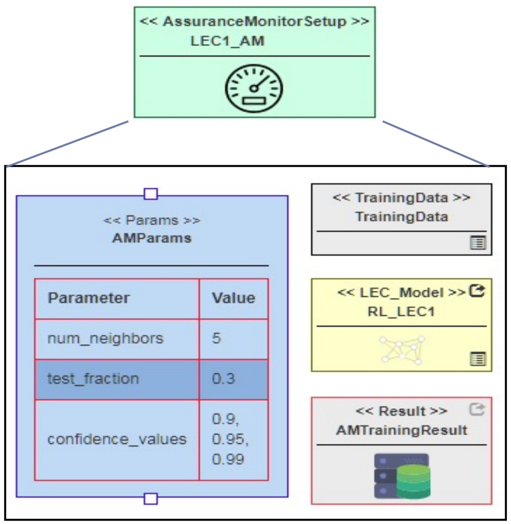


<< Params >>
Params

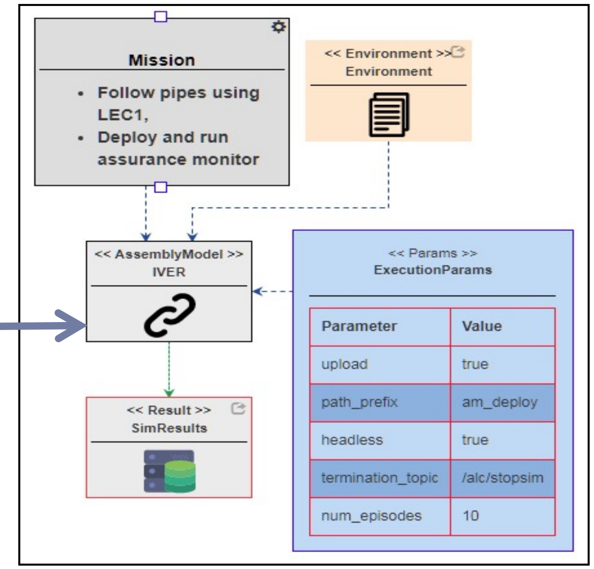
Name	Value
batch_size	64
epochs	5
useful_data_fraction	0.5
train_data_fraction	0.7
image_height	492
image_width	768
color_depth	3
loss	keras.losses.mean_squared_error
optimizer	keras.optimizers.Adam()
metrics	accuracy
upload	true

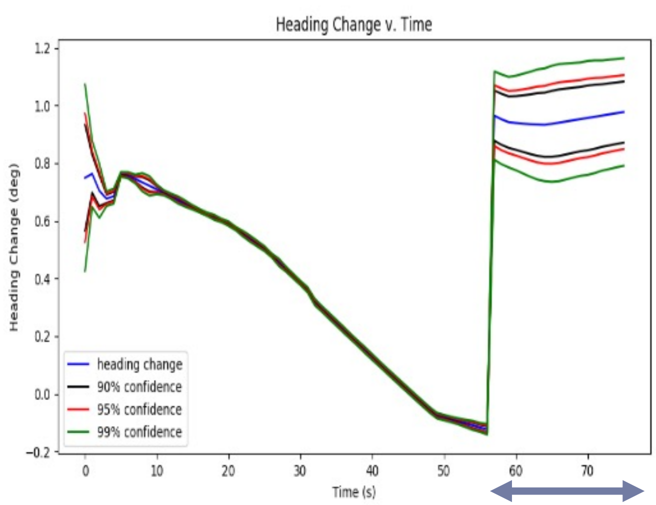
Name	Type
result-DataGen_PipeFollower-1542125401110	*metadata.json*
result-DataGen_PipeFollower-1542129035690	*metadata.json*
result-DataGen_PipeFollower-1542140169160	*metadata.json*

LEC Construction
2. Training: Assurance Monitor

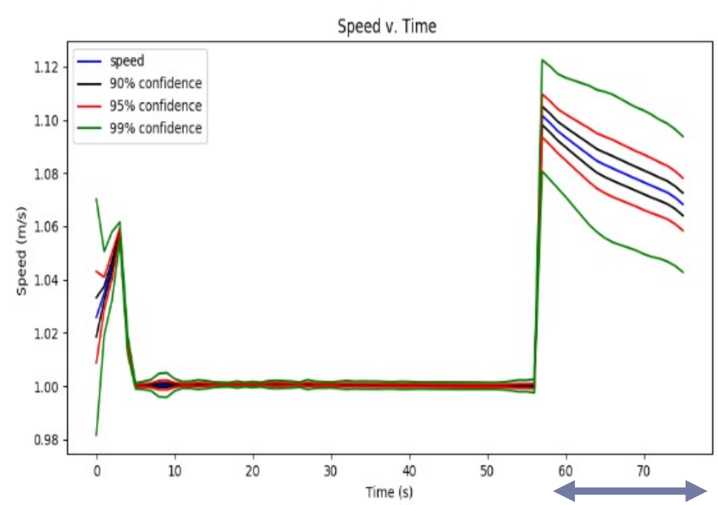


Deploy trained LEC, build assurance monitor.





Pipe Visibility Lost

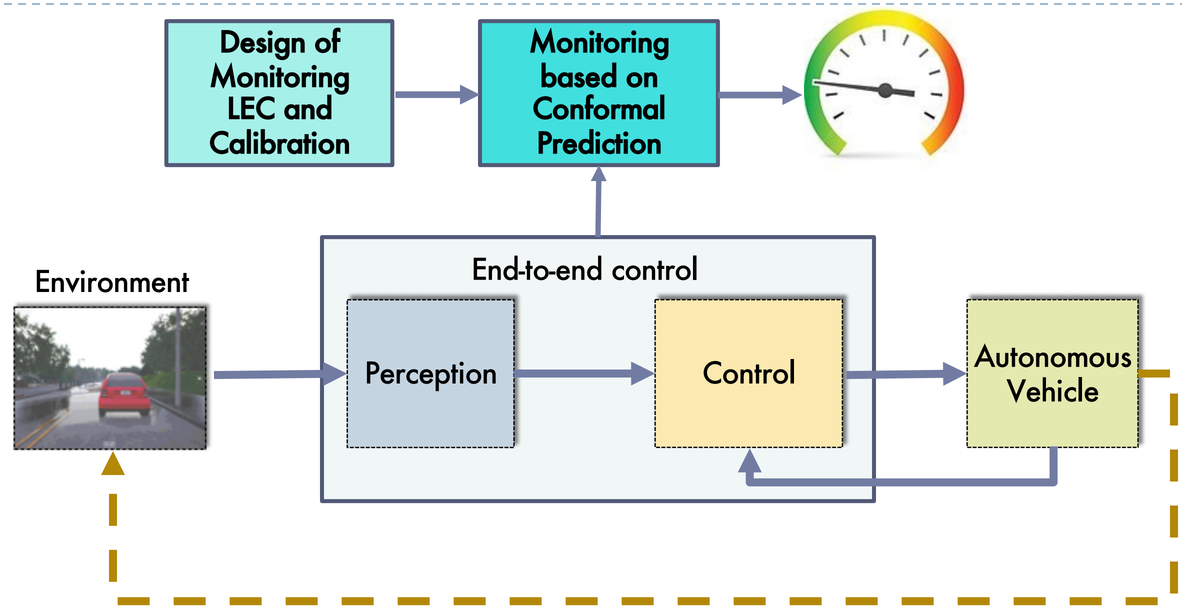


Pipe Visibility Lost

Assurance Monitors

Xenofon Koutsoukos and team

Assurance Monitoring in Learning-Enabled CPS



Assurance monitoring based on inductive conformal anomaly detection

- Variational autoencoder (VAE)
- VAE for regression/classification
- Adversarial Autoencoder (AAE)
- Deep support vector description (SVDD)

- Evaluation
  - Self-driving simulator (and open datasets)
  - Autonomous underwater vehicle

Real-Time Detection of Dataset Shifts

- LECs may compromise system safety when their predictions may have large errors
  - When the runtime data are different than the data used for training.
- Approach based on *inductive conformal prediction* and *anomaly detection*
  - *Neural network architectures* to compute efficiently the nonconformity of new inputs relative to the training data.
  - Multiple examples sampled from *generative models* to improve robustness of detection: Variational Autoencoder (VAE)
  - *Saliency maps* that identify parts of the input that contribute most to the LEC predictions improve robustness.
- Evaluation results
  - Small detection delay
  - Small number of false alarms
  - Execution time comparable to the execution time of the original LECs.

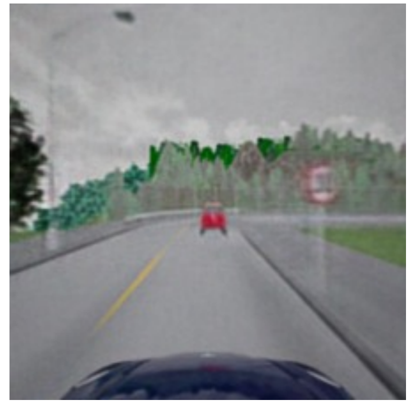
Novelty Detection in High-Dimensional Time-series

- In autonomous systems, inputs are high-dimensional sensor measurements (e.g., camera, LiDAR) and arrive one by one based on the sampling rate of the sensors
- After observing each input, inductive conformal anomaly detection is used to quantify the degree to which the input disagrees with the training data
- Main idea: Train an appropriate neural network architecture which can be used in real-time for assurance monitoring
  - Generate multiple examples sampled from a learn representation from the training distribution
  - Compute a nonconformity measure (NCM) to evaluate the degree to which a new example disagrees with the distribution of training data
  - Compute empirical *p*-values used for statistical significance testing
  - Perform a randomness test to based on the *p*-values to evaluate if the generated examples are from the distribution of training data
  - Compute an assurance measure based on the randomness test

VAE-Based Nonconformity Measure

Original Image

Reconstructed Image

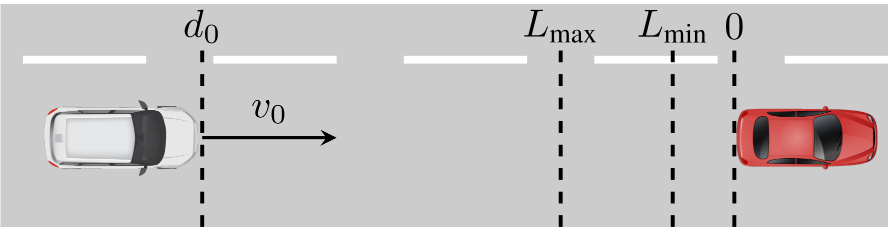


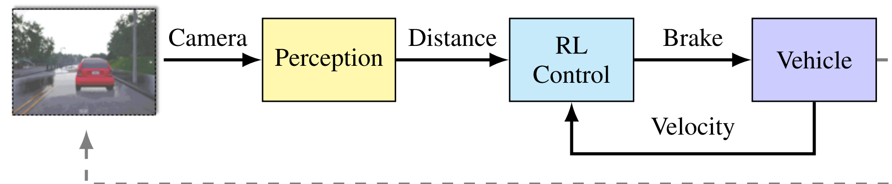
Nonconformity measure

$$\boxed{\alpha'_k = A_{\text{VAE}}(z_t, z'_k) = ||z_t - z'_k||^2}$$

- Given an input example at time *t*, the encoder portion of the VAE is used to approximate the posterior distribution of the latent space
  - Typically, the posterior of the latent space is approximated by a Gaussian distribution
- Sampling from the posterior generates multiple encodings so that the decoder is exposed to a range of variations of the input example
  - An in-distribution input should be reconstructed with a relatively small reconstruction error.
  - Conversely, an out-of-distribution input will likely have a larger error.
- The reconstruction error is a good measure of the strangeness of the input relative to the training set and it is used as the nonconformity measure

Advanced Emergency Braking System (AEBS)





Data Generation using CARLA simulator

d_0	100 m approximately
v_0	Randomly sampled between 90 and 100 km/h
L_{min}	1 m
L_{max}	3 m
CARLA precipitation parameter r	Randomly sampled between 0 and 20
Sampling period	1/20 sec = 50 ms

- Learning-Enabled Components
  - Perception: CNN with 11 layers
  - Control: Reinforcement learning controller trained using DDPG
  - VAE: CNN encoder with 4 layers, 1024 FC layer, and symmetric decoder
  - SVDD: 4 convolution layers and 1568 FC layer

