Code-Level Verification for Autonomy (TC2)

Robust and Resilient Autonomy for Advanced Air Mobility

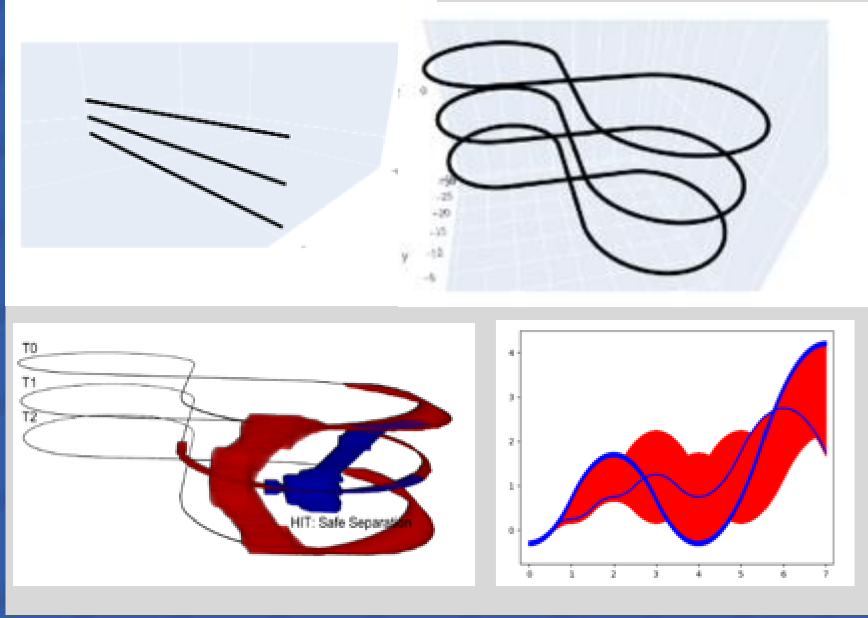
Yangge Li and **Lin Song**
Haoqing Zhu, Katherine Braught, Keyi Shen
Sayan Mitra & Chuchu Fan
University of Illinois Urbana-Champaign and
Massachusetts Institute of Technology

March 24, 2023

Credits: NASA / Lillian Gipson

AVIATE CENTER
University of Illinois Urbana-Champaign
Massachusetts Institute of Technology
Georgia Tech
North Carolina Agricultural and Technical State University
University of Nevada, Reno
Lockheed Martin
Sierra Nevada Corporation

```python
def decisionLogic(ego: State, others: List[State], track_map):
  next = copy.deepcopy(ego)
  if ego.tactical_mode == TacticalMode.Normal:
    if any((is_close(ego, other) and ego.track_mode==other.track_mode) for other in
     ↪ others):
      next.tactical_mode = TacticalMode.MoveDown
      next.track_mode = track_map.h(ego.track_mode, ego.tactical_mode,
       ↪ TacticalMode.MoveDown)
    if any((is_close(ego, other) and ego.track_mode==other.track_mode) for other in
     ↪ others):
      next.tactical_mode = TacticalMode.MoveUp
  # ...
  if ego.tactical_mode == TacticalMode.MoveUp:
    if in_interval(track_map.altitude(ego.track_mode)-ego.z, -1, 1):
      next.tactical_mode = TacticalMode.Normal
      next.track_mode = track_map.h(ego.track_mode, ego.tactical_mode,
       ↪ TacticalMode.Normal)
  # ...
  return next
```



Outline

Introduction to Verse verification tool
https://github.com/AutoVerse-ai/Verse-library

- Quadrotor example
- Demo

Background theory: Data-driven verification

- Reachability and sensitivity

Applications

- L1 Adaptive control [Lin Song et al., ICCPS WIP ’23]
- DNN control [Puthumanaillam, Ornik, et al.]
- RL air-traffic management [Peng Wei, GWU, ongoing]
- Parallel Verse [Zhu, et al.]

