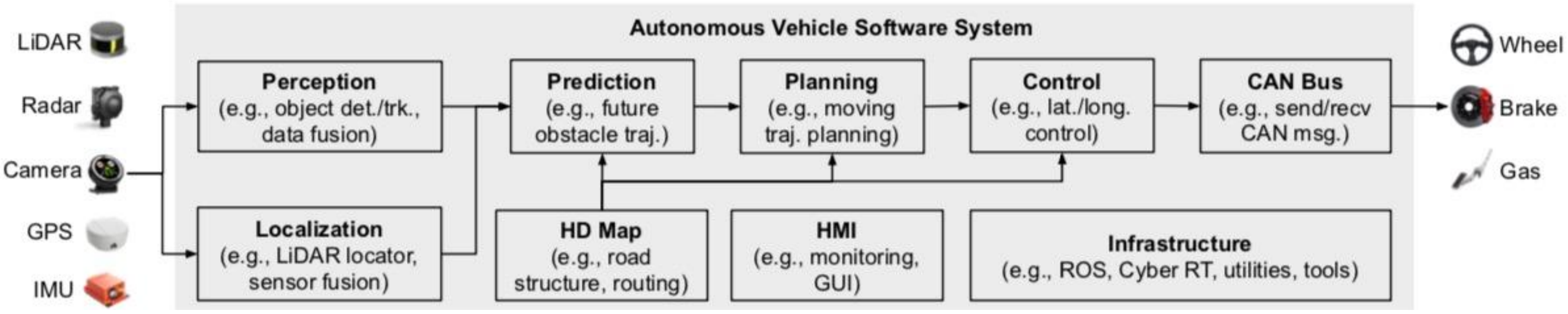
An Introduction to Testing Self-Driving Car

Ziyuan Zhong

LEVELS OF DRIVING AUTOMATION

0	1	2	3	4	5
NO AUTOMATION	**DRIVER ASSISTANCE**	**PARTIAL AUTOMATION**	**CONDITIONAL AUTOMATION**	**HIGH AUTOMATION**	**FULL AUTOMATION**
Manual control. The human performs all driving tasks (steering, acceleration, braking, etc.).	The vehicle features a single automated system (e.g. it monitors speed through cruise control).	ADAS. The vehicle can perform steering and acceleration. The human still monitors all tasks and can take control at any time.	Environmental detection capabilities. The vehicle can perform most driving tasks, but human override is still required.	The vehicle performs all driving tasks under specific circumstances. Geofencing is required. Human override is still an option.	The vehicle performs all driving tasks under all conditions. Zero human attention or interaction is required.
THE HUMAN MONITORS THE DRIVING ENVIRONMENT			THE AUTOMATED SYSTEM MONITORS THE DRIVING ENVIRONMENT		

How does a state-of-art autonomous vehicle (AV) system work?


LiDAR
Radar
Camera
GPS
IMU
Autonomous Vehicle Software System
Perception (e.g., object det./trk., data fusion)
Localization (e.g., LiDAR locator, sensor fusion)
Prediction (e.g., future obstacle traj.)
Planning (e.g., moving traj. planning)
Control (e.g., lat./long. control)
CAN Bus (e.g., send/recv CAN msg.)
HD Map (e.g., road structure, routing)
HMI (e.g., monitoring, GUI)
Infrastructure (e.g., ROS, Cyber RT, utilities, tools)
Wheel
Brake
Gas

Overview: topics will be covered in this talk

- study of autonomous vehicle bugs (Garcia et al., ICSE20)
- misbehavior prediction (Stocco et al., ICSE20)
- testing methods
  - a probabilistic programming language for testing (Fremont et al., PLDI19)
  - generating test case from police crush reports (Gambi et al., FSE19)
  - adversary test case generation
    - vision based system
      - changing weather/lighting conditions (classification) (Tian et al. ICSE18)
      - adding poster on billboard (classification) (Zhou et al., ICSE20)
      - adding poster on traffic sign (detection) (Zhao et al., CCS19)
      - adding pixel patch on vehicle back/road side (tracking) (Jia et al., ICLR20)
      - modifying behavior of pedestrian/vehicle (simulation) (Abeysirigoonawardena et al., ICRA19)
    - LiDAR based system (Cao et al., CCS19)

Overview: topics will not be covered in this talk

What are relevant in the literatures on self-driving cars but are not covered in this talk:

- vehicle/pedestrian re-identification
- pose estimation
- cooperative driving
- road extraction from GPS map
- global route planning
- semantic segmentation
- specific new algorithm/model on improving a particular component (detection, tracking, planning)
- pedestrian detections (also including pedestrian attributes recognition and path prediction)
- 3d object detection

Datasets/Simulator used in the papers

- LiDAR sensor datasets
- digital recorded vision datasets
  - Berkeley DeepDrive(100,000 HD video sequences)
  - Udacity self-driving car challenge dataset(101,396 images)
  - Dave testing dataset(45568 images)
  - Kitti(14999 images)
- physically self-recorded vision datasets
- simulators
  - GTAV
  - BeamNG.research (specialized in simulating traffic accidents)
  - Udacity self-driving car simulator
  - Carla

Study of AV bugs - Summary

A Comprehensive Study of Autonomous Vehicle Bugs, ICSE 2020, Garcia et al.

Motivation: we need to understand what are the existing bugs before we can test and repair them.

Summary: focus on L4 AV systems categorize bugs in open sourced self-driving car software system into 13 root causes, 20 bug symptoms, and 18 categories.

Methodology: closed and merged pull requests on Autoware and Baidu Apollo that fix defects before July.15, 2019

Study of AV bugs - Finding 1

- Incorrect algorithmic implementations and configurations are root causes for half of all bugs.