False alarms and average delay

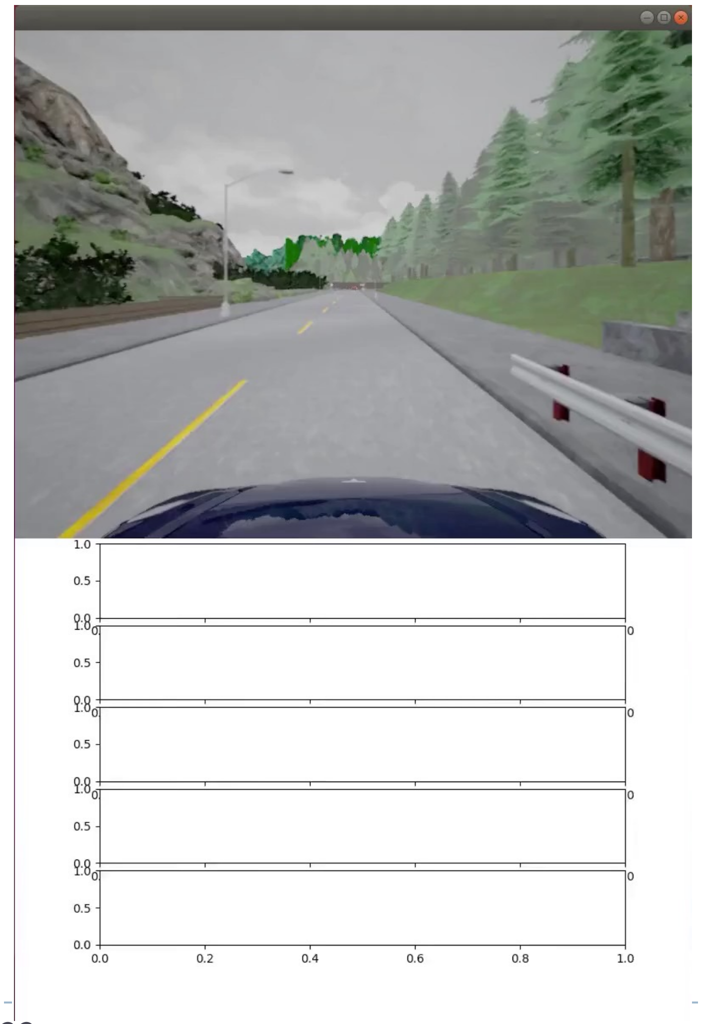
False positive	False negative	Average delay (frames)
2/108	0/92	17.91

Execution Times

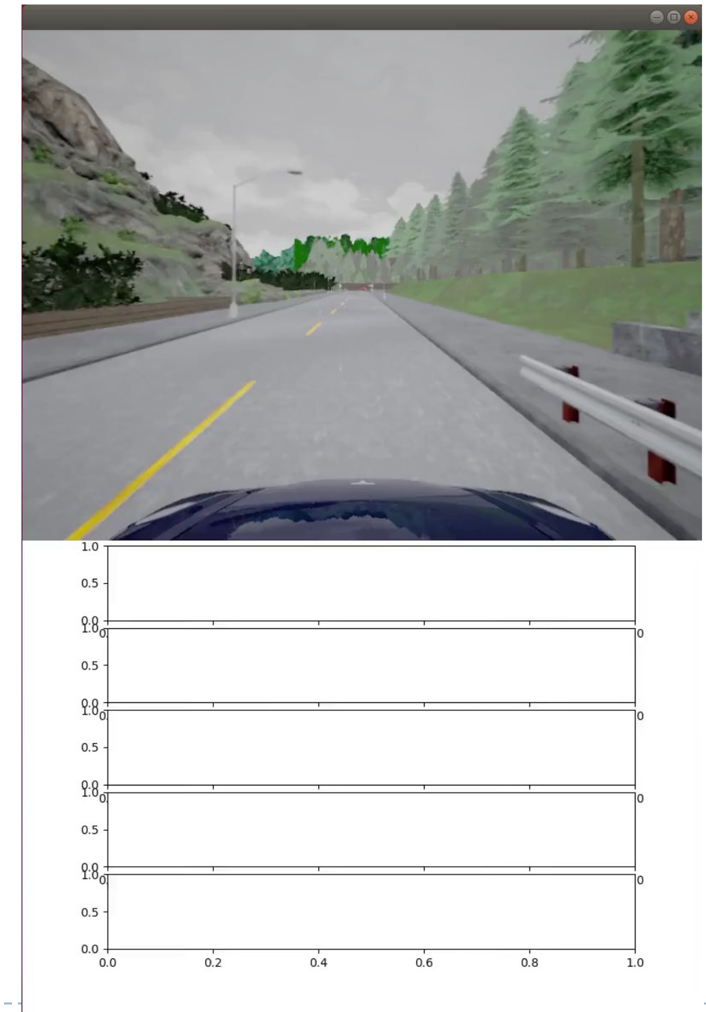
min (ms)	Q_1 (ms)	Q_2 (ms)	Q_3 (ms)	max (ms)
34.61	34.75	34.78	34.82	35.10

Simulation Results
Distribution Shift due to Weather

In-distribution

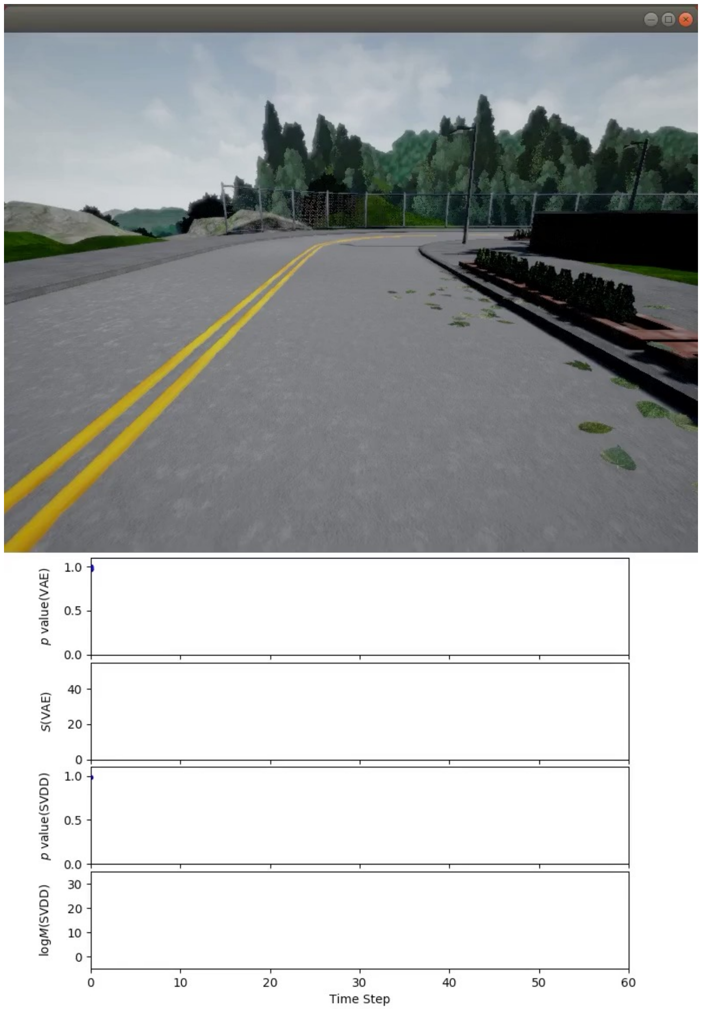


Out-of-distribution

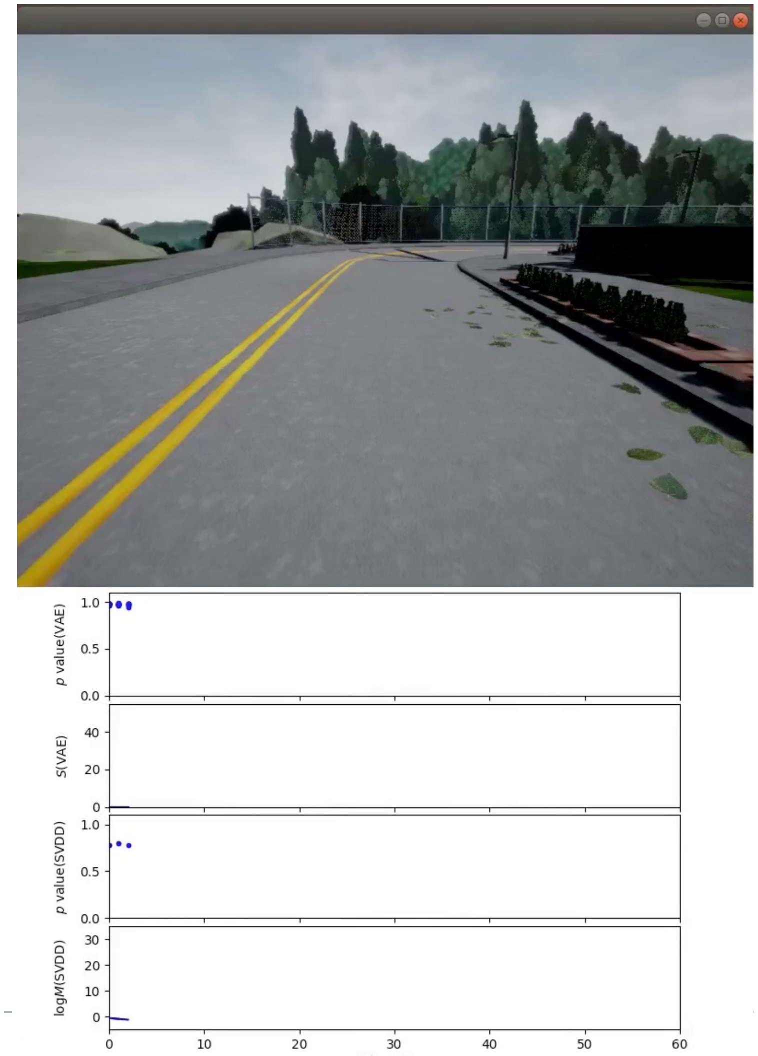


Simulation results
Adversarial input

No attack



Attack



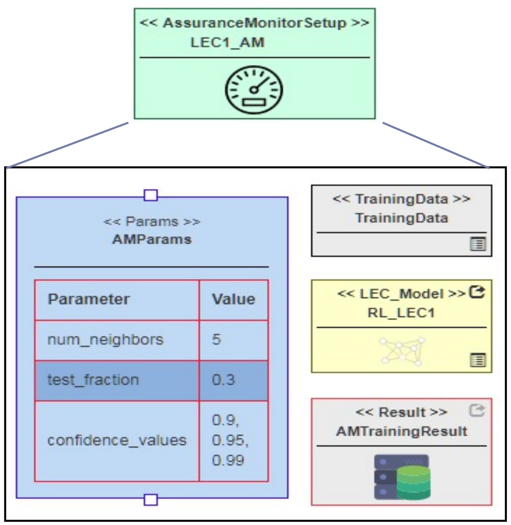
Highlights

- Train LECs that allow effective *assurance monitoring* based on deep learning and statistical significance testing
- Integration into a toolchain for model-based design of cyber-physical systems with learning-enabled components
- Evaluation with simulators
  - Small number of false positives and detection delay
  - Execution time is comparable to the execution time of the original LECs

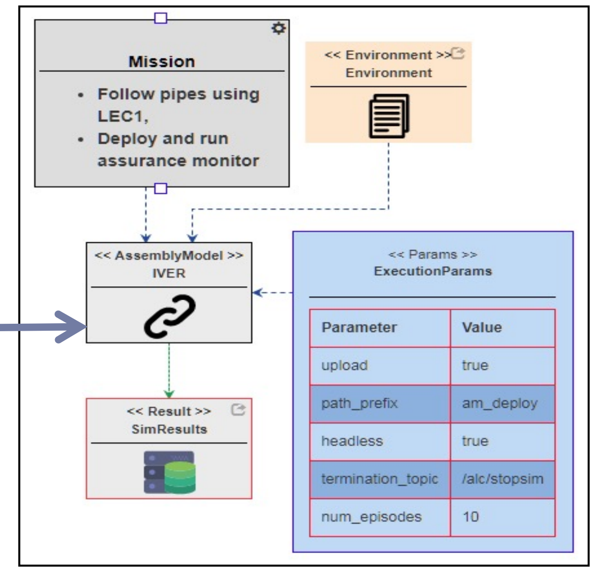
ALC Toolchain

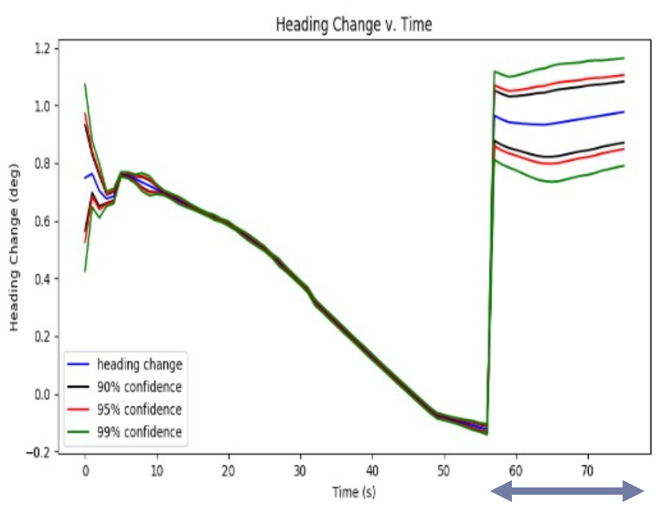
Continued

LEC Construction
2. Training: Assurance Monitor

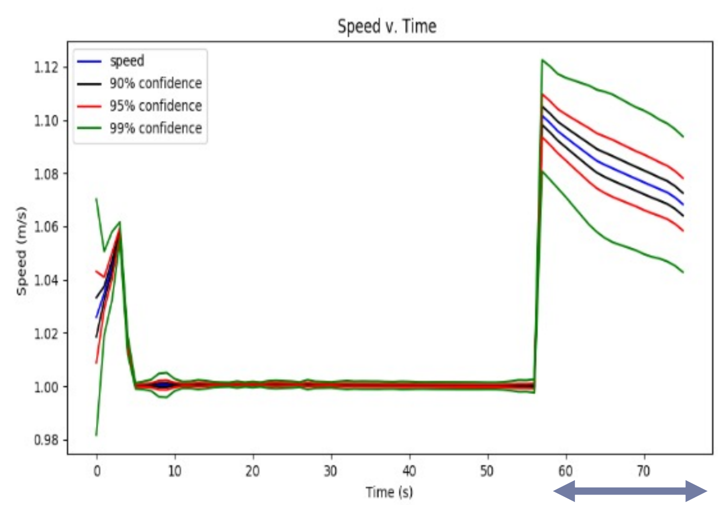


Deploy trained LEC, build assurance monitor.