Verification problem and the Verse tool

Input

System + Requirements

Multi-agent hybrid scenarios + Invariance / safety

Output

Counter-example or

Proof that all behaviors of the System meet Requirements

Basic Verse Approach

Probabilistic sensitivity analysis +

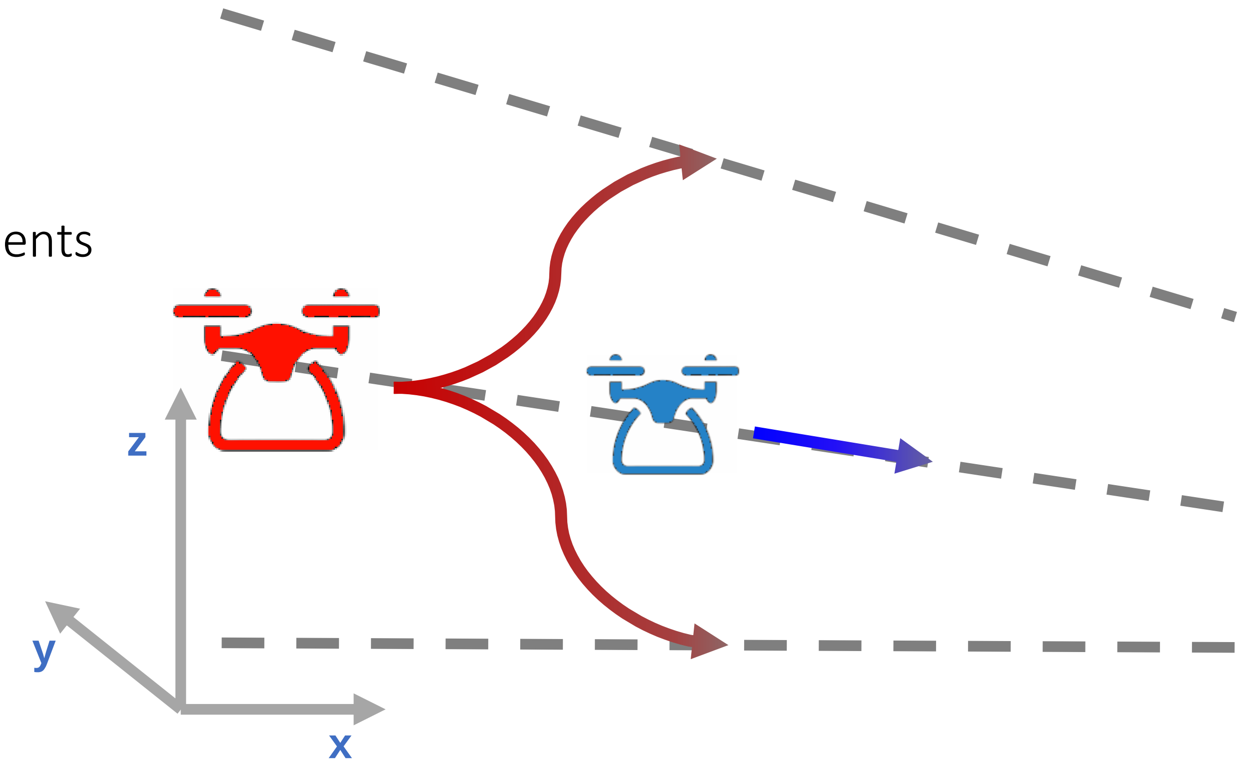
Deterministic reachability analysis

Advanced Approaches

Parallelized verification

Incremental verification

Verification with neural network controllers



Creating scenarios in Verse --- quick and easy

Drones flying along three parallel vertically separated straight tracks

Scenario in Verse defined by a set of *agents*, a *map*, and a *sensor*

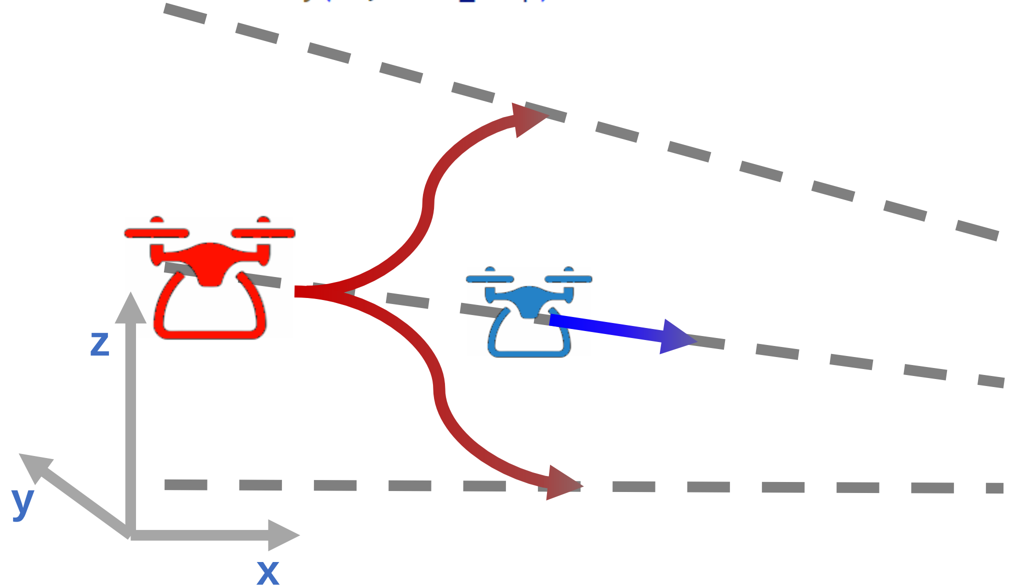
Maps define sequences of waypoints or motion primitives

Sensors specify information available to one agent about other agents

Agent: *Decision logic* + *Dynamics*

- 6D model + Bang-bang controller
- 18D model + L1 controller

```
if __name__ == "__main__":
    scenario = Scenario()
    drone_red = DroneAgent('drone_red', file_name='drone_controller.py')
    drone_red.set_initial([init_l_1, init_u_1],(CraftMode.Normal, TrackMode.T1))
    scenario.add_agent(drone_red)
    drone_blue = DroneAgent('drone_blue', file_name='drone_controller.py')
    scenario.add_agent(drone_blue)
    # ...
    scenario.set_map(M6())
    scenario.set_sensor(BaseSensor())
    #traces = scenario.simulate(40, time_step)
    traces = scenario.verify(40, time_step)
```



Writing Decision Logic

```python
def decisionLogic(ego: State, others: List[State], track_map):
    next = copy.deepcopy(ego)

    if ego.tactical_mode == TacticalMode.Normal:
        if any((is_close(ego, other) and ego.track_mode == other.track_mode)\
            for other in others):
            if track_map.Tg_exist(ego.track_mode, ego.tactical_mode, \
                    TacticalMode.MoveUp):
                next.tactical_mode = TacticalMode.MoveUp
                next.track_mode = track_map.Tg(
                    ego.track_mode, ego.tactical_mode, TacticalMode.MoveUp)
            if track_map.Tg_exist(ego.track_mode, ego.tactical_mode, \
                    TacticalMode.MoveDown):
                next.tactical_mode = TacticalMode.MoveDown
                ...
    if ego.tactical_mode == TacticalMode.MoveUp:
        if 1 > track_map.altitude(ego.track_mode)-ego.z > -1:
            ...

    assert not any(safe_seperation(ego, other) for other in others), "Safe Seperation"

    return next
```

Informal logic: "Nondeterministically switch to track above or below if too close to any other drone in the same track"

Decision Logic (DL) modifies agent's mode written in Python

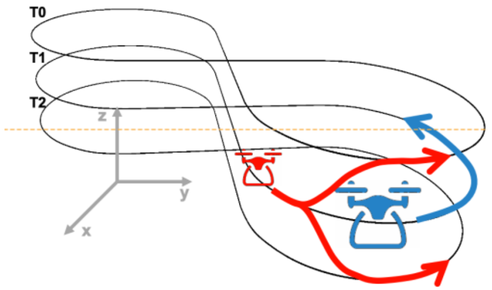
If exists multiple possible transitions, both will be explored

Using *any* and *all* to quantify over other agents in the scenario

DL supports *user defined functions*

Safety specified using python *asserts* (no need to learn new logics)

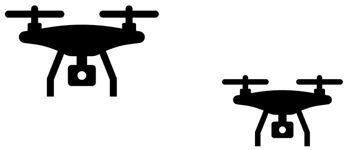
Verse under the hood



Scenario

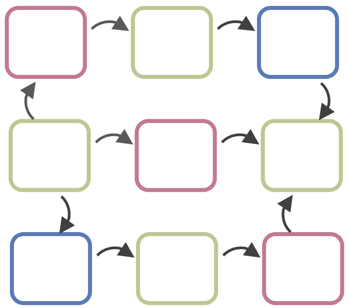
```
def decisionLogic(ego: State, others: List[State], track_map):
    next = copy.deepcopy(ego)

    if ego.tactical_mode == TacticalMode.Normal:
        if any((is_close(ego, other) and ego.track_mode == other.tı
            for other in others):
            if track_map.Tg_exist(ego.track_mode, ego.tactical_mode
                    TacticalMode.MoveUp):
                next.tactical_mode = TacticalMode.MoveUp
                next.track_mode = track_map.Tg(
                    ego.track_mode, ego.tactical_mode, TacticalMode
            if track_map.Tg_exist(ego.track_mode, ego.tactical_mode
                    TacticalMode.MoveDown):
                next.tactical_mode = TacticalMode.MoveDown
                ...
    if ego.tactical_mode == TacticalMode.MoveUp:
        if 1 > track_map.altitude(ego.track_mode)-ego.z > -1:
            ...

    assert not any(safe_seperation(ego, other) for other in others)

    return next
```

Decision logic



Multi-agent interactions

Transition logic



transition structure extracted

Executable access to mode dynamics

+

Black-box physics simulator

Learning sensitivity from data **[Fan CAV17]**

Reachability, composition, fixpoints

Learning reachability functions [Sun TACAS22]

Exploiting symmetries in dynamics [Sibai CAV21, TACAS19]

Core hybrid verification algorithms

Verse scenario to hybrid system

Consider a scenario SC with k *agents*,

Agent i has:

- Continuous state space $\mathrm{X_i}$
- Mode space $\mathrm{D_i}$
- Guard $\mathrm{G_i}$ and reset $\mathrm{R_i}$ extracted from Decision Logic. For a pair of modes $\mathrm{d, d' \in D_i}$
  - $\mathrm{G_i(d, d') \subseteq \prod_j X_j}$ defines transition condition
  - $\mathrm{R}_i\mathrm{(d, d') : \prod_j X_j \to X_i}$ defines change of continuous state after transition
- Flow function $\mathrm{F_i : X_i \times D \times \mathbb{R}^{\geq 0} \to X_i}$

Verse scenario to hybrid system cont.