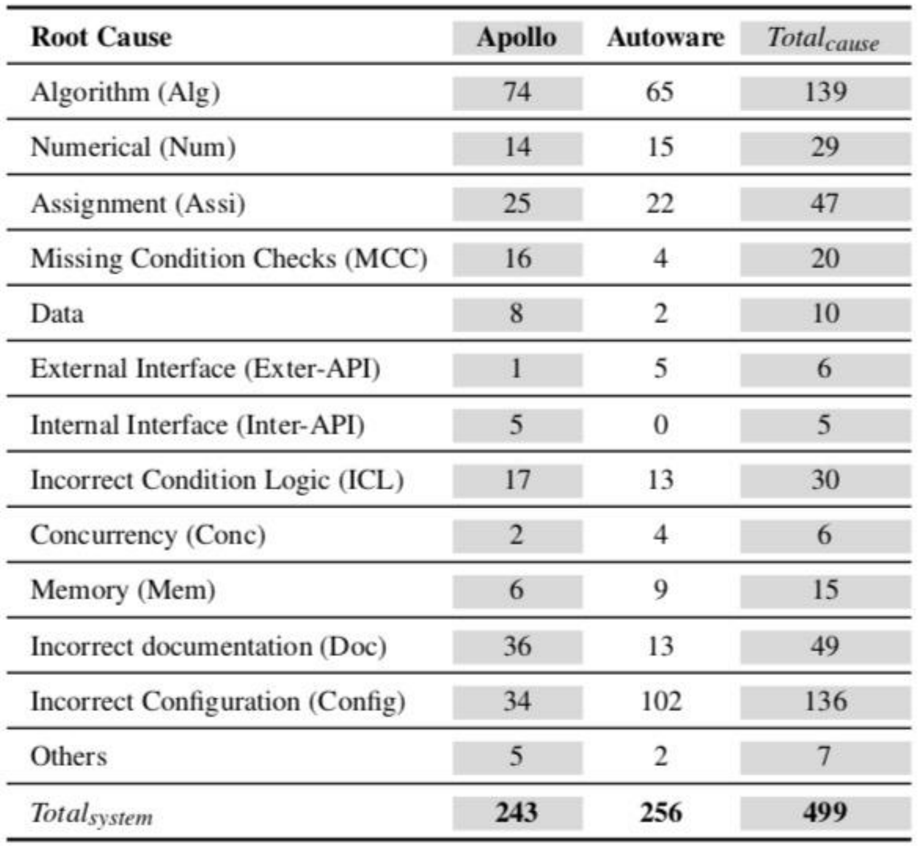


Table 4: Root Causes of Bugs in AV Systems

Root Cause	**Apollo**	**Autoware**	*$Total_{cause}$*
Algorithm (Alg)	74	65	139
Numerical (Num)	14	15	29
Assignment (Assi)	25	22	47
Missing Condition Checks (MCC)	16	4	20
Data	8	2	10
External Interface (Exter-API)	1	5	6
Internal Interface (Inter-API)	5	0	5
Incorrect Condition Logic (ICL)	17	13	30
Concurrency (Conc)	2	4	6
Memory (Mem)	6	9	15
Incorrect documentation (Doc)	36	13	49
Incorrect Configuration (Config)	34	102	136
Others	5	2	7
$Total_{system}$	**243**	**256**	**499**

Study of AV bugs - Finding 2

- 28.06% of bugs directly affect driving functionality of AVs with speed and velocity control, trajectory, and lane positioning and navigation occur the most frequently at 8.42%, 6.01%, and 5.01%, respectively.

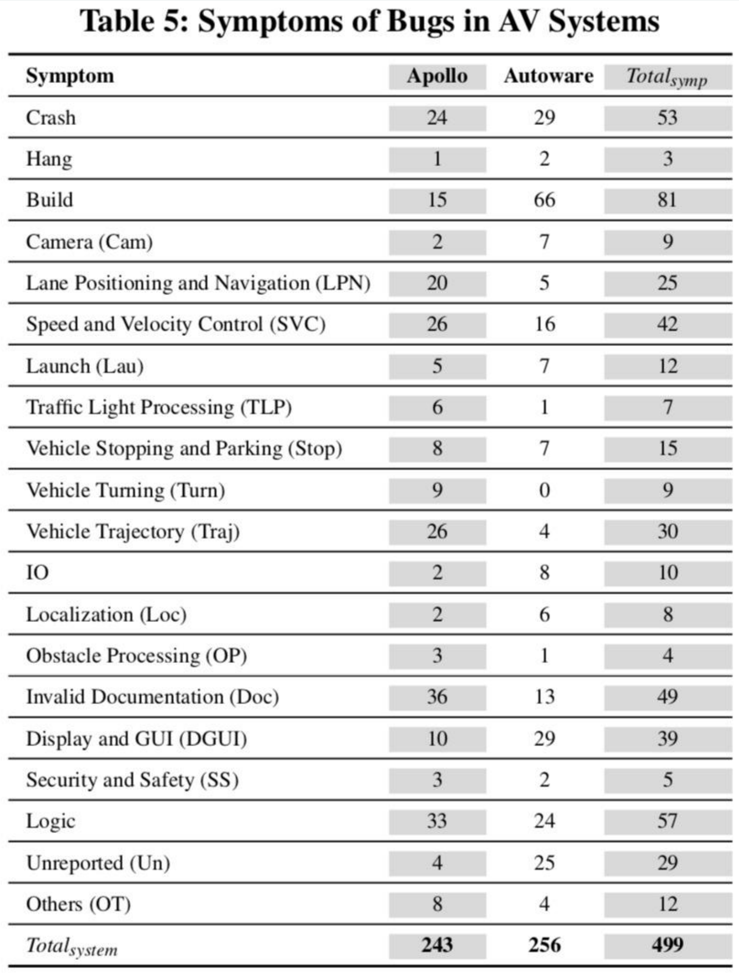


Table 5: Symptoms of Bugs in AV Systems

Symptom	**Apollo**	**Autoware**	$Total_{symp}$
Crash	24	29	53
Hang	1	2	3
Build	15	66	81
Camera (Cam)	2	7	9
Lane Positioning and Navigation (LPN)	20	5	25
Speed and Velocity Control (SVC)	26	16	42
Launch (Lau)	5	7	12
Traffic Light Processing (TLP)	6	1	7
Vehicle Stopping and Parking (Stop)	8	7	15
Vehicle Turning (Turn)	9	0	9
Vehicle Trajectory (Traj)	26	4	30
IO	2	8	10
Localization (Loc)	2	6	8
Obstacle Processing (OP)	3	1	4
Invalid Documentation (Doc)	36	13	49
Display and GUI (DGUI)	10	29	39
Security and Safety (SS)	3	2	5
Logic	33	24	57
Unreported (Un)	4	25	29
Others (OT)	8	4	12
$Total_{system}$	**243**	**256**	**499**