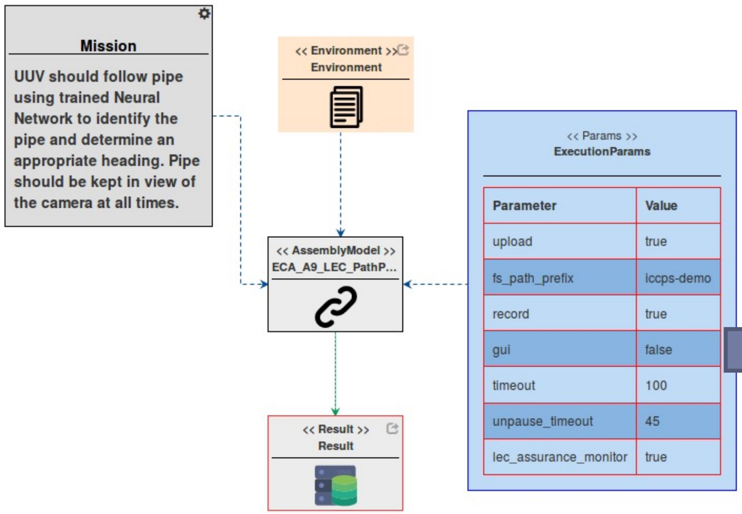


Pipe Visibility Lost



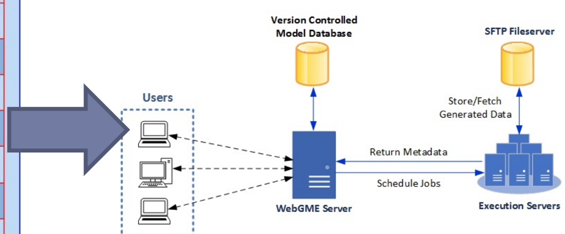
Pipe Visibility Lost

LEC Construction:
3. Evaluation: Testing/Verification

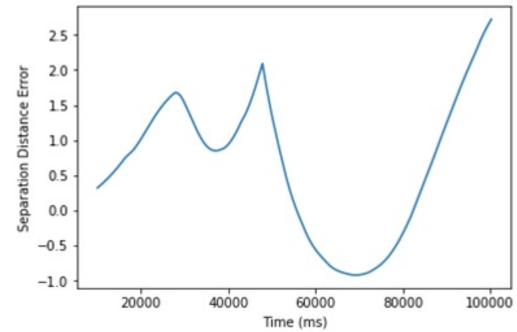


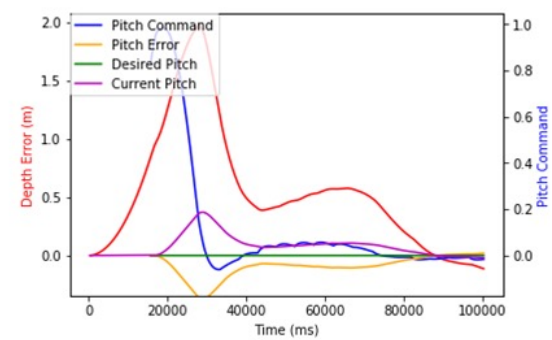
Analysis in Jupyter Notebook

Also, "single step" the process for debugging



Execute on Remote Server/s





Training Model Data Managed on GitLab

- Trained Neural Net can be tested in the simulator with another experiment model
- Performance metrics are recorded for LEC evaluation, e.g.:
  - Distance from ideal path
  - Pipe within camera field of view

Name	Type	Size	Creation Date
result-NN_Training_Test-1542127634867	*model.keras*	267 B	11/13/2018
result-NN_Training_Test-1542128784700	*model.keras*	267 B	11/13/2018

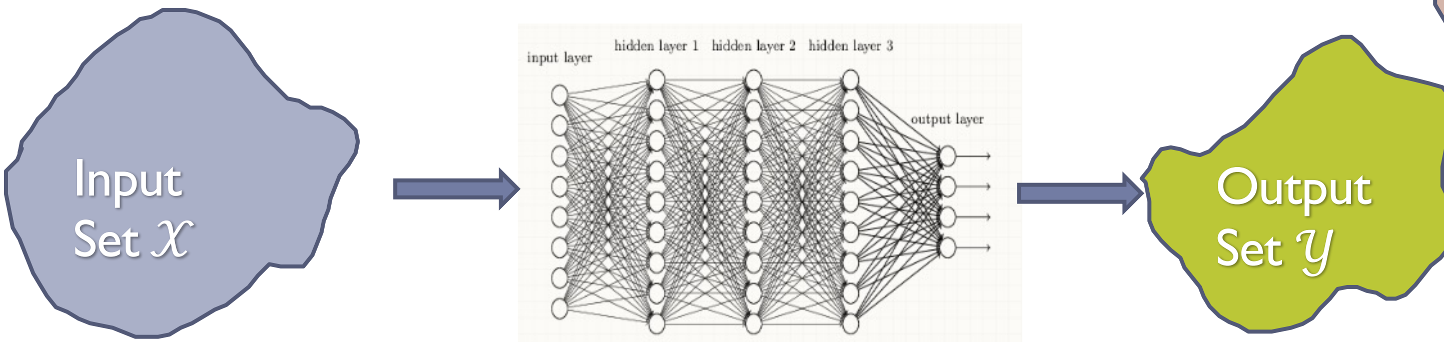
Results in file store + git, cross-linked for data provenance

Verification of Deep Neural Networks

Taylor Johnson and team

LEC Verification: Reachability Analysis of Feedforward/Convolutional Neural Networks

- Given a NN F & an input set $\mathcal{X}$, the **output reachable set** of F is
$$\mathcal{Y} = \{y \mid y = F(x), x \in \mathcal{X}\}$$

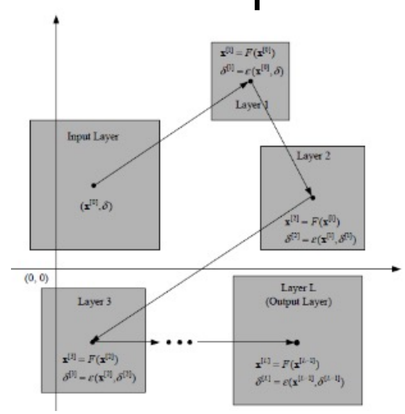


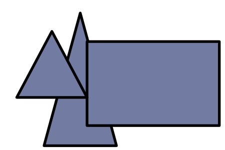
- Computationally: Given a NN F, a convex initial set of inputs I represented as a polytope $\mathrm{poly}(\mathcal{X})$, compute the output set Y = F(I) of the network

$\mathrm{I{=}poly}(\mathcal{X})$

$$I = \{x \mid Ax \leq B, x \in R^n\}$$

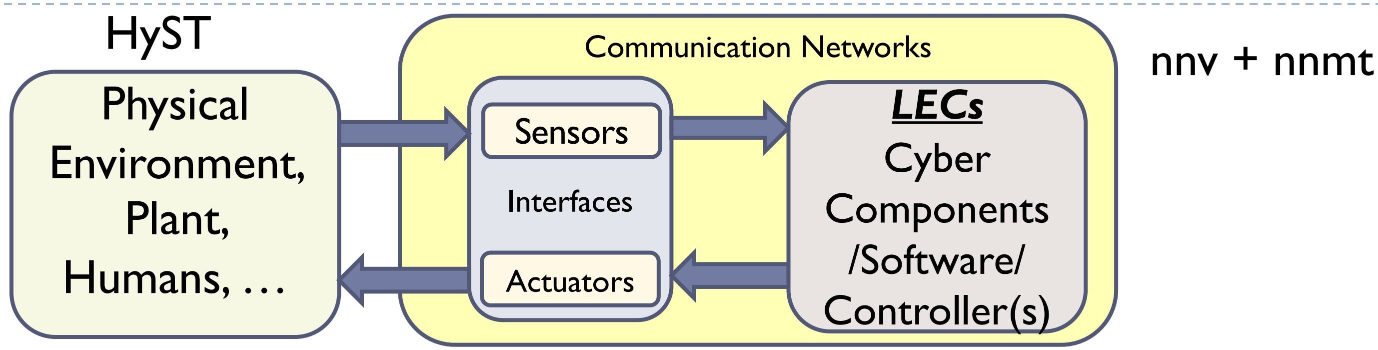
Layer-by-Layer Propagation of Polytopes





$$Y = F(I) = ?$$

Closed-Loop Control with LECs: Verification Flow and Tools



https://github.com/verivital/hyst
https://cps-vo.org/group/hyst

https://github.com/verivital/nnv
https://github.com/verivital/nnmt

- Plant models: **hybrid automata**, or networks thereof, represented in HyST/SpaceEx/CIF formats
  - Hybrid automaton: **finite state machine** + set of real-valued variables that evolve continuously over intervals of real time according to **ordinary differential equations (ODEs)**
  - **Hybrid** behaviors: discrete transitions and continuous trajectories over real time
  - Plant dynamics: linear, nonlinear, hybrid, continuous-time, discrete-time, …
- LEC and cyber models: for now, feedforward **neural networks**, represented in **ONNX** format (compatible with Keras, Tensorflow, Matlab, etc.)
  - ReLUs, CNNs (max pool, etc.), tanh, sigmoid, …
- Specifications: primarily **safety properties** for now, some **reachability properties**
- Verification: composed LEC and plant analysis: autonomous closed-loop CPS
  - **Bounded model checking**: k control periods, alternating reachability analysis of controller and plant