Hybrid system $\mathrm{H(SC)} = \langle \mathrm{X}, \mathrm{X}^0, \mathrm{D}, \mathrm{d}^0, \mathrm{G}, \mathrm{R}, \mathrm{TL} \rangle$ from composition of agents

Continuous state space $\mathrm{X} = \prod_{\mathrm{k}} \mathrm{X_i}$

- An element $\mathrm{x} \in \mathrm{X}$ is called a state; X^0: set of initial states

Mode space $\mathrm{D} = \prod_{\mathrm{k}} \mathrm{D_i}$

- An element $\mathrm{d} \in \mathrm{D}$ is called a mode. d^0: an initial mode

For a pair of modes $\mathrm{d}, \mathrm{d}' \in \mathrm{D}$

- guard $\mathrm{G(d, d')} \subseteq \mathrm{X}$, $\mathrm{x} \in G(d, d')$ iff $\mathrm{x} \in G_i(d_i, d_i')$ and $d_j = d_j'$ for $j \neq i$
- reset $\mathrm{R(d, d')}: \mathrm{X} \to \mathrm{X}$, $R(d, d')(X) = \prod_k R_i(d, d')(X)$

TL is a set of pairs $\langle \xi, \mathrm{d} \rangle$

- Trajectory $\xi: [0, \mathrm{T}] \to \mathrm{X}$ describes evolution of continuous states in mode $\mathrm{d} \in \mathrm{D}$
- Given $\mathrm{d} \in \mathrm{D}$, $\mathrm{x} \in \mathrm{X}$, $\forall \mathrm{t}, \xi(\mathrm{t}) = \prod_j \mathrm{F_j}\left(\mathrm{x_j}, \mathrm{d_j}\right)(\mathrm{t})$
- $\xi.fstate, \xi.lstate, \xi.ltime$ the first state $\xi(0)$, the last state of the trajectory $\xi(T)$, and $\xi.ltime = T$

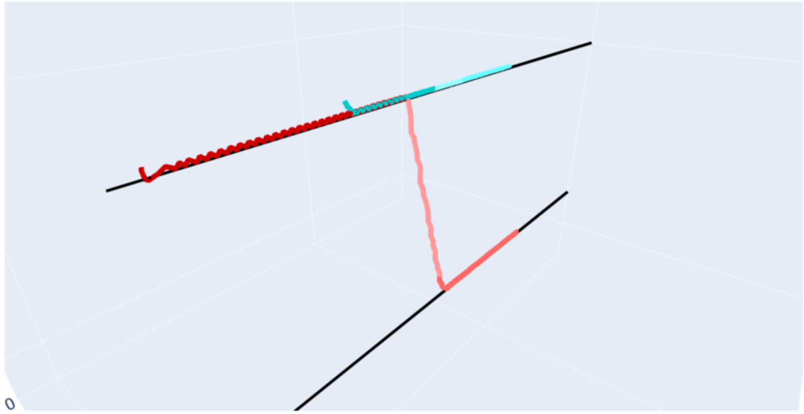
Execution and reachable states for $H(SC)$

δ-execution of $\mathrm{H(SC)}$ is a sequence of m labeled trajectories $\alpha \coloneqq \langle \xi^0, d^0 \rangle, \ldots, \langle \xi^{m-1}, d^{m-1} \rangle$

- $\xi^0.\mathrm{fstate} \in X^0$
- $\xi^{i-1}.\mathrm{lstate} \in G(d^{i-1}, d^i)$ and $\xi^i.\mathrm{fstate} = R\left(d^{i-1}, d^i\right)(\xi^{i-1}.\mathrm{lstate}), i \in \{1, m-1\}$
- $\xi^i.\mathrm{ltime} = \delta$ if $i \neq m-1$ and $\xi^{m-1}.ltime \leq \delta$, $i \in \{1, m-1\}$

$\mathrm{Reach}\left(x^0, d^0, T_{max}\right)$: reachable states from $x^0 \in X, d^0 \in D$ along an execution α defined as $\bigcup_{i\in[0,m)} \bigcup_{t\in[0,\delta)} \xi^i(t)$, with $m = \left\lceil \frac{T_{max}}{\delta} \right\rceil$

- $\mathrm{Reach}_H(T_{max}) = \bigcup_{x^0 \in X^0} \mathrm{Reach}\left(x^0, d^0, T_{max}\right)$



To prove safety, we can check $\mathrm{Reach}_H(T_{max}) \cap \mathrm{Unsafe} = \emptyset$

Reachability Tree in Verse

For a pair of modes $\mathbf{d}, \mathbf{d}'$, define discrete and continuous post operators

- $\mathrm{postCont}(X, d) = X'$ iff $X' = \bigcup_{x \in X} \prod_K F_i(x_i, d, \delta)$
- $\mathrm{postDisc}(X, d, d') = X'$ iff $\forall x \in X$, $x \in G(d, d')$ and $X' = \bigcup_{x \in X} R(d, d')(x)$

Verse constructs reachability tree $\mathbf{Tree} = \langle \mathbf{V}, \mathbf{E} \rangle$ up to depth $\mathbf{m}$

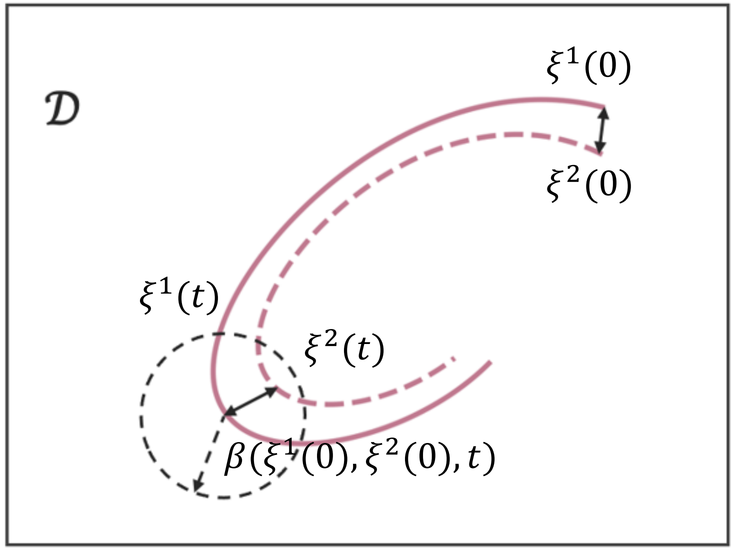
- Each vertex $\langle S, d \rangle \in V$ is a pair of a set of continuous states and a mode
- Root $\langle X^0, d^0 \rangle$
- There is an edge from $\langle S, d \rangle$ to $\langle S', d' \rangle$, iff $S' = \mathrm{postCont}(\mathrm{postDisc}(S, d, d'), d')$

Reachability tree constructed by Verse is an over-approximation of $\mathbf{Reach}_{\mathbf{H}}(\mathbf{T}_{\mathbf{max}})$

PostCont: Using Discrepancy Function

A *discrepancy function* $\beta: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^{\geq 0}$ is a uniformly continuous function such that for any pair of trajectories $\langle \xi^1, \mathrm{d} \rangle, \langle \xi^2, \mathrm{d} \rangle \in \mathrm{TL}$, and any $\mathrm{t} \in \xi^1.\mathrm{dom} \cap \xi^2.\mathrm{dom}$

- $|\xi^1(\mathrm{t}) - \xi^2(\mathrm{t})| \leq \beta(\xi^1.\mathrm{fstate}, \xi^2.\mathrm{fstate}, \mathrm{t})$
- $\beta(.,.,\mathrm{t}) \to 0$ as $\xi^1.\mathrm{fstate} \to \xi^2.\mathrm{fstate}$

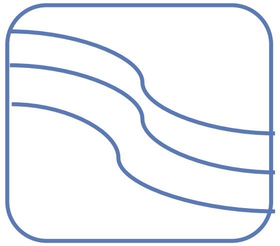


Compute $\mathbf{postCont}$ for input set of states $\mathrm{X}^0 = \mathrm{Ball}(\mathrm{x}^0, \mathrm{r})$ in mode $\mathbf{d}$

- Obtain trajectory ξ^0 starting from x^0 labeled by mode $\mathbf{d}$
- Obtain discrepancy function β
- An over-approximation of reachable set can be obtained by $\cup_t \xi^0(\mathrm{t}) \oplus \beta(\mathrm{x}_0, \mathrm{x}_0 + \mathrm{r}, \mathrm{t})$