Study of AV bugs - Finding 3

- The core AV components with the greatest number of bugs across both systems are Planning, Perception, and Localization—ordered from most bug-ridden to least—with 135 (27.05%), 83 (16.63%), and 57 (11.42%) bugs, respectively.
- Planing constitutes 87/140 (62.14%) **driving bugs**

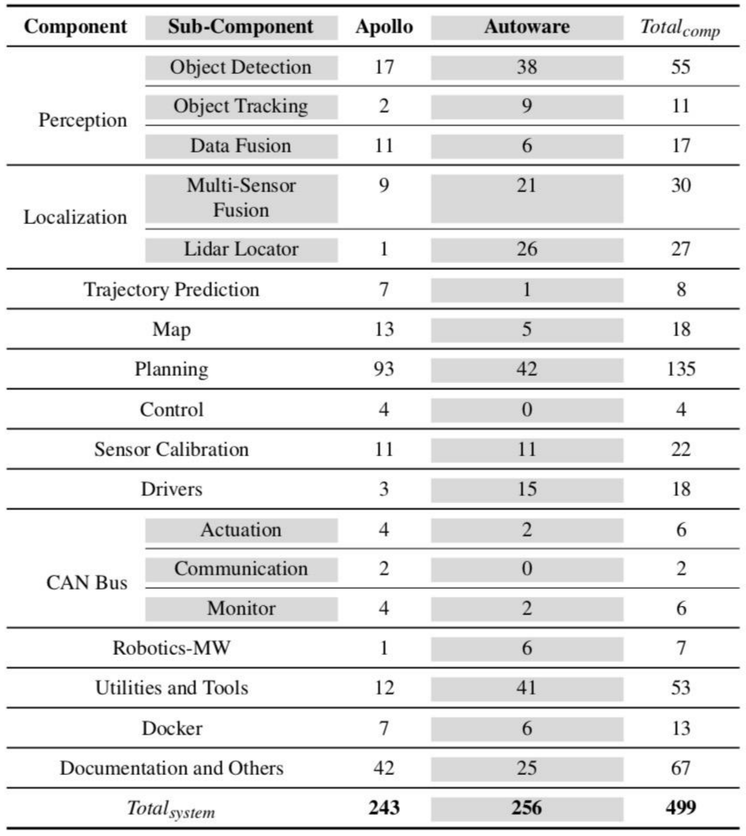


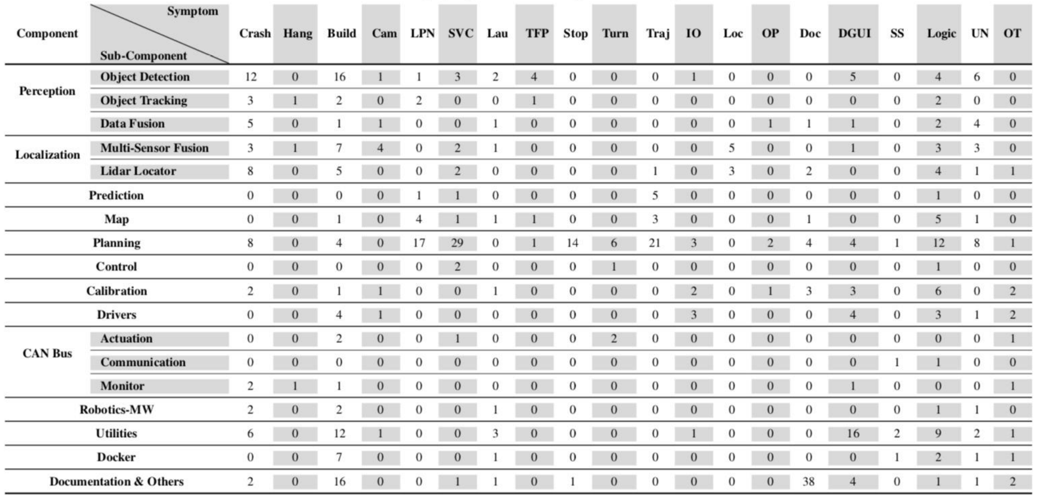
Table 7: Frequency of bug occurrences for each AV component

Component	Sub-Component	Apollo	Autoware	$Total_{comp}$
Perception	Object Detection	17	38	55
	Object Tracking	2	9	11
	Data Fusion	11	6	17
Localization	Multi-Sensor Fusion	9	21	30
	Lidar Locator	1	26	27
Trajectory Prediction		7	1	8
Map		13	5	18
Planning		93	42	135
Control		4	0	4
Sensor Calibration		11	11	22
Drivers		3	15	18
CAN Bus	Actuation	4	2	6
	Communication	2	0	2
	Monitor	4	2	6
Robotics-MW		1	6	7
Utilities and Tools		12	41	53
Docker		7	6	13
Documentation and Others		42	25	67
$Total_{system}$		**243**	**256**	**499**

Study of AV bugs - Finding 4

- Crash bugs occur throughout critical AV components—especially Perception, Localization, and Planning— making them susceptible to more dangerous secondary effects.