Symbolic State-Space Representation: Star Sets

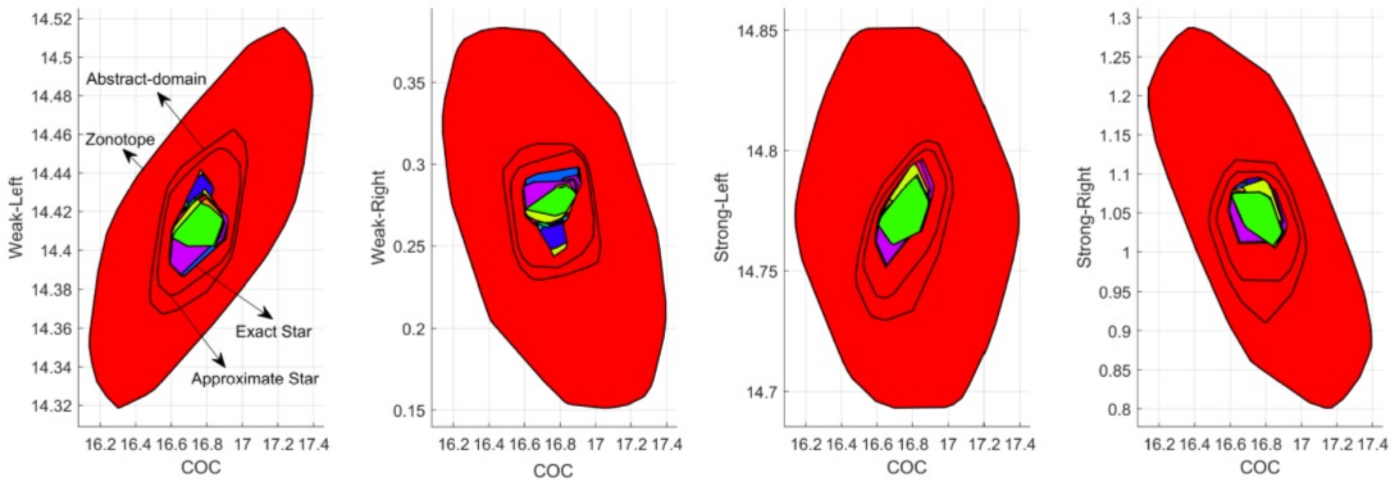
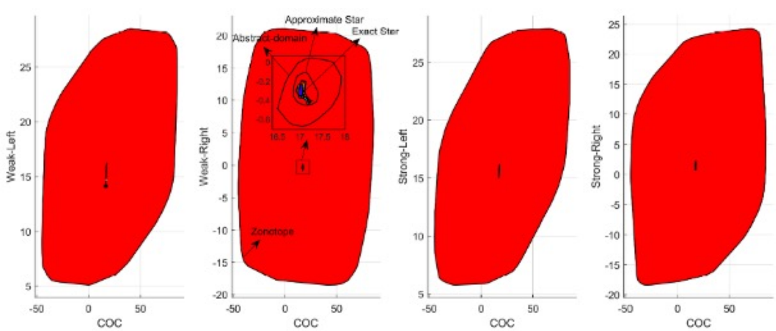
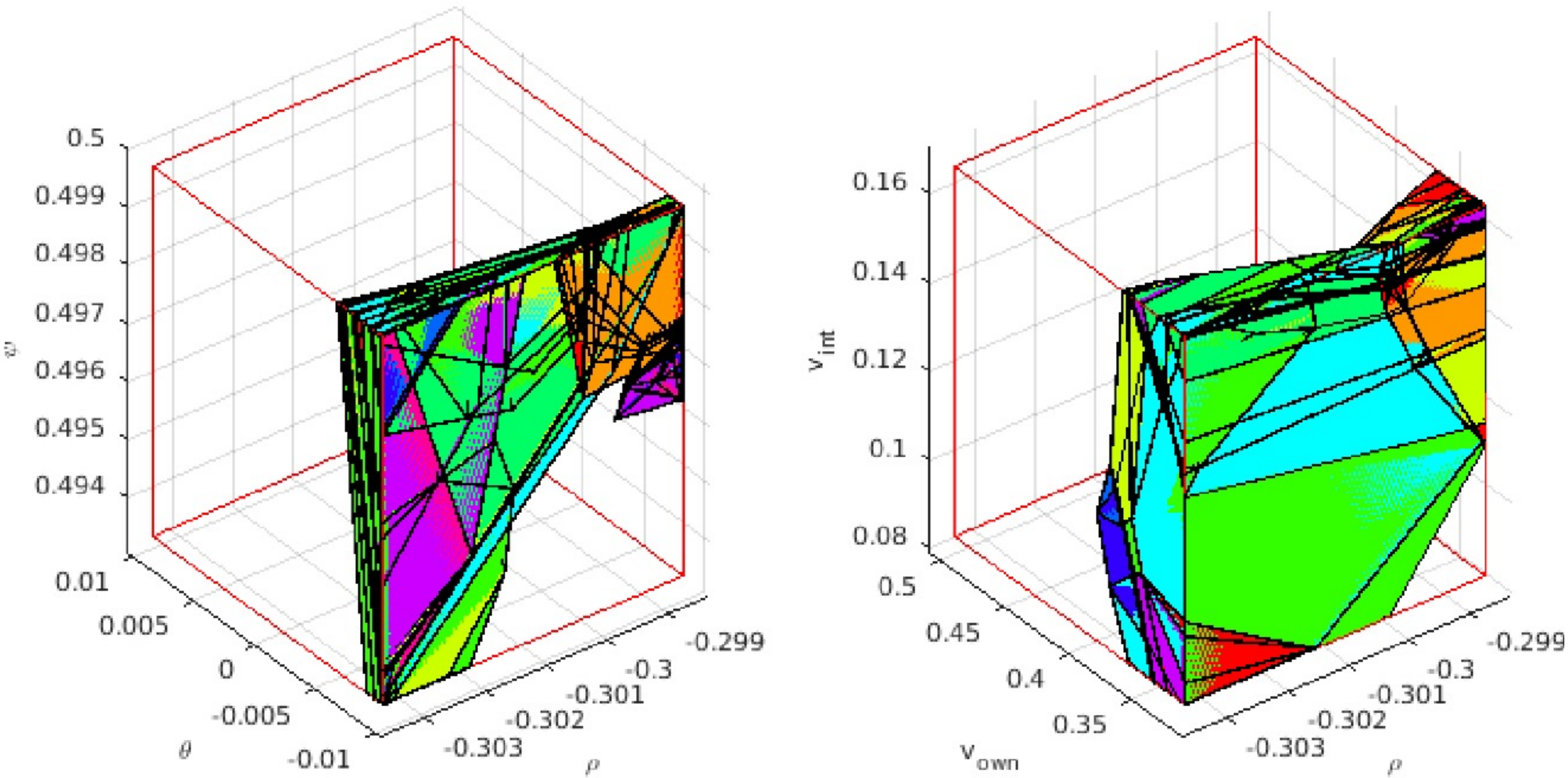


Illustration of **overapproximation conservativeness** for different symbolic state-space representations (zonotopes, abstract domains, approximate star sets, and exact star sets) within an ACAS Xu benchmark, illustrating the accuracy provided by star set representations, as they are the smallest sets



Star sets **minimize overapproximation error**, so properties may be efficiently verified with them versus other symbolic state space representations that are too imprecise (zonotopes, abstract domains, polytopes, intervals, etc.) as in DeepZ, DeepPoly, ReluVal, …

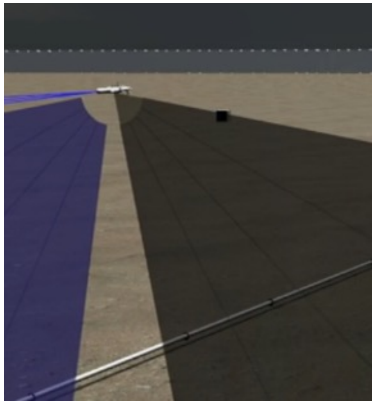
Counterexample Construction

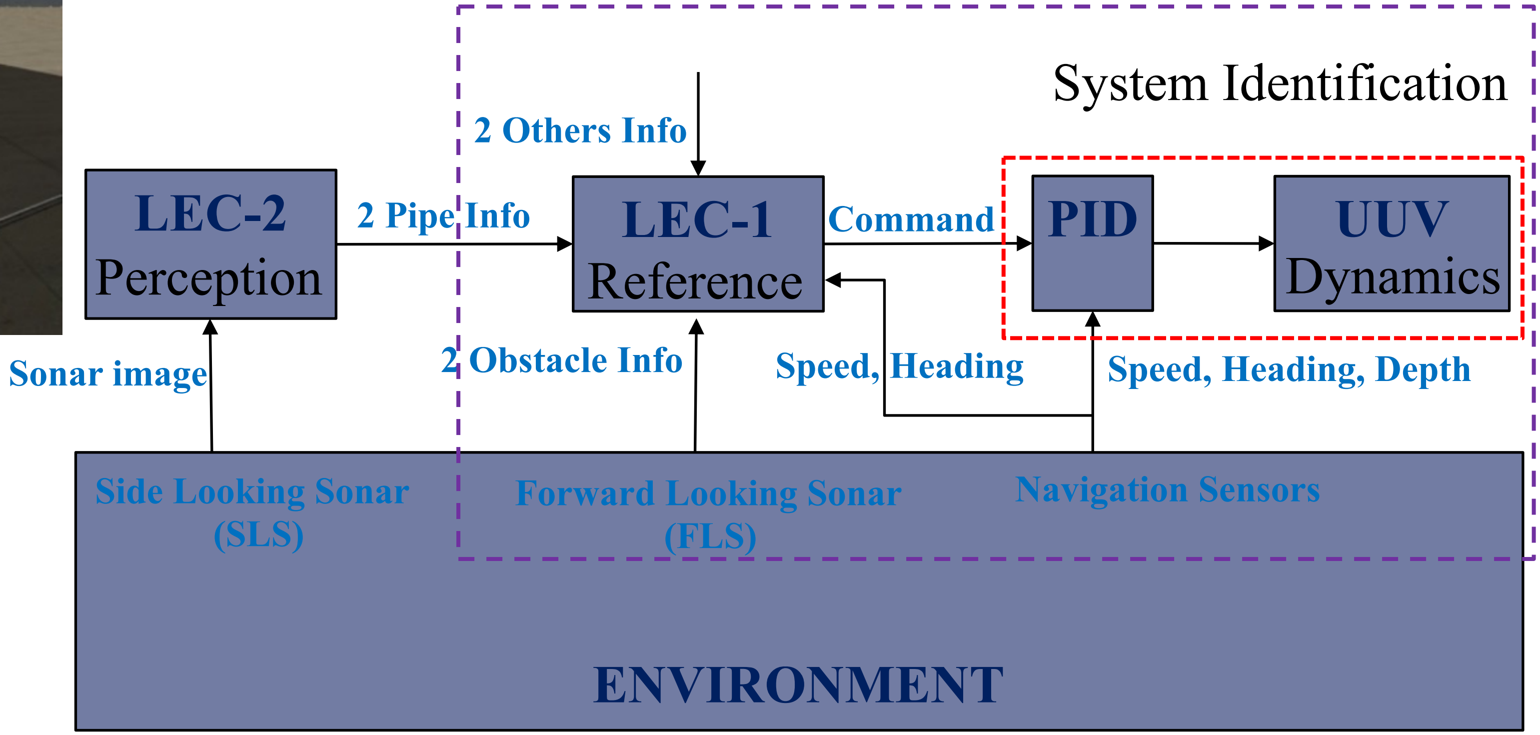


Our approach can construct a "complete" set of counterexamples for NNs

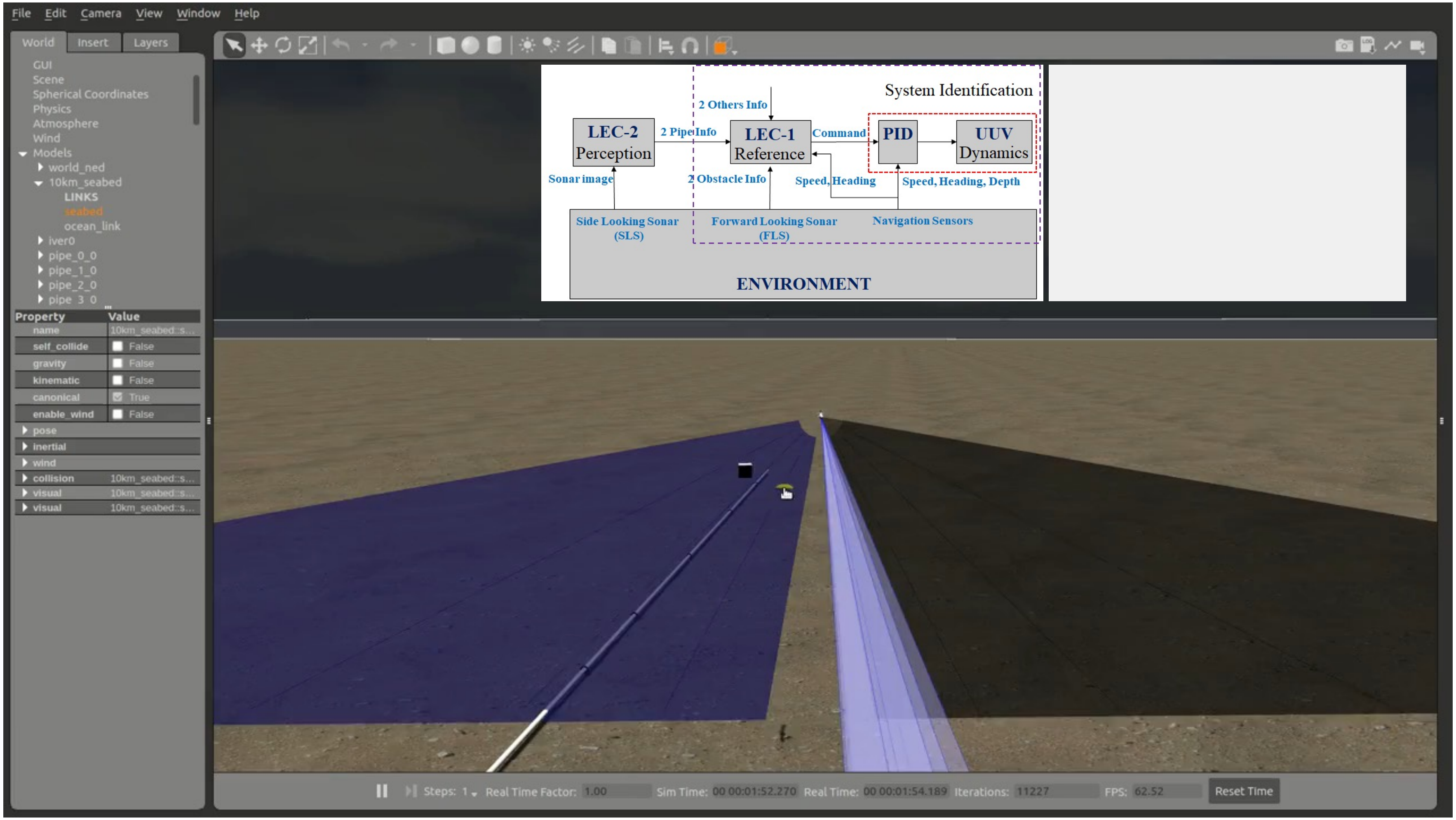
UUV One-step Safety Verification

- One-step Safety verification for NGC UUV
  - (LEC-1 + UUV plant model)