Learn discrepancy from data

Use a template for exponential discrepancy $\beta(x_1, x_2, t)$



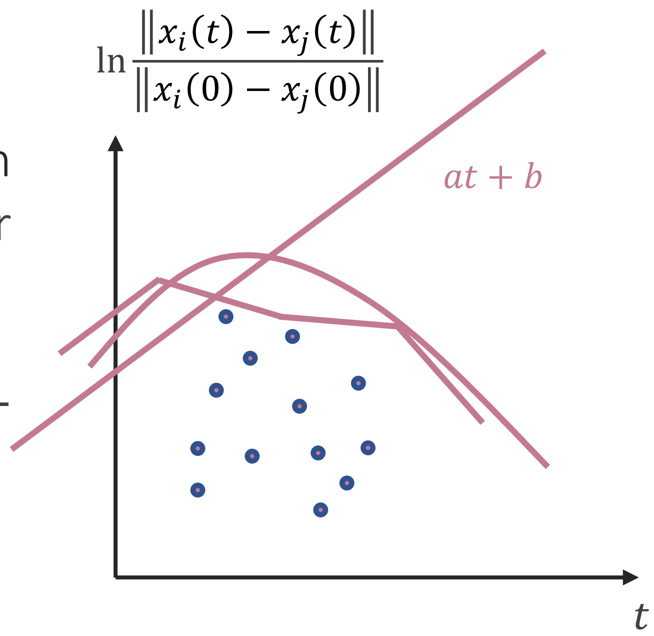
$$\|x_1(t) - x_2(t)\| \le \beta(x_1, x_2, t) = \|x_1(0) - x_2(0)\| e^{at+b}$$

Taking log: $\forall t, \ln \frac{\|x_1(t)-x_2(t)\|}{\|x_1(0)-x_2(0)\|} \le at + b$

Find a and b by **learning a linear separator**

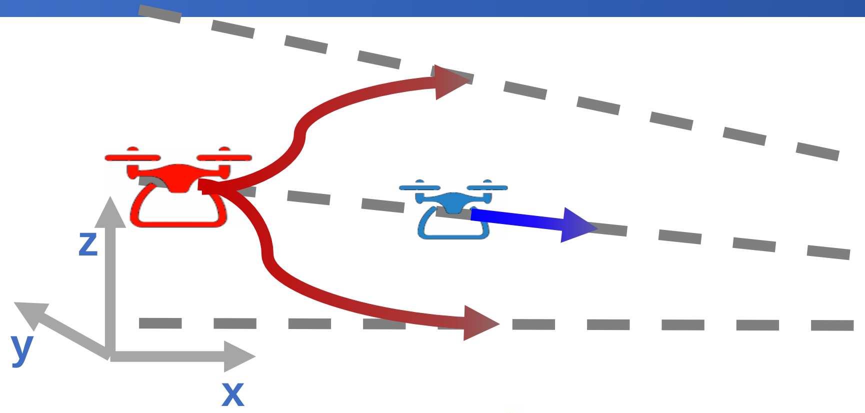
Theorem [CAV17]: Given the training set, the global exponential discrepancy function that gives the tightest reach set over-approximation can be found by solving a Linear programming (LP) problem

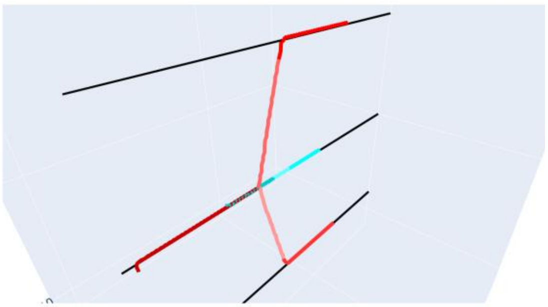
Proposition [CAV17]: $\forall \epsilon, \delta > 0$, if sampling number $n \ge \frac{1}{\epsilon} \ln \frac{1}{\delta}$, then with probability $1 - \delta$, the algorithm finds (a, b) such that $err_{\mathcal{D}}(a, b) < \epsilon$

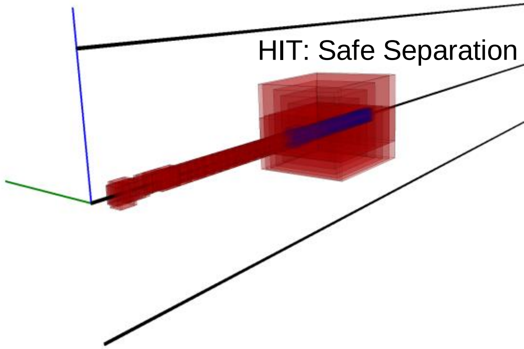


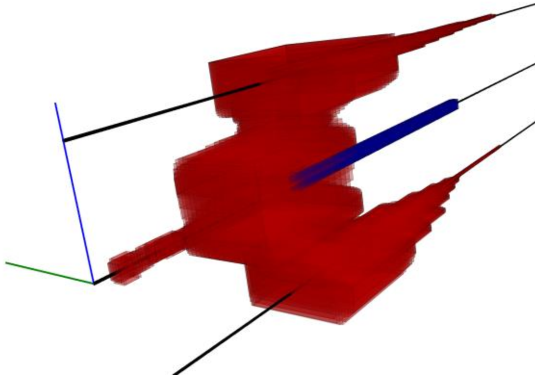
Result highlights

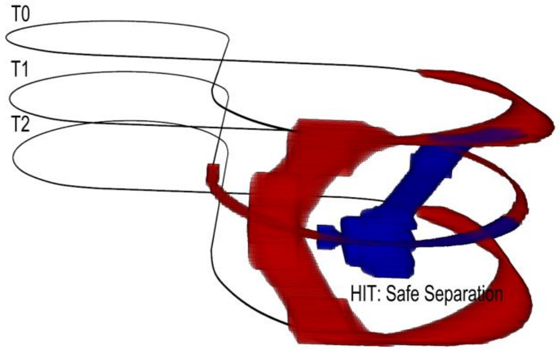
- Computed trajectory of 2-drone scenario
- Reachability analysis find potential safety violation
- By Modifying parameters in the decision logic, we can mitigate the safety violation
- Easily modify scenario to test more interesting behaviors of the agents
  - Further show with live demo