Table 8: Occurrences of bug symptoms in components of Apollo and Autoware



Component	Symptom / Sub-Component	Crash	Hang	Build	Cam	LPN	SVC	Lau	TFP	Stop	Turn	Traj	IO	Loc	OP	Doc	DGUI	SS	Logic	UN	OT
Perception	Object Detection	12	0	16	1	1	3	2	4	0	0	0	1	0	0	0	5	0	4	6	0
	Object Tracking	3	1	2	0	2	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0
	Data Fusion	5	0	1	1	0	0	1	0	0	0	0	0	0	1	1	1	0	2	4	0
Localization	Multi-Sensor Fusion	3	1	7	4	0	2	1	0	0	0	0	0	5	0	0	1	0	3	3	0
	Lidar Locator	8	0	5	0	0	2	0	0	0	0	1	0	3	0	2	0	0	4	1	1
Prediction		0	0	0	0	1	1	0	0	0	0	5	0	0	0	0	0	0	1	0	0
Map		0	0	1	0	4	1	1	1	0	0	3	0	0	0	1	0	0	5	1	0
Planning		8	0	4	0	17	29	0	1	14	6	21	3	0	2	4	4	1	12	8	1
Control		0	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	0	1	0	0
Calibration		2	0	1	1	0	0	1	0	0	0	0	2	0	1	3	3	0	6	0	2
Drivers		0	0	4	1	0	0	0	0	0	0	0	3	0	0	0	4	0	3	1	2
CAN Bus	Actuation	0	0	2	0	0	1	0	0	0	2	0	0	0	0	0	0	0	0	0	1
	Communication	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
	Monitor	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
Robotics-MW		2	0	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0
Utilities		6	0	12	1	0	0	3	0	0	0	0	1	0	0	0	16	2	9	2	1
Docker		0	0	7	0	0	0	1	0	0	0	0	0	0	0	0	0	1	2	1	1
Documentation & Others		2	0	16	0	0	1	1	0	1	0	0	0	0	0	38	4	0	1	1	2

Survey of auto bugs - Discussion

- Crash can be caused by many components.
- Planning and Localization cause many bugs but are less well-studied.
- Even in Perception, two sub-components (object tracking and data fusion) cause bugs and are less well-studied in terms of testing and analysis.

Misbehaviour Prediction for Autonomous Driving Systems - Summary

Misbehaviour Prediction for Autonomous Driving Systems, Stocco et al., ICSE20

Motivation: alerting a driver potentially dangerous situations

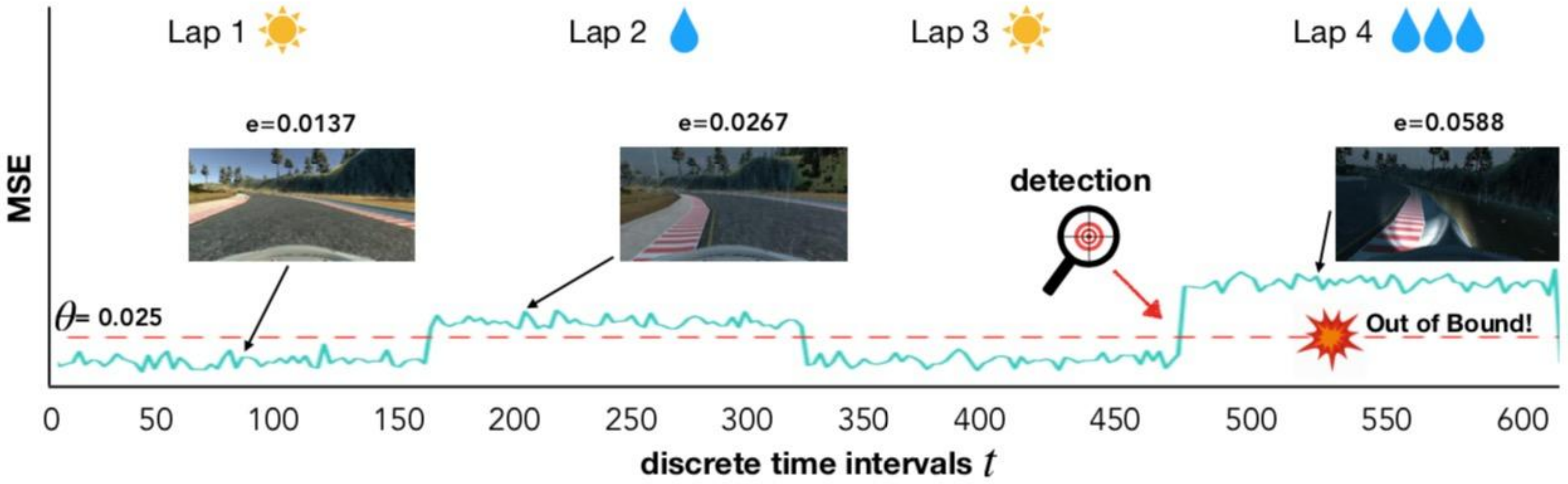
Dataset: dataset(images) are collected from manually controlling a car to lap in the Udacity simulator(with the addition of a weather generator and a collision detector)

Model: CNN-based (Dave-2, Chauffer)