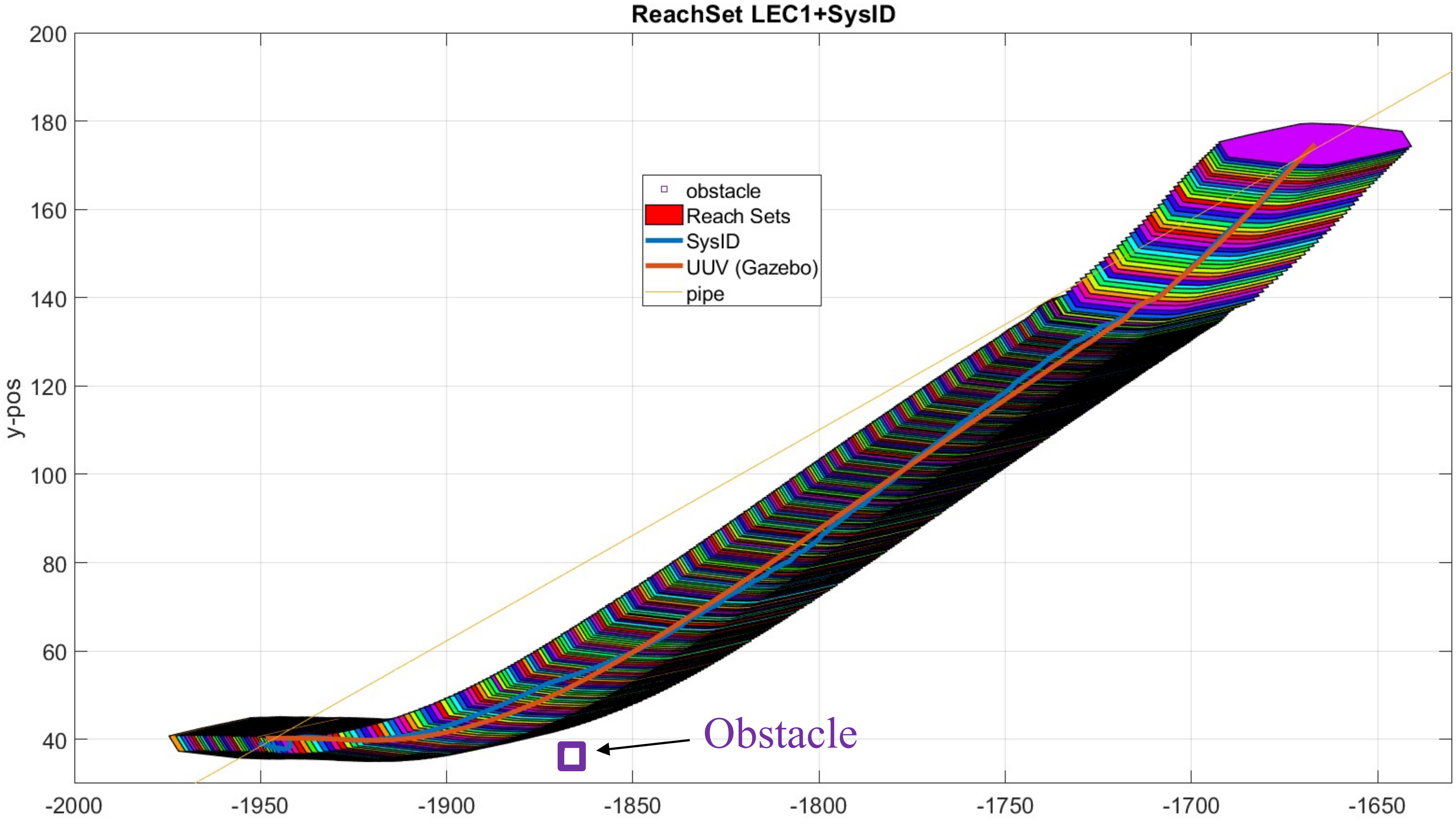




UUV One-step Safety Verification

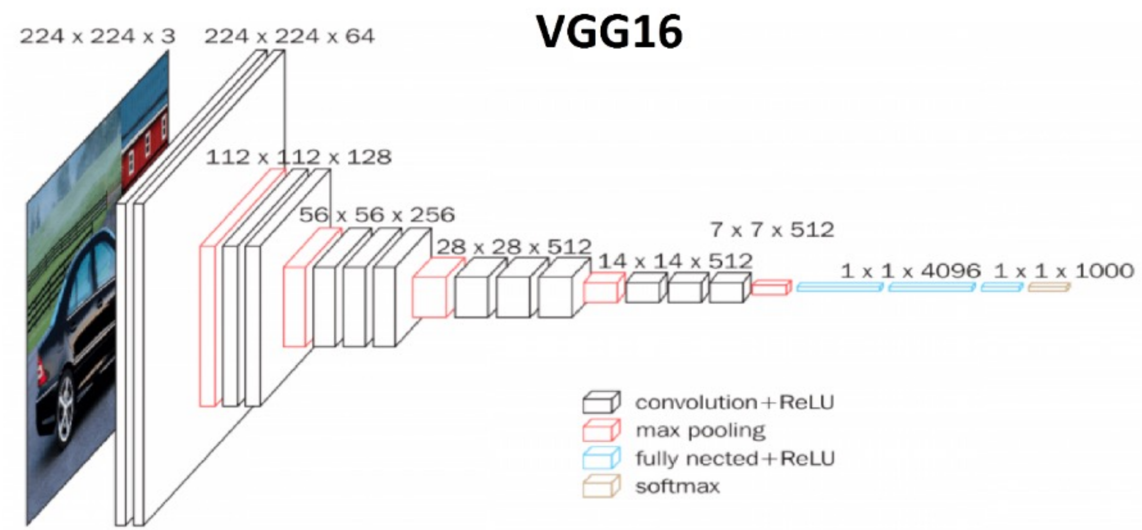


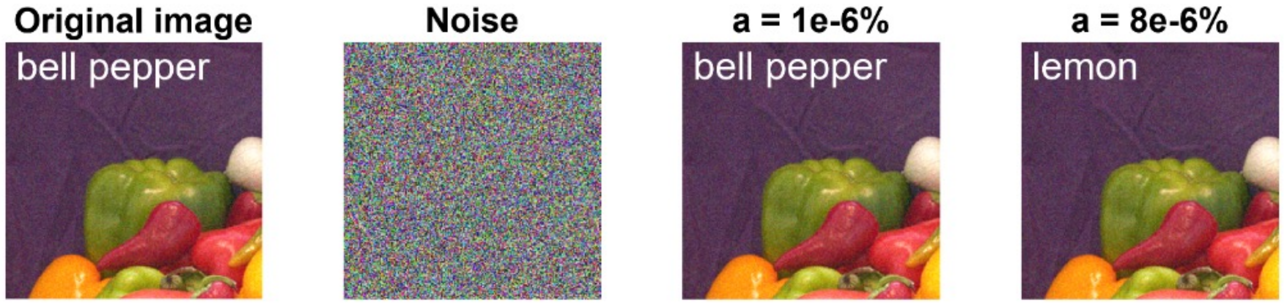
NGC UUV One-step Safety Verification



UUV does not collide with the obstacle

Robustness Verification of Perception





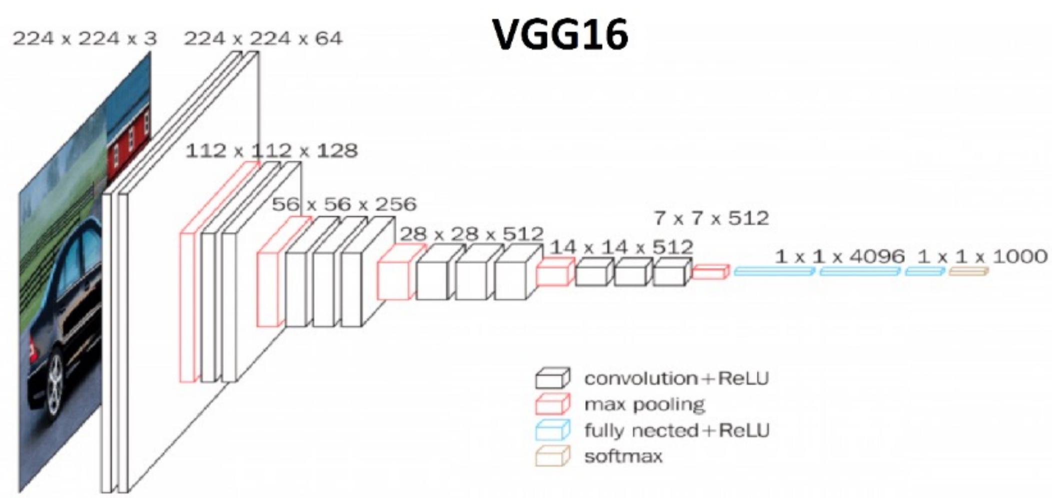
Disturbed images = Original image + a x Noise

Is VGG16 robust with FGSM attack for $a \leq 2\times10^{-8}$?

VGG16 and VGG19 Verification

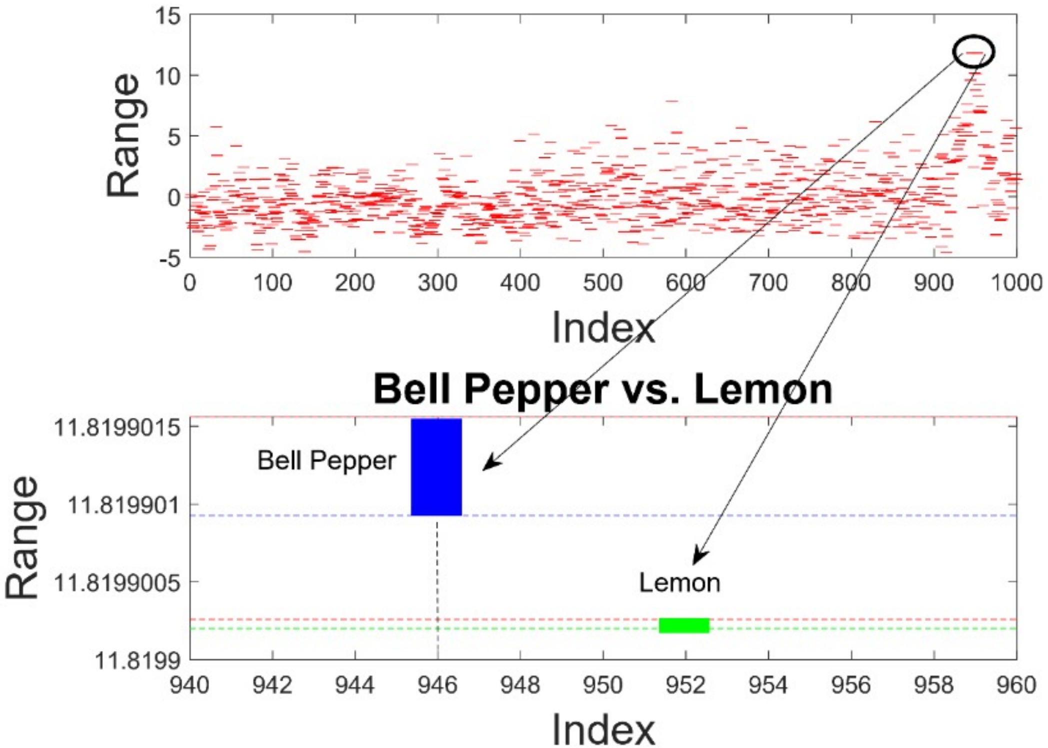
- **One of the most accurate image classifiers**
  - **~93% accuracy in top-5 classification on ImageNet**
- VGG16: **16** layers, **138M** parameters
- VGG19: **19** layers, **144M** parameters
- Classify images into **1000 classes**, e.g., car, horse, bell pepper, …

- **Layers of interests**
  - Convolutional layer
  - Average pooling layer
  - Max pooling layer
  - Fully connected layer
  - ReLU layer

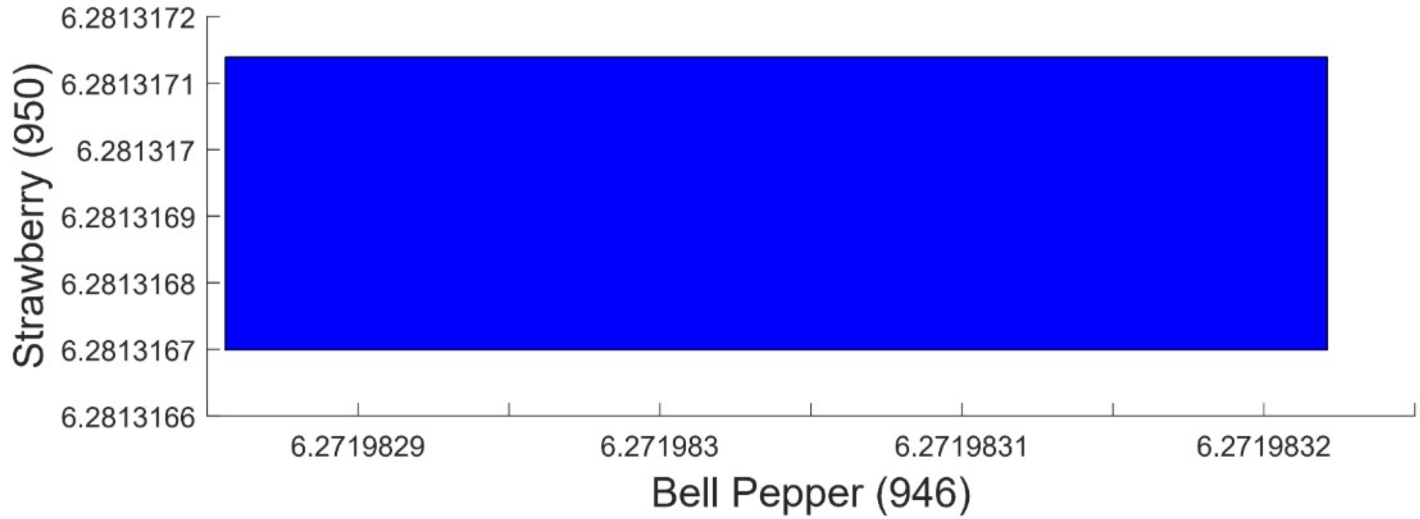


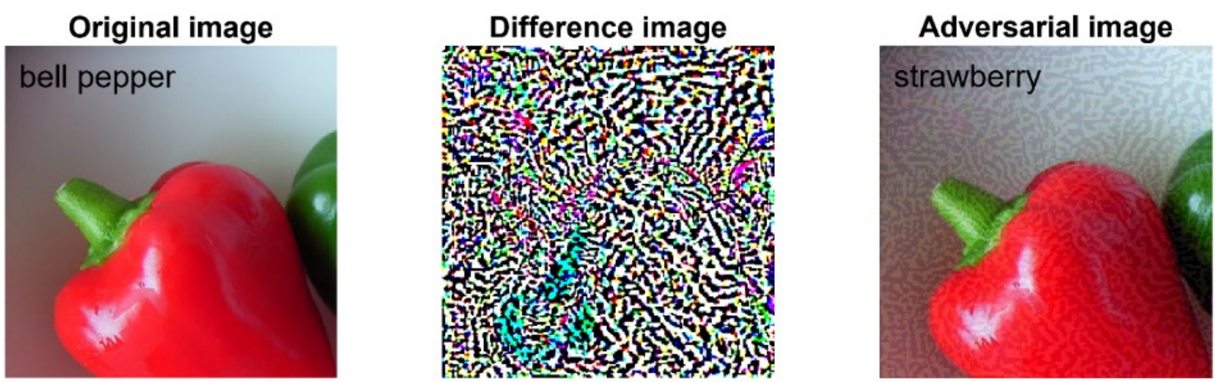
VGG16 Robustness Verification

- Reachable set computation time: **518** seconds
- Verifying Robustness Time: **56** seconds
- Number of ImageStars in the output reachable set: **8**
- Total Verification Time: **574** seconds (≈ 10 minutes)
- Number of cores: **1**
- **Robust? Yes**