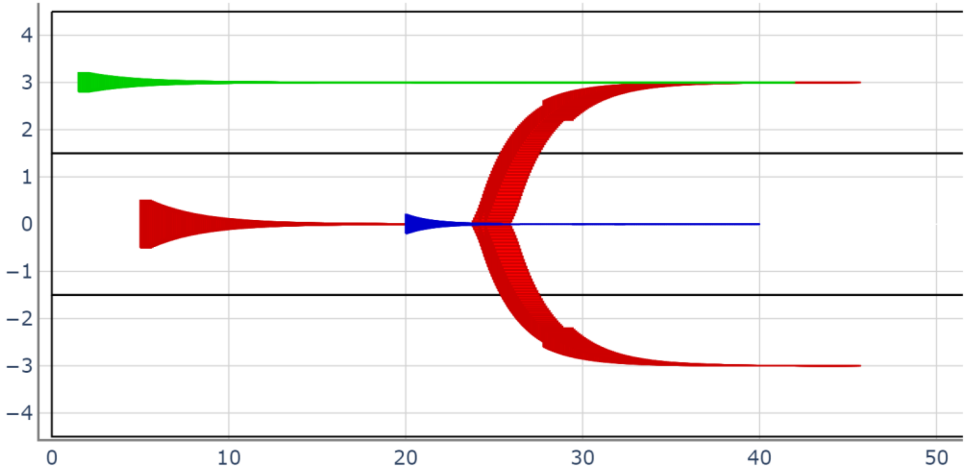


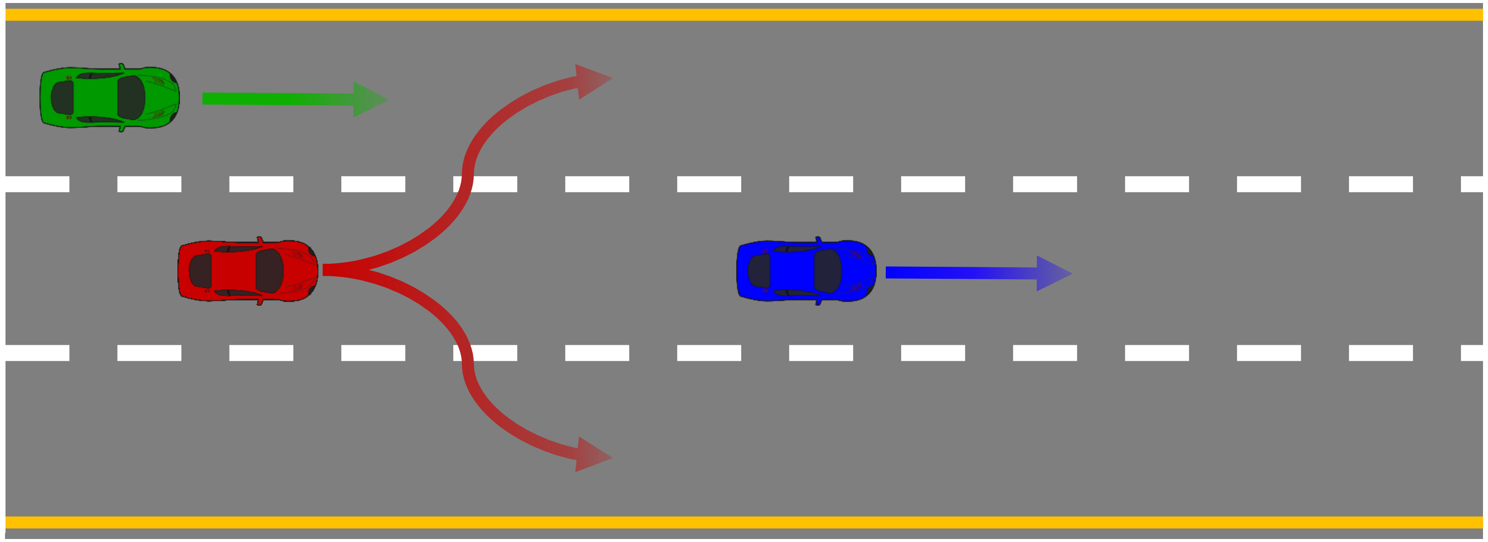




Live Demo1: Lane Switch Scenario

- Three cars
- Red and green cars running at 1 m/s and blue cars running at 0.5m/s
- Red car can perform lane switch when there’s another car 5m in front
- Three vehicles all have uncertain initial condition
- Safety condition: The red vehicle should be 1m away from all other vehicles.



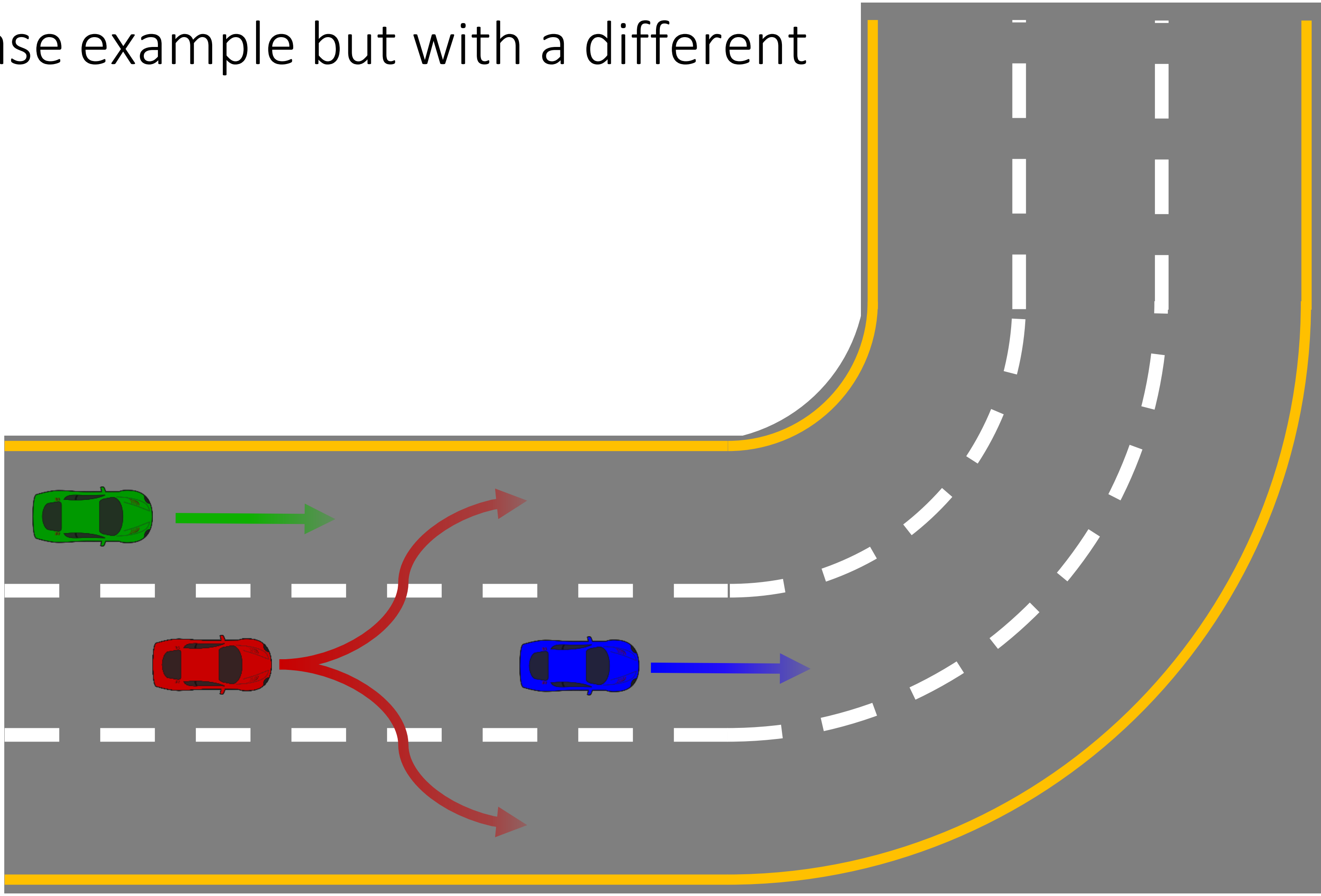


Live Demo2: Easy Modification Detect Safety Violation

- Exact same setting as base example but with a different Map

```
car = CarAgent('car1', file_name=input_code_name)
scenario.add_agent(car)
car = NPCAgent('car2')
scenario.add_agent(car)
car = NPCAgent('car3')
scenario.add_agent(car)
tmp_map = M1()
scenario.set_map(tmp_map)
```

```
car = CarAgent('car1', file_name=input_code_name)
scenario.add_agent(car)
car = NPCAgent('car2')
scenario.add_agent(car)
car = NPCAgent('car3')
scenario.add_agent(car)
tmp_map = M3()
scenario.set_map(tmp_map)
```