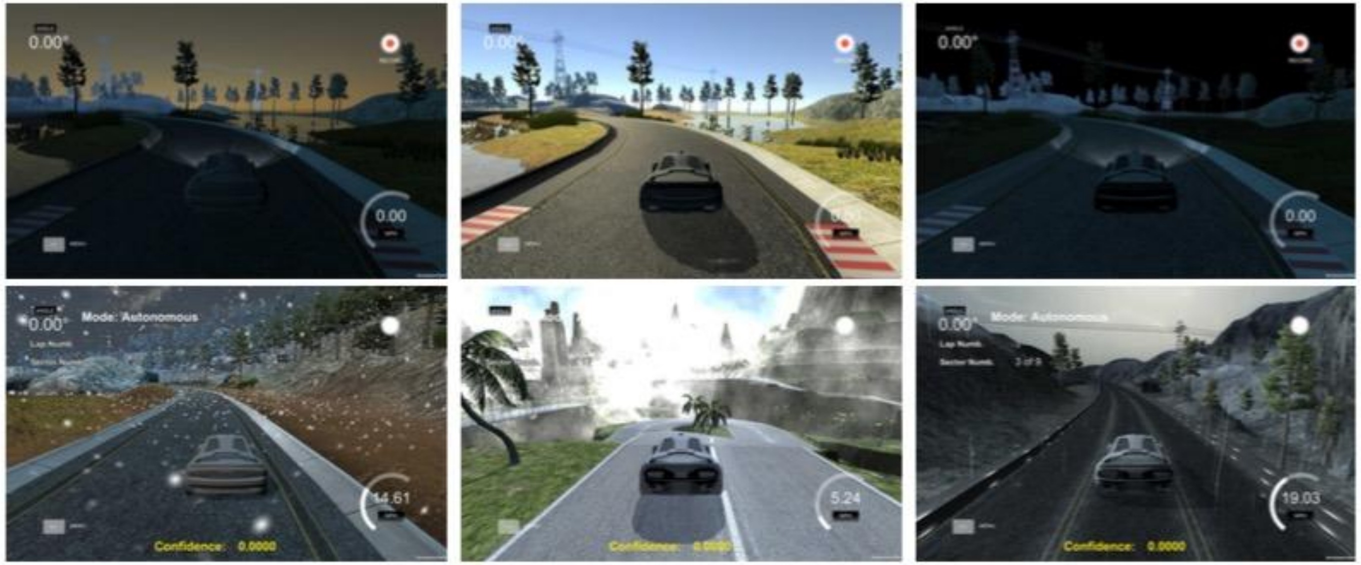
Objective: correctly alerting a driver during anomaly window for a period that has misbehavior

Definition of misbehavior: crush or out of road

Misbehaviour Prediction for Autonomous Driving Systems - Illustration



Misbehaviour Prediction for Autonomous Driving Systems - Scenarios



Misbehaviour Prediction for Autonomous Driving Systems - Methodology (Training)

reconstructor: CNN (single-image based), RNN (sequence based), reconstruction error as the loss function

---> fit probability distribution (one pixel ~ mean 0 gaussian

---> sum ~ gamma) on reconstruction errors via MLE **--->** choose a threshold using inverse CDF .

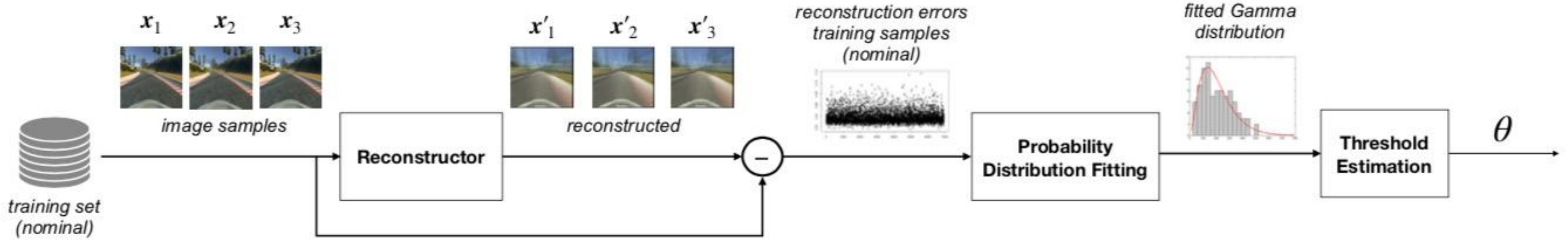


Figure 2: Model Training under Nominal Driving Behaviour.

Misbehaviour Prediction for Autonomous Driving Systems - Methodology (Inference)

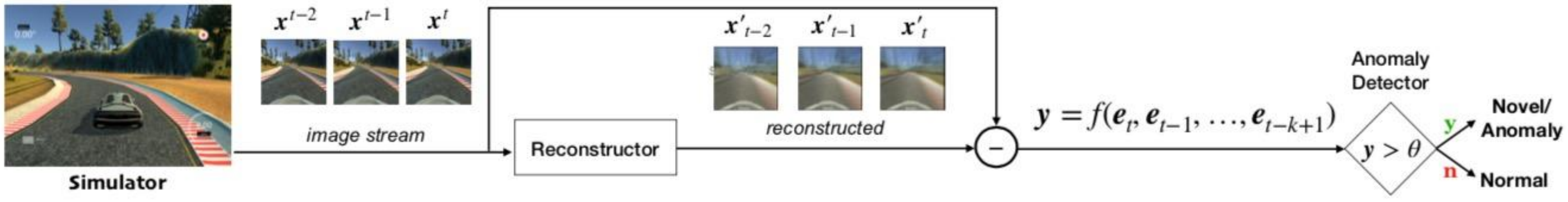


Figure 4: Usage Scenario of SELFORACLE.

Misbehaviour Prediction for Autonomous Driving Systems - Discussion

- The model is trained with no negative example in the training set. So there is no rare events in the training set to help the system to learn how to avoid such situations

- The testing setup has very simple background and is thus a bit limited.

Generating Effective Test Cases for Self-Driving Cars from Police Reports - Summary

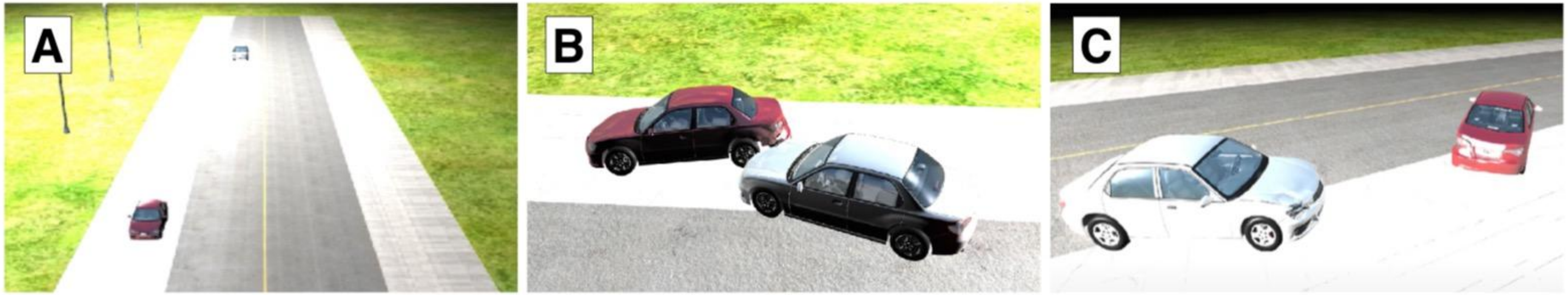
Generating Effective Test Cases for Self-Driving Cars from Police Reports, Gambi et al., FSE18

Motivation: reconstruct car crash scenario to test vision-based self-driving car system.