VGG19 Robustness verification: Counter-examples

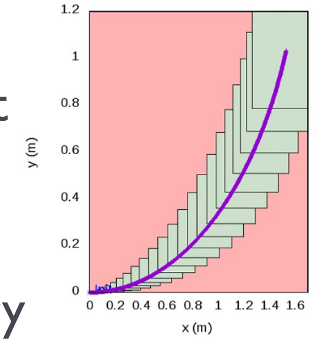




Counterexample generation

Runtime (Online) Verification of Autonomous Systems with Real-Time Reachability

- While improving confidence of such LECs before they are deployed is important, **online monitoring at runtime** is essential
- How can we provide formal and provable guarantees of system-level behaviors, such as safety, **online at runtime**?
  - Key idea: abstract LEC behaviors (see other approaches on out of distribution detection, etc.) and simply ***observe the influence of their behavior on plant/system-level at runtime***
  - Necessary technology: **online reachability analysis** of plant models, ideally with worst-case execution time (**WCET**) guarantees for implementation in embedded hardware
  - Builds on **real-time reachability** of linear/nonlinear ordinary differential equations (ODEs) and hybrid automata with WCET guarantees, implemented as an **anytime** algorithm [FORTE'19, TECS'16, RTSS'14]

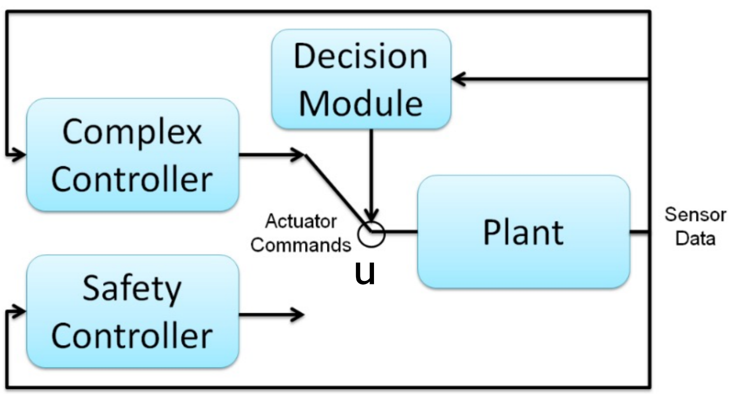


[Tran et al, "Decentralized Real-Time Safety Verification for Distributed Cyber-Physical Systems", **FORTE'19**]
[Johnson et al, "Real-Time Reachability for Verified Simplex Design", **TECS'16**]
[Bak et al, "Real-Time Reachability for Verified Simplex Design", **RTSS'14**]

http://www.verivital.com/rtreach/

Supervisory Control and Monitoring LECs in the Loop

- Complex controller: can do **anything**, be output from LECs, etc., abstracted to just produce control inputs (u) for the plant
- Assumptions: analytical (linear or nonlinear ordinary differential equation [ODE]) plant model available, and controller input remains fixed over finite time horizon
- **Supervisory control** via **Simplex architecture**
- Check these control inputs on closed-loop for a finite time horizon using **reachability analysis with real-time (WCET) guarantees**, if there's a problem, fall back to safety strategy



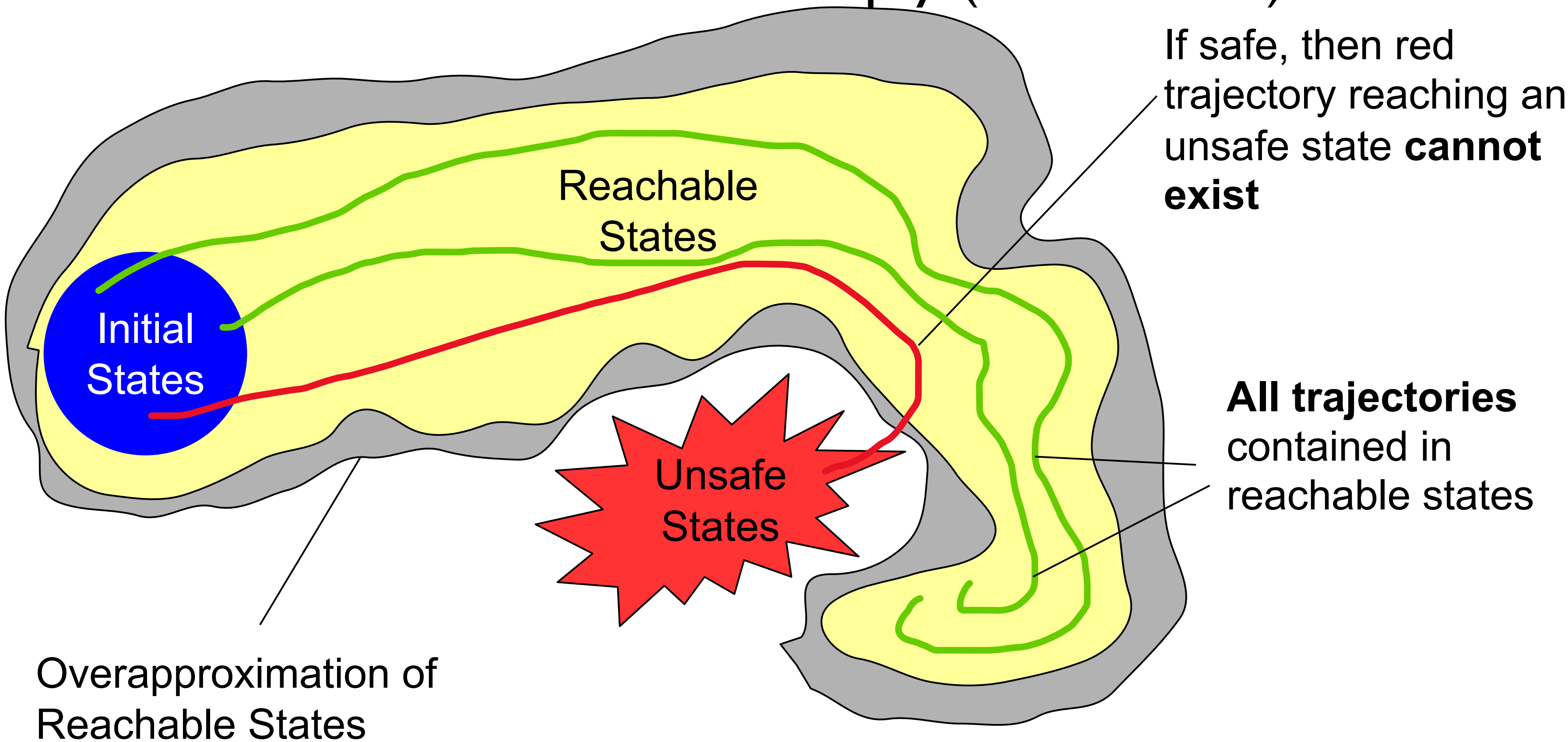
Real-time reachability algorithm implementation is cross-platform C (x86, ARM, AVR, etc.) with no dynamic memory allocation, recursion, or library dependencies

https://github.com/verivital/rtreach

[Taylor T. Johnson, Stanley Bak, Marco Caccamo, Lui Sha, "Real-Time Reachability for Verified Simplex Design", In ACM Transactions on Embedded Computing Systems (TECS), 2016 / RTSS'14]

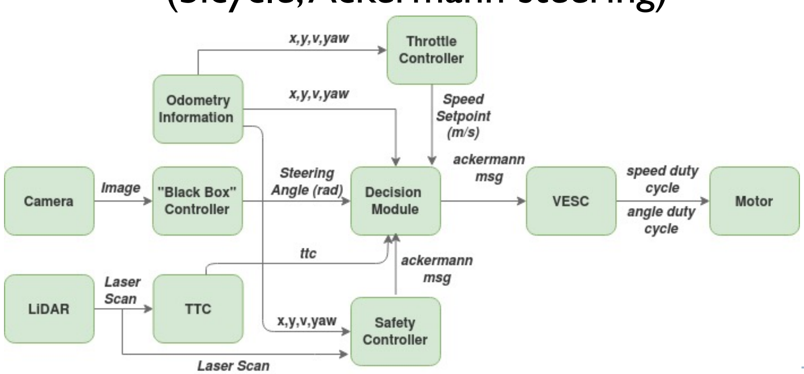
Safety Verification with Reachability

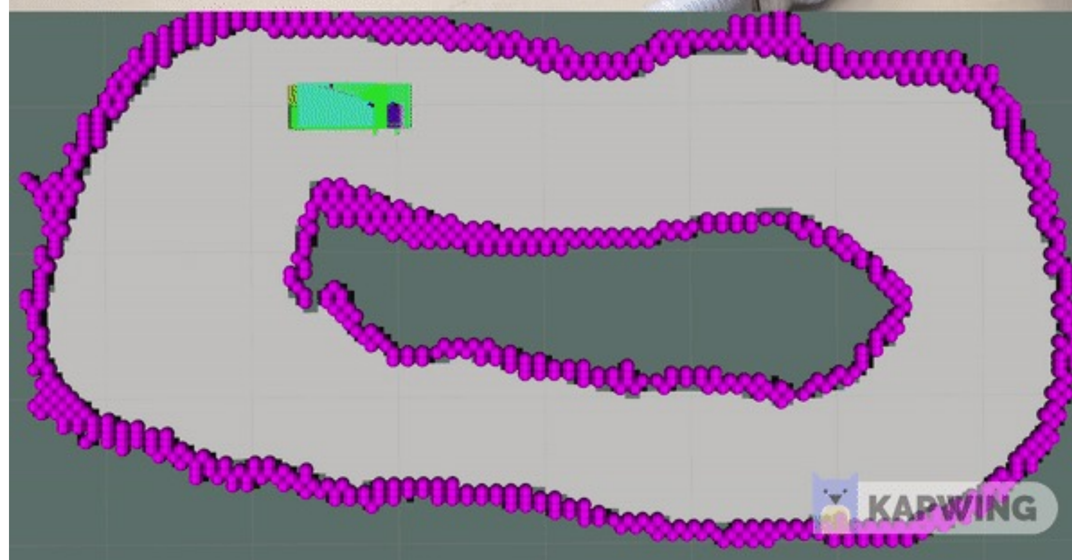
- **<u>Safe</u>** if intersection of overapproximation of reachable states with unsafe states is empty (**soundness**)



F1/10 Ground Vehicle End-to-End (E2E) LEC Demo

- End-to-end (E2E) controller: takes images and produces steering control inputs
- Classification-based control: determining steering angle (straight, weak left, weak right, etc.) with fixed speed
- Reachable sets visualized below right: if intersection with obstacles occurs, use fallback safety controller
- Plant model: nonlinear ODEs (bicycle, Ackermann steering)