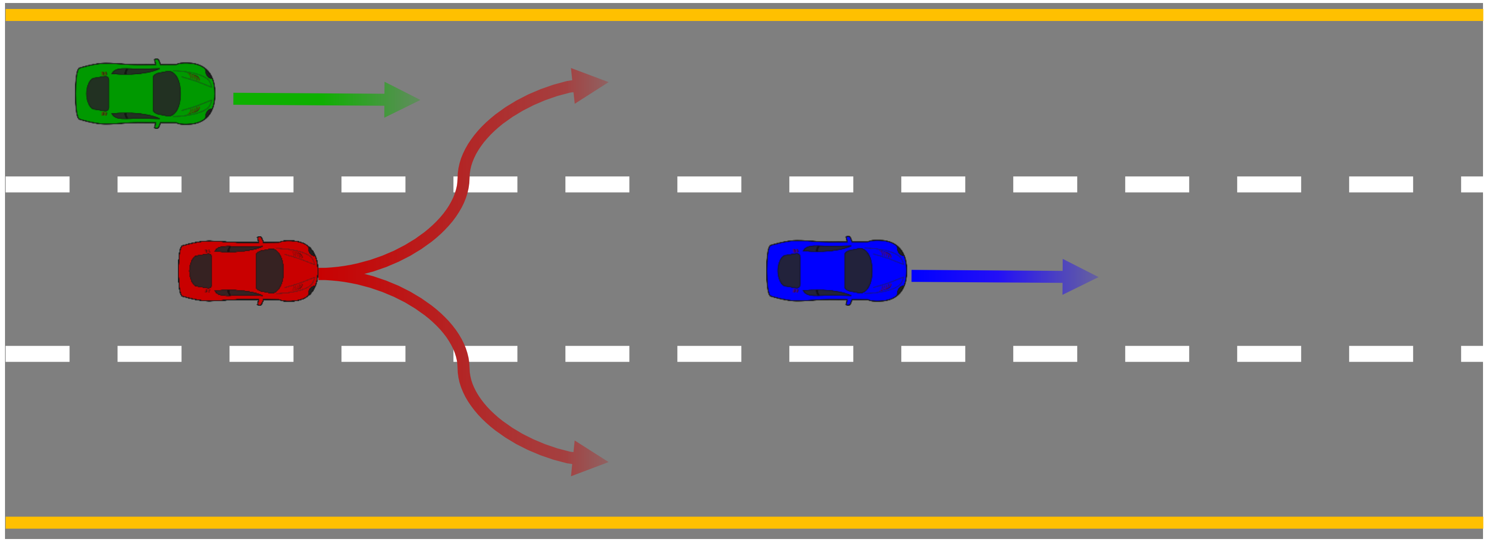


Live Demo3: Handle Uncertainty in Perception

- Exact same setting as base example but with different sensor model with noise

```python
car = CarAgent('car1', file_name=input_code_name)
scenario.add_agent(car)
car = NPCAgent('car2')
scenario.add_agent(car)
car = NPCAgent('car3')
scenario.add_agent(car)
tmp_map = M1()
scenario.set_map(tmp_map)
```

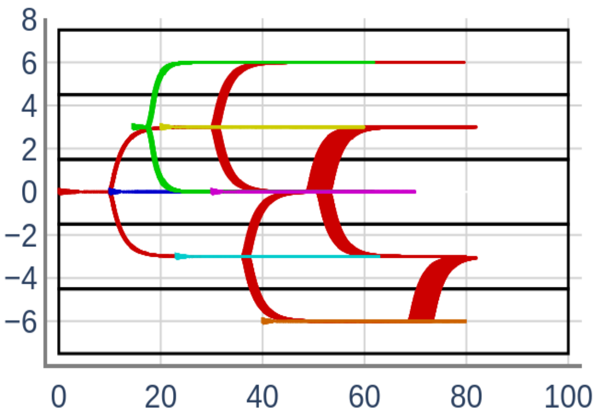
```python
config = ScenarioConfig(init_seg_length=5)
scenario = Scenario()
car = CarAgent('car1', file_name=input_code_name)
scenario.add_agent(car)
car = NPCAgent('car2')
scenario.add_agent(car)
car = NPCAgent('car3')
scenario.add_agent(car)
tmp_map = M1()
scenario.set_map(tmp_map)
scenario.set_sensor(NoisyVehicleSensor(\
    (0.5, 0.5), (0.0, 0.0)))
```

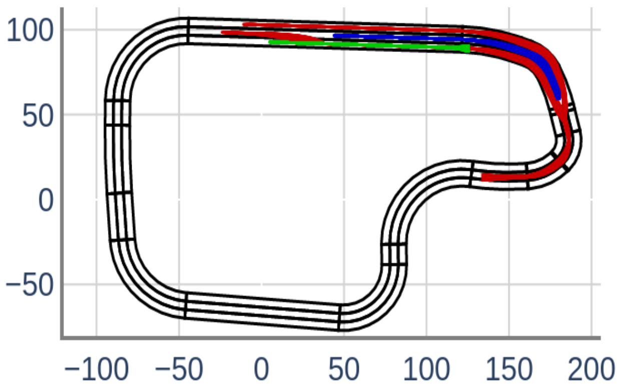


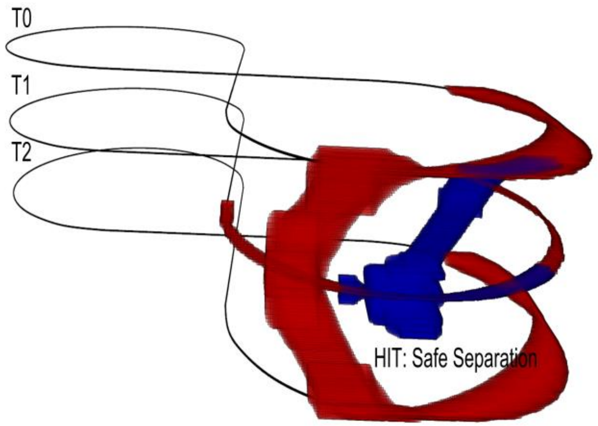
More Scenarios Verified by Verse

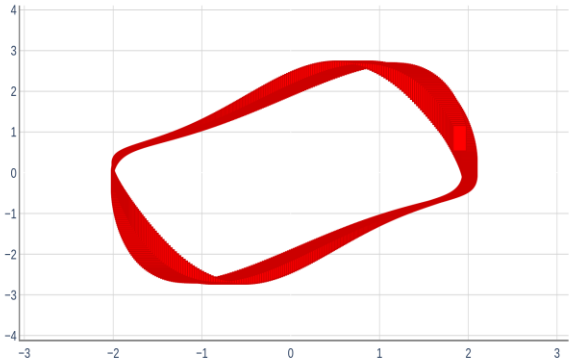
Table 1: Runtime for verifying examples in Section 5. Columns are: number of agents ($\#\mathcal{A}$), agent type ($\mathcal{A}$), map used (Map), reachability engine used (`postCont`), sensor type (Noisy $\mathcal{S}$), number of mode transitions #TR, and the total run time (Rt). N/A for not available.

$\#\mathcal{A}$	$\mathcal{A}$	Map	`postCont`	Noisy $\mathcal{S}$	#Tr	Rt (s)	$\#\mathcal{A}$	$\mathcal{A}$	Map	`postCont`	Noisy $\mathcal{S}$	#Tr	Rt (s)
2	D	$\mathcal{M}6$	DryVR	No	8	55.9	2	D	$\mathcal{M}5$	DryVR	No	5	18.7
2	D	$\mathcal{M}5$	NeuReach	No	5	1071.2	3	D	$\mathcal{M}5$	DryVR	No	7	39.6
7	C	$\mathcal{M}2$	DryVR	No	37	322.7	3	C	$\mathcal{M}1$	DryVR	No	5	23.4
3	C	$\mathcal{M}3$	DryVR	No	4	34.7	3	C	$\mathcal{M}4$	DryVR	No	7	118.3
3	C	$\mathcal{M}1$	DryVR	Yes	5	29.4	2	C	$\mathcal{M}1$	DryVR	No	5	21.6
2	C	$\mathcal{M}1$	NeuReach	No	5	914.9	1	V	N/A	DryVR	N/A	1	0.33
1	S	N/A	DryVR	N/A	3	2.3	1	G	N/A	DryVR	N/A	3	67.14