Dataset: generated using BeamNG.research (frames)

Model: DeepDriving (CNN to predict several driving affordance indicators from the images -> rule-based driving controller which computes the driving actions.)

Generating Effective Test Cases for Self-Driving Cars from Police Reports - Example

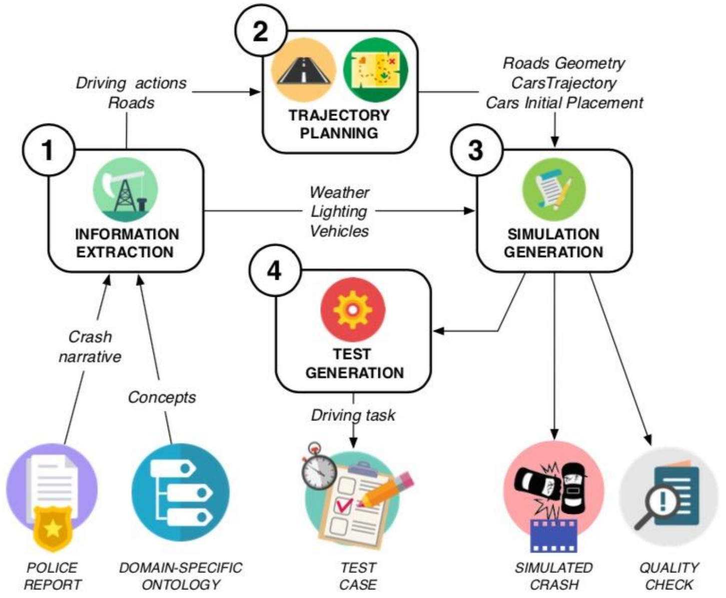


“The crash occurred on a two-way, two-lane straight, level, bituminous residential street with a speed limit of 25 mph. Conditions at the time of the weeknight crash were cloudy, dark, and dry. V1, a 2001 Kia Sephia, driven by a 28 year-old female, was driving southbound on the road when it left the travel lane and the front of V1 struck the back of a legally parked, unoccupied vehicle on the right side of the road.”

Generating Effective Test Cases for Self-Driving Cars from Police Reports - Methodology

Overview and Information Extraction

- extract grammatical dependency
- match words with domain-specific ontology at different levels:
    - top: environment_property, car crashes storyline
    - middle: traffic_participant, road, weather
    - bottom: vehicle_type, pavement_material, lighting, etc.
- grammatical dependency to extract road quantitative properties
- extracting vehicle properties

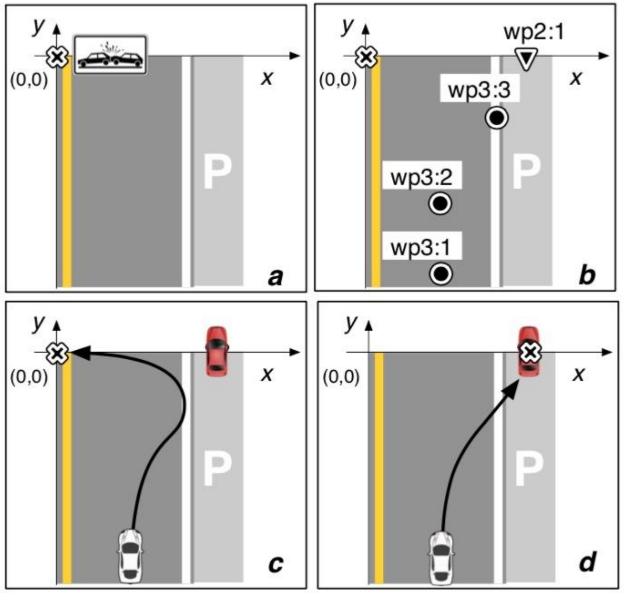


Generating Effective Test Cases for Self-Driving Cars from Police Reports - Methodology

Trajectory Planning and Simulation Generation

- place the impact point on the origin
- continue backwards by placing a new waypoint for each of the driving actions.
- for each driving action, use basic trigonometry and a simplified kinematics model to compute each segment of the trajectory.
- adjusts the position of the impact point and updates the road geometry.

generates code using a customized, template-based approach



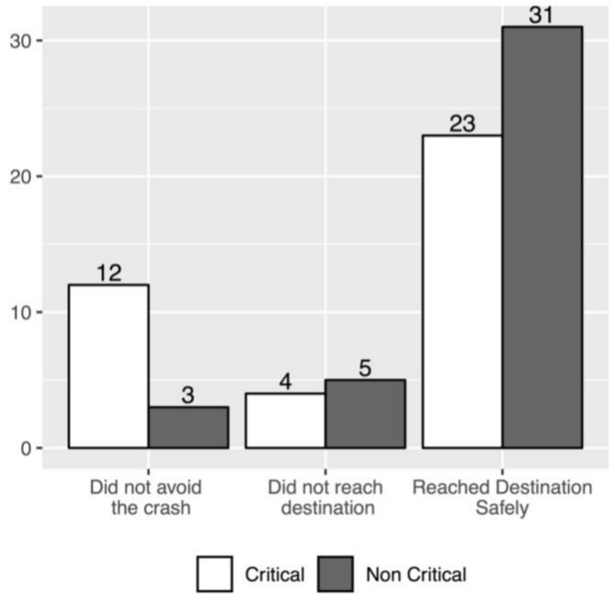
Generating Effective Test Cases for Self-Driving Cars from Police Reports - Results and Discussions

Manually inspected all the failed tests in order to check if they correspond to actual problems or false positives

Findings: in eight cases DeepDriving failed to avoid preventable crashes

Conclusion: AC3R can put the test subject in the more hazardous situations. AC3R can find the failure of CNN and the failure of controller.