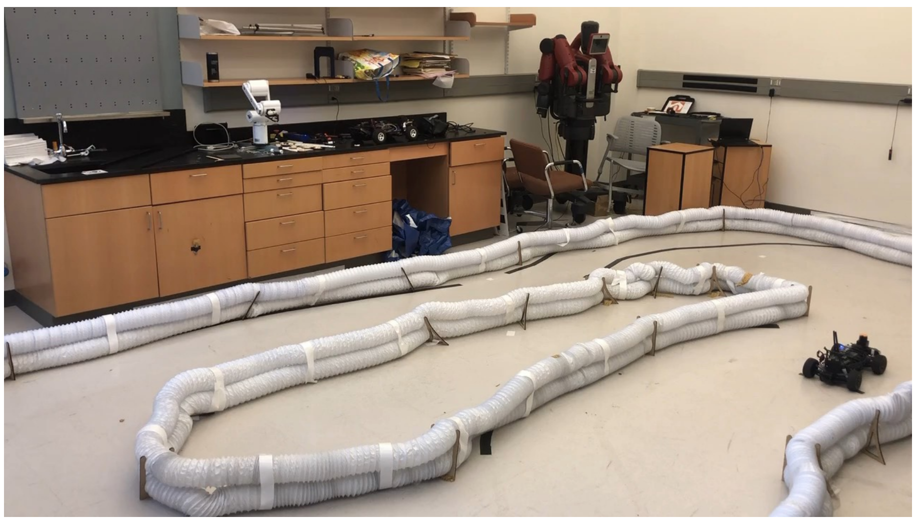


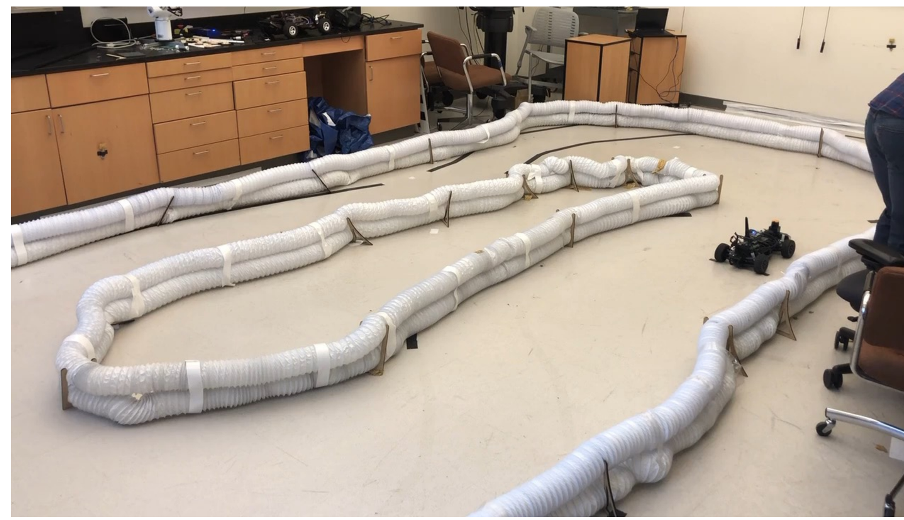


F1/10 Ground Vehicle Demo Comparison

E2E Control Without Runtime Verification: collisions

E2E Control With Runtime Verification: no collisions

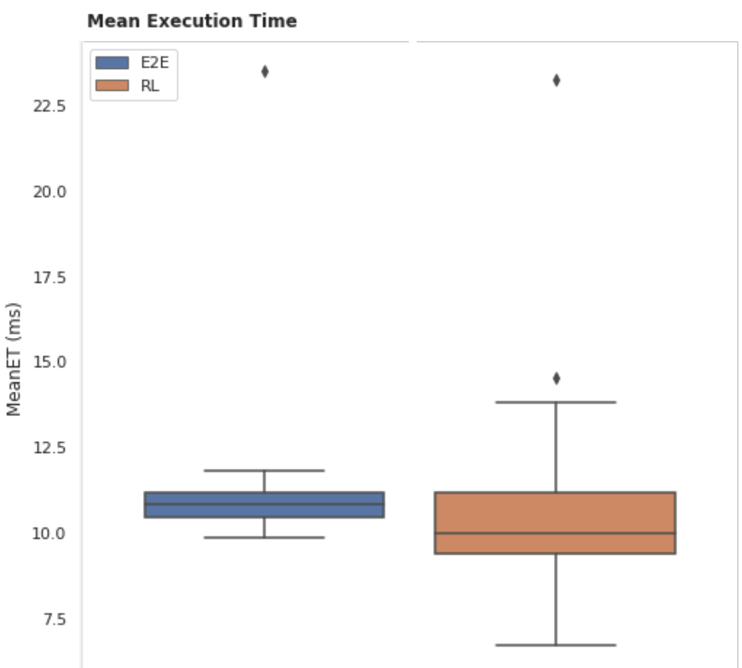


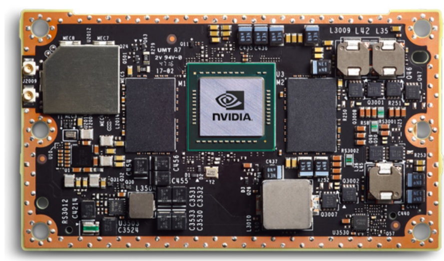


Complex controller: VGG-based neural network taking camera images as input and producing steering angles at constant vehicle speed (1m/s)
Safety controller: slower vehicle speed (0.7m/s) gap following method

Runtime Performance Evaluation

- Evaluated 20 runs each with E2E and RL based controllers monitored with real-time reachability, on NVIDIA Jetson TX2 (ARM), running on Denver 2 cores
- Mean execution time overhead: ~7-20ms





Technical Specifications

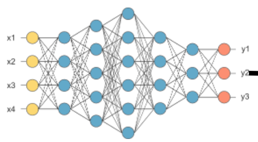
GPU	256-core NVIDIA Pascal™ GPU architecture with 256 NVIDIA CUDA cores
CPU	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM® Cortex®-A57 MPCore
Memory	8GB 128-bit LPDDR4 Memory 1866 MHx - 59.7 GB/s
Storage	32GB eMMC 5.1
Power	7.5W / 15W

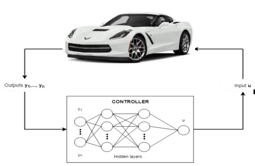
Previous work: Design-Time (Offline) Verification with NNV

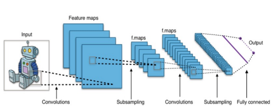
Tool available standalone: https://github.com/verivital/nnv
Also integrated in Vanderbilt ALC toolchain

The Neural Network Verification (NNV) Tool

$\boldsymbol{M} \triangleq$







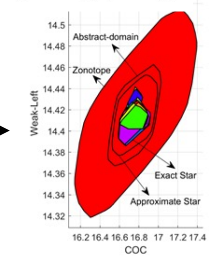
Feedforward Neural Networks (FFNN)

Neural Network Control Systems (NNCS)

Convolutional Neural Networks (CNN)

Reachability Solvers

Visualizer

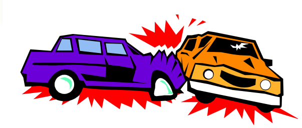


Verifier
$\boldsymbol{M} \models \boldsymbol{S}$?

No: bug

Yes: proof

$\boldsymbol{S} \triangleq \neg$



$\boldsymbol{S} \triangleq \neg$

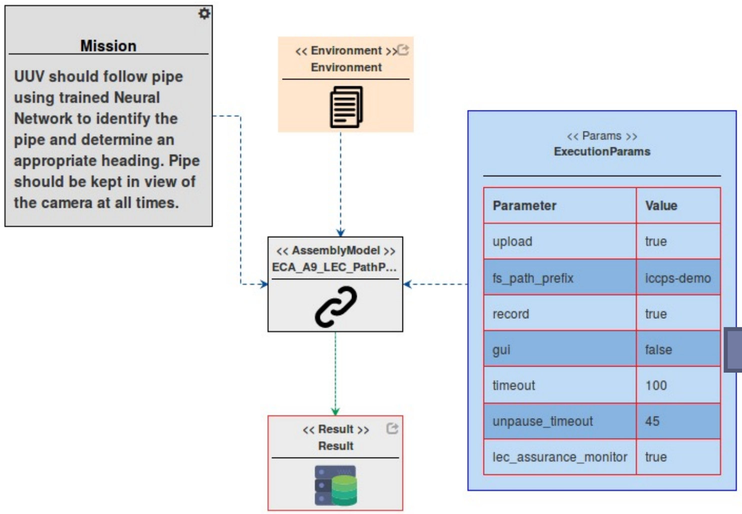
[Eykholt et al, CVPR 2018]

[Xiang et al, "Output Reachable Set Estimation and Verification for Multi-Layer Neural Networks", **TNNLS'18**]
[Tran et al, "Star-Based Reachability Analysis for Deep Neural Networks", **FM'19**]
[Tran et al, "Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control", **EMSOFT'19**]
[Tran et al, "NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems", **CAV'20**]
[Tran et al, "Verification of Deep Convolutional Neural Network using ImageStars", **CAV'20**]
[Bak et al, "Improved Geometric Path Enumeration for Verifying ReLU Neural Networks", **CAV'20**]
[Xiang et al, "Reachable Set Estimation for Neural Network Control Systems: A Simulation-Guided Approach", **TNNLS'20**]

ALC Toolchain

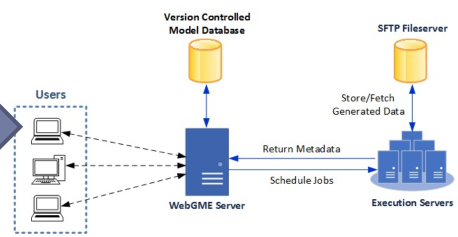
Continued

LEC Construction:
3. Evaluation: Testing/Verification

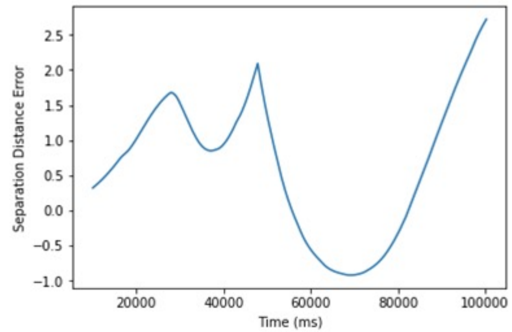


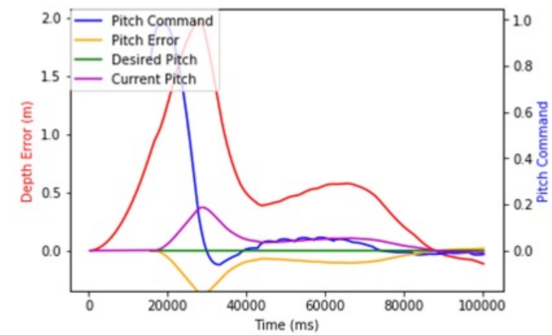
Analysis in Jupyter Notebook

Also, "single step" the process for debugging



Execute on Remote Server/s





Training Model Data Managed on GitLab

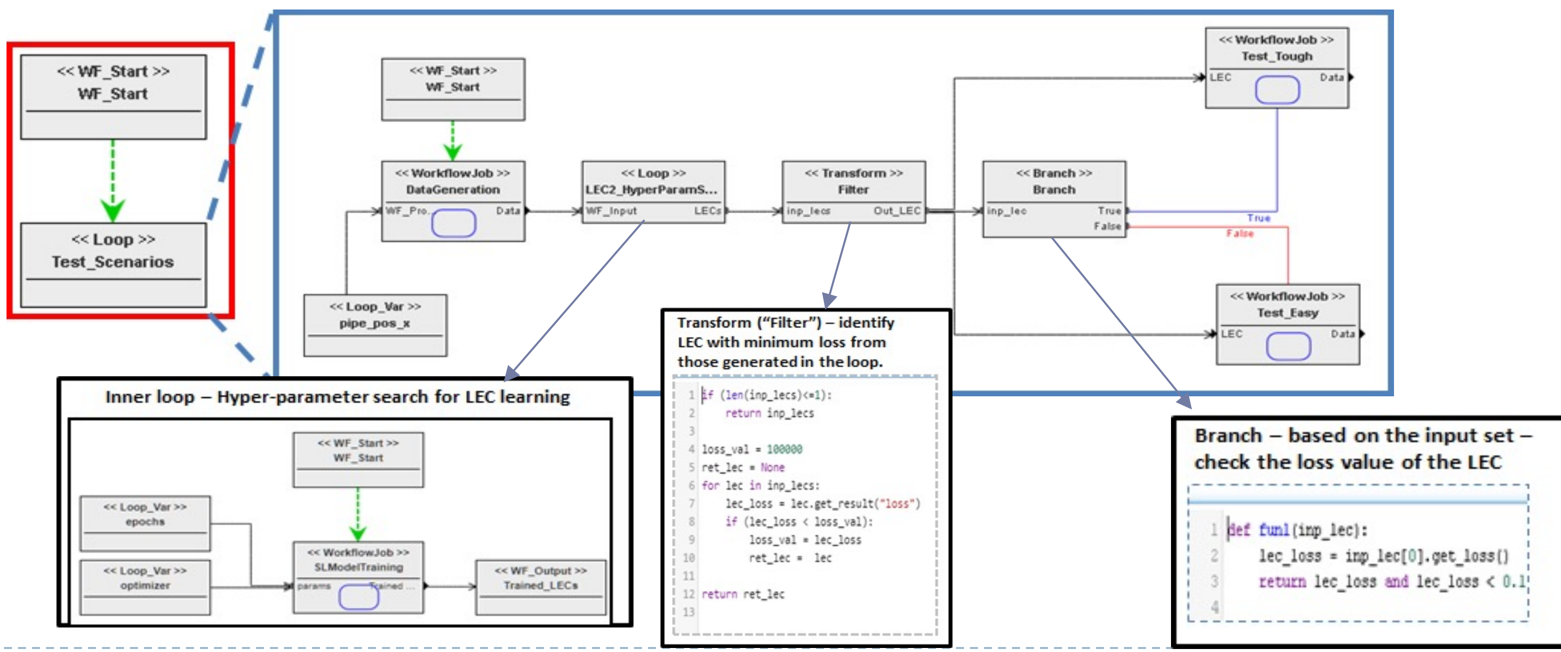
- Trained Neural Net can be tested in the simulator with another experiment model
- Performance metrics are recorded for LEC evaluation, e.g.:
  - Distance from ideal path
  - Pipe within camera field of view