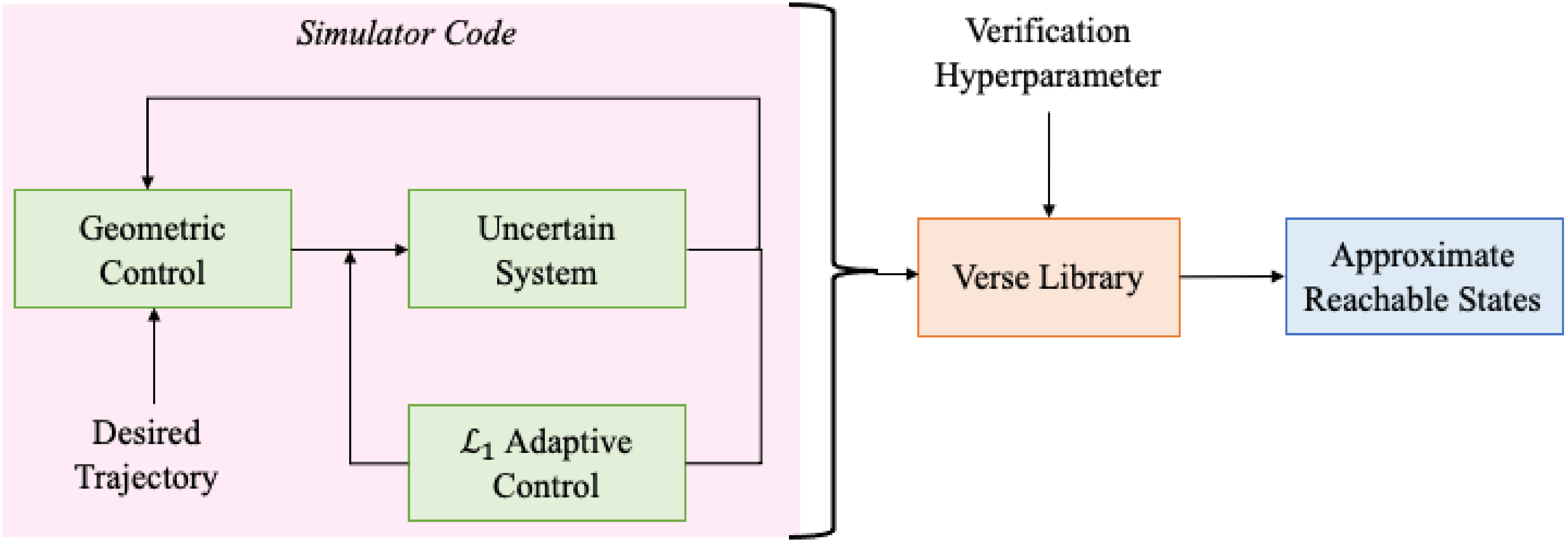






Application: Verification of L1AC

- L_1 Adaptive Control (L1AC) verification architecture using the Verse Library [Song et al. ICCPS-WIP 23]



L. Song, Y. Li, S. Cheng, P. Zhao, S. Mitra, N. Hovakimyan, Verification of $\mathcal{L}_1$ Adaptive Control using Verse Library: A Case Study of Quadrotors, *In Proceedings of* 13th IEEE International Conference on Cyber Physical Systems (ICCPS) Demo/Poster/Work-in-Progress, San Antonio, TX, 2023.

Application: Verification of L1AC

- L1AC verification objectives

  Formally verify the following two properties of L1AC:

  - Transient performance guarantees;
    - Scenario: an 18-dimensional drone model subject to rapidly changing uncertainty
    - Expected outcome: L1AC's capability for fast adaptation
  - Guaranteed delay margins.
    - Scenario: an 18-dimensional drone model subject to time delay in the control input
    - Expected outcomes:
      - L1AC preserves delay margin bounded away from zero;
      - Graceful performance degradation provided by L1AC.

Application: Verification of L1AC

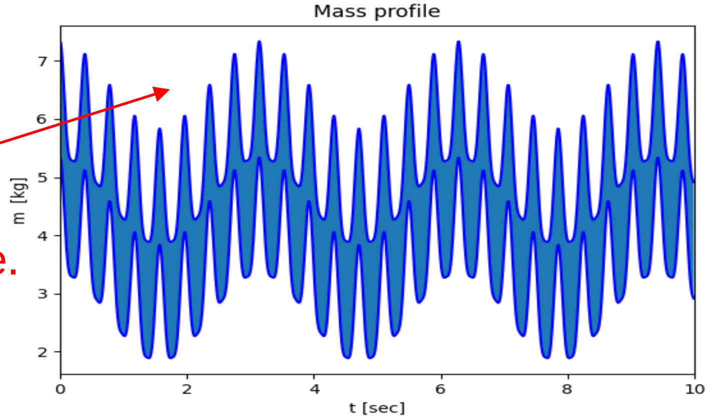
- L1AC verification objectives

  Formally verify the following two properties of L1AC:

  - Transient performance guarantees;
    - Scenario: an 18-dimensional drone model subject to rapidly changing uncertainty
    - Expected outcome: L1AC's capability for fast adaptation
  - Guaranteed delay margins.
    - Scenario: an 18-dimensional drone model subject to time delay in the control input
    - Expected outcomes:
      - L1AC preserves delay margin bounded away from zero;
      - Graceful performance degradation provided by L1AC.

Application: Verification of L1AC

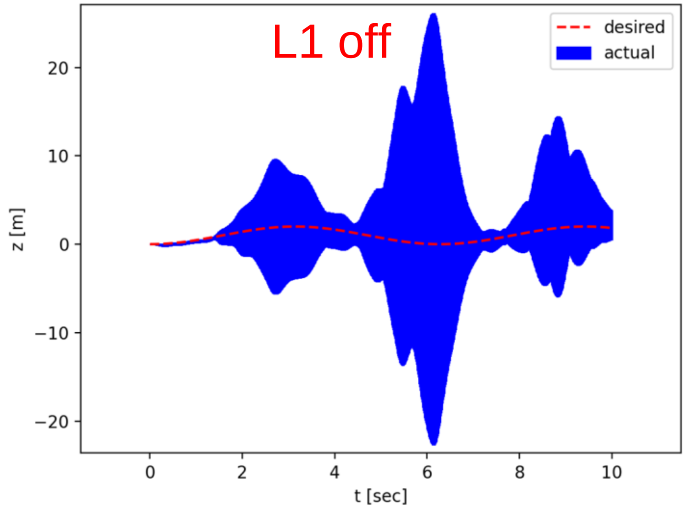
- Scenario 1: transient performance verification

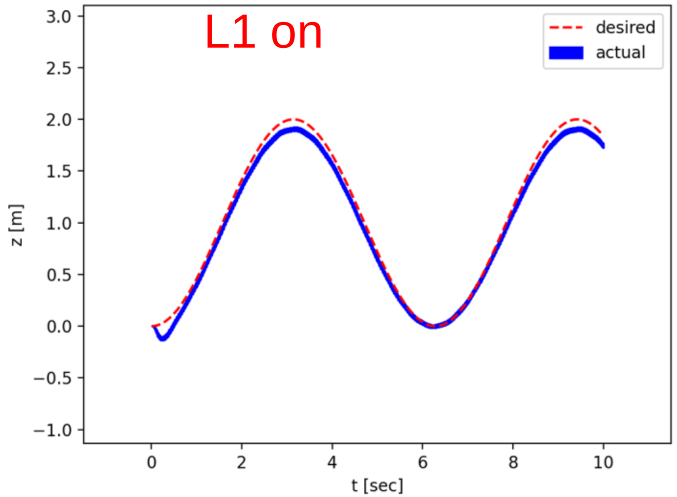


This is the source of uncertainty;

the uncertain parameter set also evolves with time.

The drone has **time-varying** mass parameter (unknown to the controller) with prescribed (time-varying) bounds.





Verification of L1AC capability for fast adaptation

Application: Verification of L1AC

- L1AC verification objectives

  Formally verify the following two properties of L1AC:

  - Transient performance guarantees;
    - Scenario: an 18-dimensional drone model subject to rapidly changing uncertainty
    - Expected outcome: L1AC's capability for fast adaptation
  - Guaranteed delay margins.
    - Scenario: an 18-dimensional drone model subject to time delay in the control input
    - Expected outcomes:
      - L1AC preserves delay margin bounded away from zero;
      - Graceful performance degradation provided by L1AC.