Limitation: the number of cases in the evaluation seem to be limited.



ADVERSARIAL ATTACK AGAINST MOT - Summary

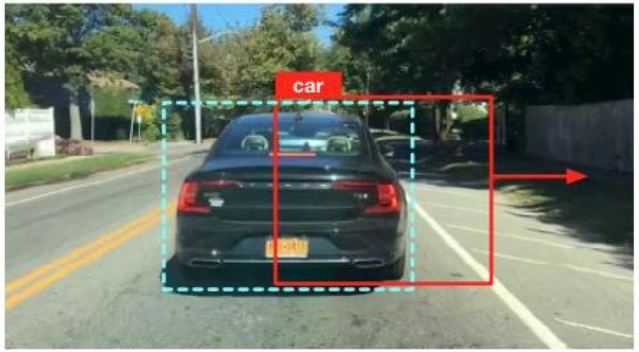
FOOLING DETECTION ALONE IS NOT ENOUGH: ADVERSARIAL ATTACK AGAINST MULTIPLE OBJECT TRACKING, Jia et al., ICLR20

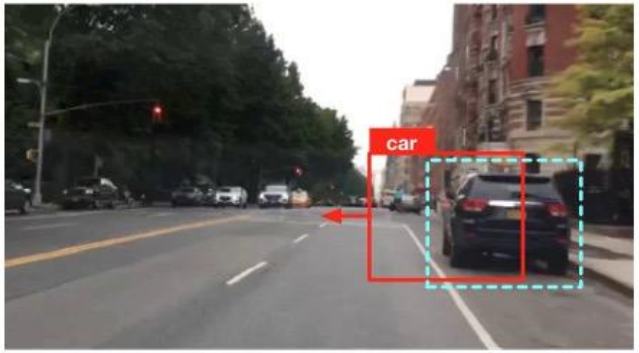
datasets: 100,000 HD video sequences of over 1,100-hour driving experience across many different times in the day, weather conditions, and driving scenarios.

motivation: No previous work attacking beyond detection/classification for self-driving car. A success rate of over 98% is needed for them to actually affect the tracking results.

They can carefully design AEs to attack the tracking error reduction process in MOT to deviate the tracking results of existing objects towards an attacker-desired moving direction.

ADVERSARIAL ATTACK AGAINST MOT - Illustration of tracker hijacking





How MOT and attacking MOT work

detection -> bipartite matching detected objects and trackers (using spatial cost)

- a new tracker is created when the object has been constantly detected for a certain number of frames (H=0.2*fps)
- a tracker will be deleted if no object is associated with for a duration of R frames (R=2*fps)

physical MOT attack (essentially attacking a 1st order Kalman Filter(update a state based on previous state and new observation)):

- move the detection box to desired direction but keep its match with the tracker in the previous round.
- projected gradient ascent on removing original bounding box and having the wanted bounding box on the targeted patch area.

ADVERSARIAL ATTACK AGAINST MOT - Discussion

- Limitation 1: Dataset is tiny (10 clips for move-in and move-out).

- Limitation 2: No physical attack experiments have been conducted.

Attack Lidar - Pipeline

Adversarial Sensor Attack on LiDAR-based Perception in Autonomous Driving, Cao et al., CCS19

Dataset: sampled 300 3D point cloud frames from the real-world LiDAR sensor data trace.

Model: Baidu Apollo 3.0 LiDAR perception module

Summary: spoof partial lidar information by laser to attack an AV

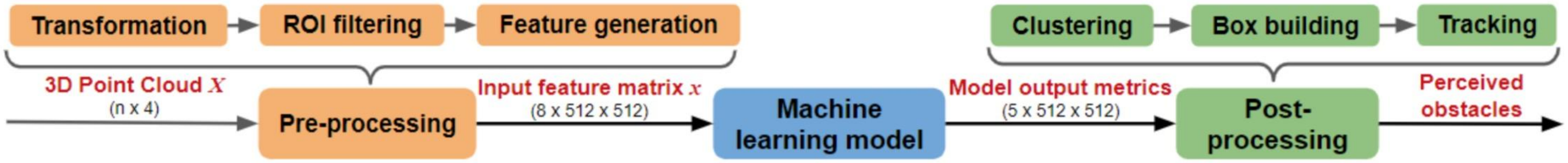
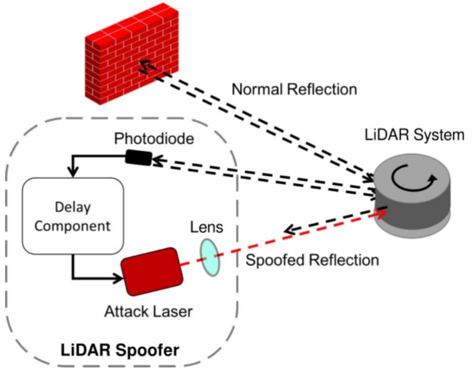
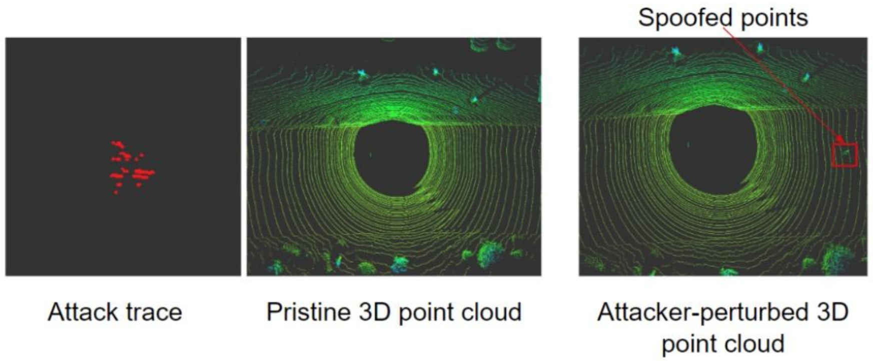
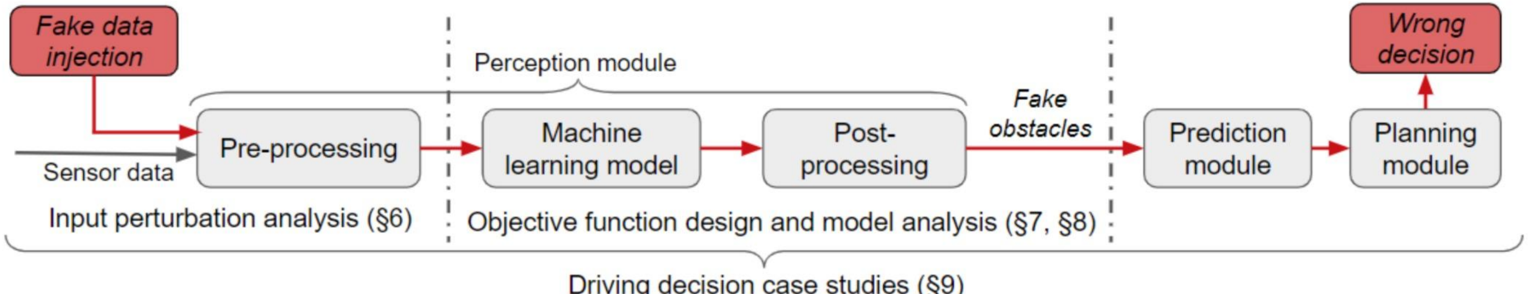