Name	Type	Size	Creation Date
result-NN_Training_Test-1542127634867	*model.keras*	267 B	11/13/2018
result-NN_Training_Test-1542128784700	*model.keras*	267 B	11/13/2018

Results in file store + git, cross-linked for data provenance

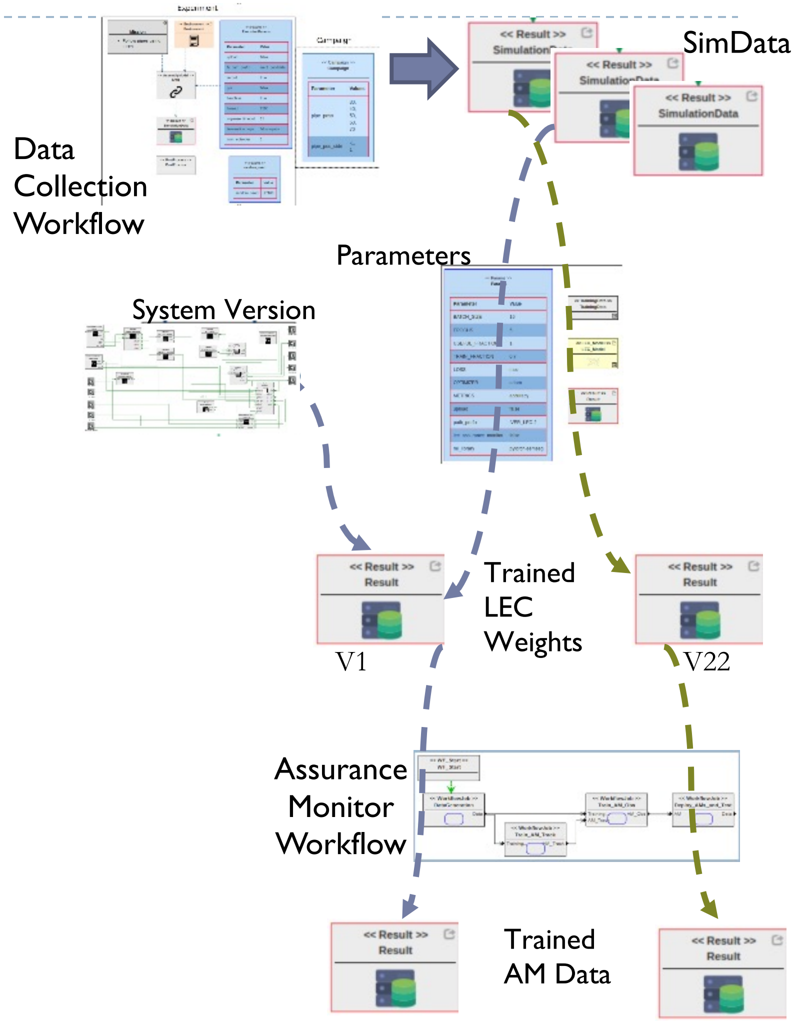
Toolchain Automation
Workflow Models

- Workflow models are for the specification and execution of job graphs
  - Each workflow job specifies execution of one or more activity models
  - Data dependencies between jobs are handled automatically
- Workflow supports
  - Loops – For (parallel), while/ do-while (sequential)
  - Transforms - Filter / Join (subset or aggregation of results)
  - Branch – execution path based on user-specified condition
- Example workflow to train and optimize a LEC

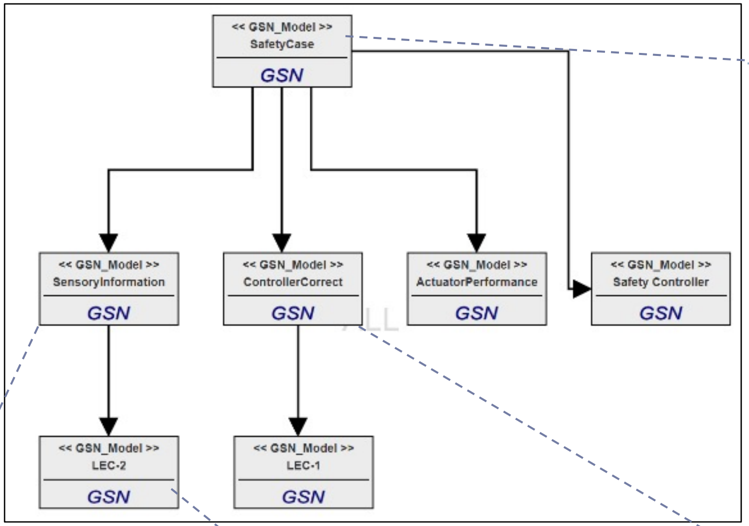


Toolchain Automation Support for Data Provenance

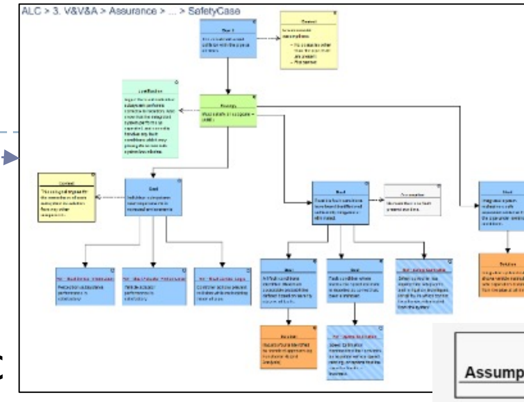
- All artifacts – generated during data collection, training, evaluation
- Recorded for each execution:
  - Parameter settings
  - LEC(s) Models (Deployed/Initial)
  - Data used in training, validation and evaluation
- Allows re-execution of any step/workflow
- Track the evolution of data/ LECs/ Assurance
- Maintain traceability links at each stage to:
  - Data used in training LECs
  - Initial trained model used in training LECs
  - LECs used in generating data sets/Assurance

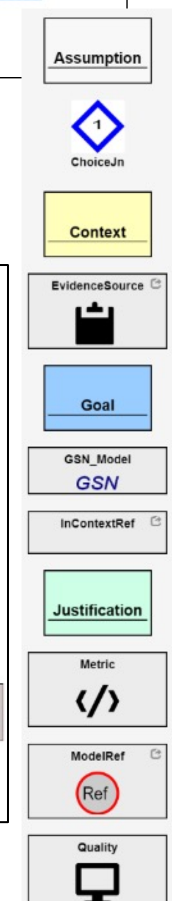


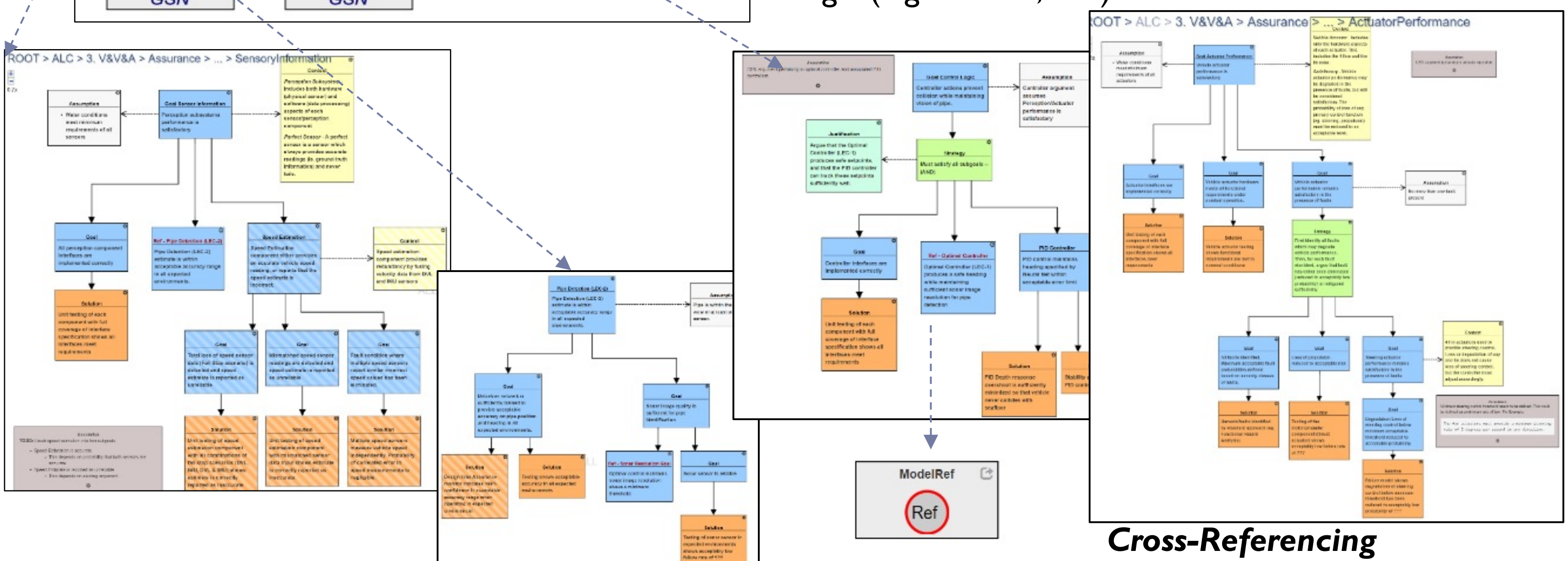
System Assurance Case: GSN



- Top-level goals correspond to high level safety claims
- Leaf goals correspond to claims which can be directly supported by evidence/solutions
- Evaluation metrics from LEC experiments can be used as evidence for leaf goals
- User Defined Combination Logic (E.g. M-of-N, etc.)







Cross-Referencing Components, Datasets, for Context/Evidence

Example GSN Model for UUV

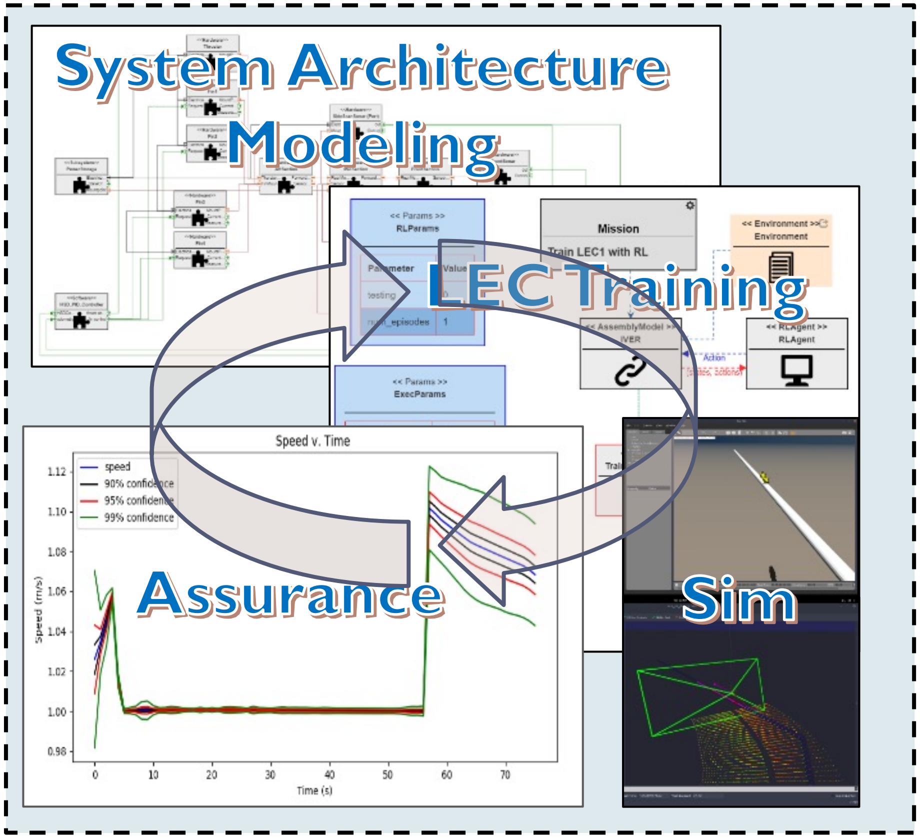
Summary: ALC Toolchain
Design Automation for CPS with LEC-s

- **Problem**

  How to support the engineering: i.e. design, analysis, implementation, and assurance of CPS that include Learning-Enabled Components (LEC)?
- **Technical Approach**

  A model-based, tool-enhanced engineering process assists in the construction, analysis, verification, and assurance of LECs in the context of the engineered system
- **Results**

  Comprehensive model-based design automation toolchain that directly supports training data collection, training, verification, and assurance of LEC-based CPS, in addition to conventional model-based systems and software engineering activities (architecture modeling and analysis, software synthesis, simulation, and others). All activities are configured and orchestrated via graphical models, all engineering data (including training data) is archived in a version-controlled project database, for reproducibility.



Hartsell, Ch. Et al. "Model-based design for CPS with learning-enabled components." In Proceedings of the Workshop on Design Automation for CPS and IoT, pp. 1-9. ACM, 2019.