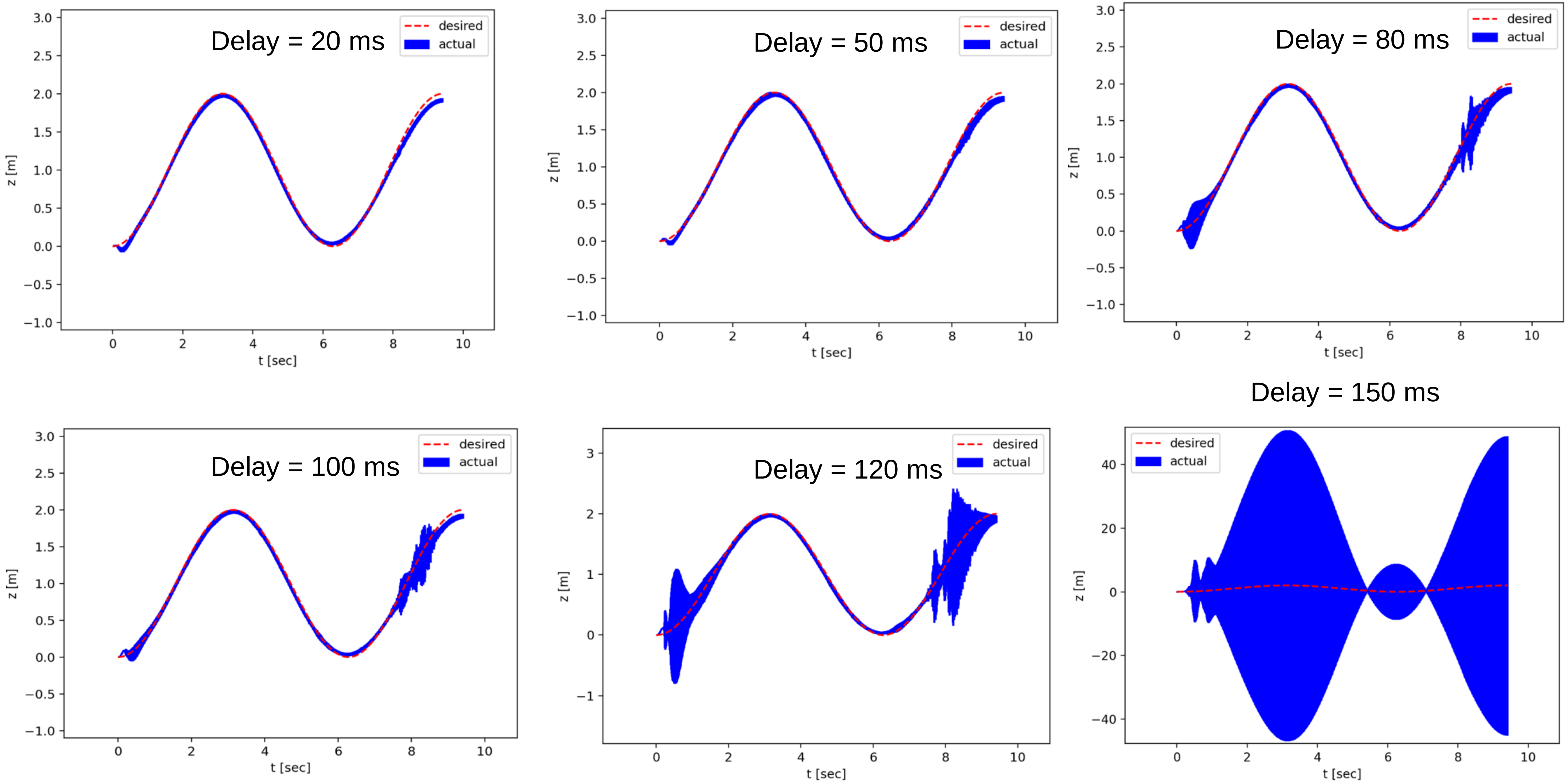
Application: Verification of L1AC

- Scenario 2: delay margin verification

Control input of the drone system is subject to **time delay**.

Only consider the delay margin achieved by L1AC, and we implement the verification procedure under **a range of** time delay amount.

Application: Verification of L1AC



Application: Verification of L1AC

- Future directions:
  - Verification of L1AC plus learning-enabled component;
    - An example: Contraction L1AC + Gaussian Processes [Gahlawat et al. L4DC 2021]
  - Verification of L1AC on systems involving switch, either on model or controller;
    - An example: learn-to-fly [Snyder et al. JGCD 2022], vehicle subject to driving environment changes [Mao et al. ACM TCPS 2023]
    - Tool/Method: deploy the mode switch feature of the Verse Library
  - Verification of the controller-parameter tuning process.
    - An example: Difftune^{+} [Cheng et al. L4DC 2023]
    - Tool/Method: Postdisc + Postcont -- 'one-step' reachability analysis feature of Verse

Other applications

- Application to DNN-based control [Puthumanaillam, Ornik, et al.]
- Application to RL-based air-traffic management [Peng Wei, GWU, ongoing]
- Parallel Verse [Zhu, et al.]

Summary

- Verse is designed to make hybrid system verification accessible
  - Python DL, nondeterministic agents, scenarios, sensors, asserts, OpenDrive maps
- Under the hood Verse uses tree-based reachability, sensitivity analysis for postCont
  - Can handle uncertainty in initial states, transitions, parameters
  - Plug-in Post computations DryVR, NeuReach, Monotonicity, ...
- In the future
  - Incremental verification, parallelization
  - verifying DRL controllers
- We welcome your feedback!

References

- Fan, C., Qi, B., Mitra, S., Viswanathan, M. (2017). DRYVR: Data-Driven Verification and Compositional Reasoning for Automotive Systems. In: Majumdar, R., Kunčak, V. (eds) Computer Aided Verification. CAV 2017
- Sun, D., Mitra, S. (2022). NeuReach: Learning Reachability Functions from Simulations. In: Fisman, D., Rosu, G. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2022
- Sibai, H., Li, Y., Mitra, S. (2021). SceneChecker: Boosting Scenario Verification Using Symmetry Abstractions. In: Silva, A., Leino, K.R.M. (eds) Computer Aided Verification. CAV 2021
- L. Song, Y. Li, S. Cheng, P. Zhao, S. Mitra, N. Hovakimyan, Verification of $\mathcal{L}_1$ Adaptive Control using Verse Library: A Case Study of Quadrotors, In Proceedings of 13th IEEE International Conference on Cyber Physical Systems (ICCPS) Demo/Poster/Work-in-Progress, San Antonio, TX, 2023.
- A. Gahlawat, A. Lakshmanan, L. Song, A. Patterson, Z. Wu, N. Hovakimyan, EA. Theodorou. Contraction L1-Adaptive Control using Gaussian Processes. In Learning for Dynamics and Control 2021 May 29 (pp. 1027-1040). PMLR.
- S. Snyder, P. Zhao, N. Hovakimyan. L1 Adaptive Control with Switched Reference Models: Application to Learn-to-Fly. Journal of Guidance, Control, and Dynamics. 2022 Dec;45(12):2229-42.
- Y. Mao, Y. Gu, N. Hovakimyan, L. Sha, P. Voulgaris. S$\mathcal{L}$1-Simplex: Safe Velocity Regulation of Self-Driving Vehicles in Dynamic and Unforeseen Environments. ACM Transactions on Cyber-Physical Systems. 2023 Feb 20;7(1):1-24.
- S. Cheng, L. Song, M. Kim, S. Wang, N. Hovakimyan. DiffTune^+ : Hyperparameter-Free Auto-Tuning using Auto-Differentiation. accepted to Learning for Dynamics and Control 2023 (to appear).

Thank You Very Much!