Figure 1: Overview of the data processing pipeline for LiDAR-based perception in Baidu Apollo.

Attack Lidar - Methodology 1







Attack Lidar - Methodology 2

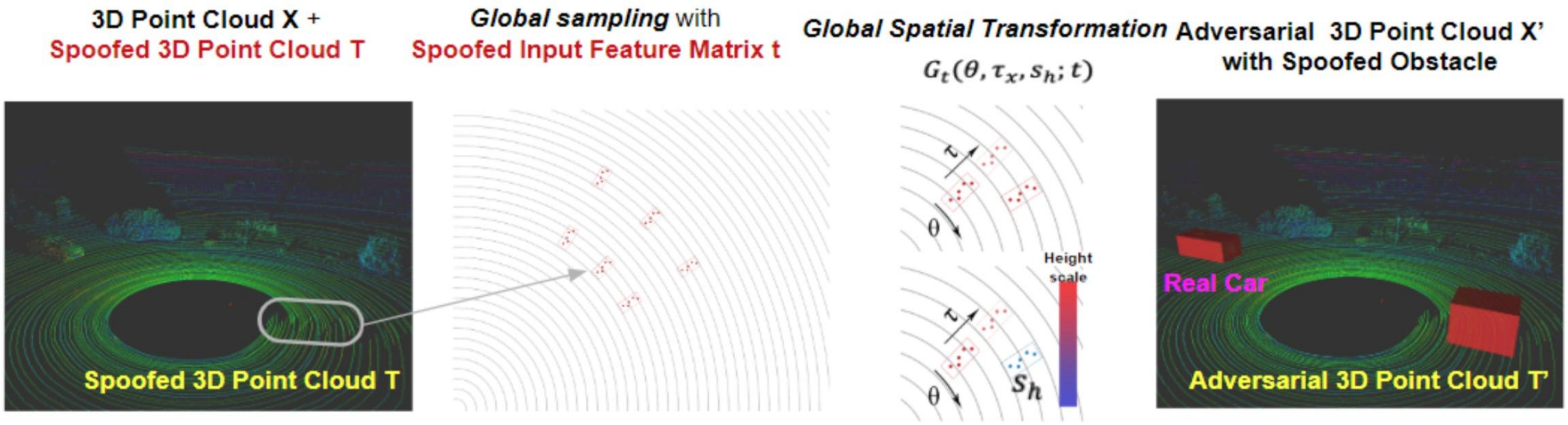
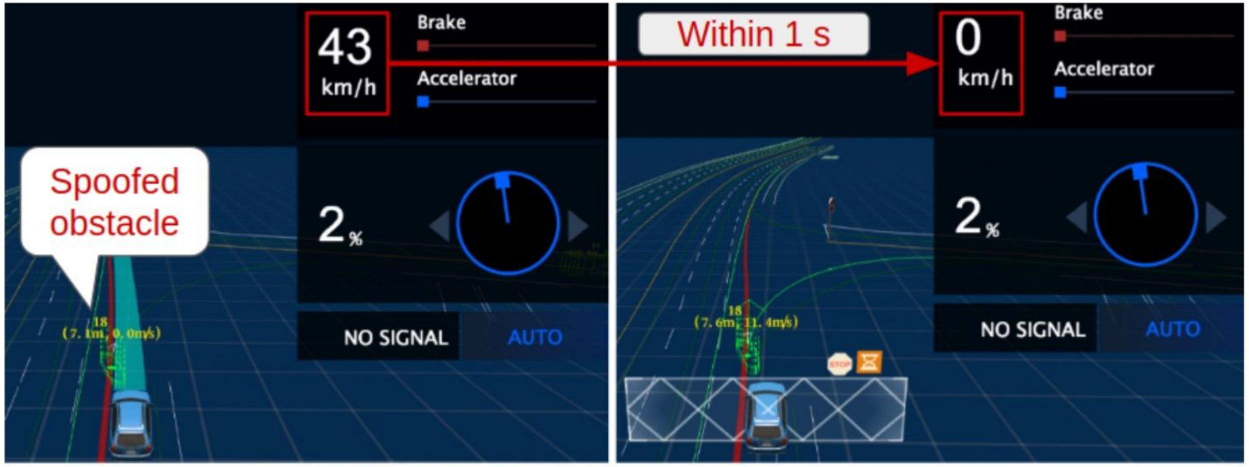
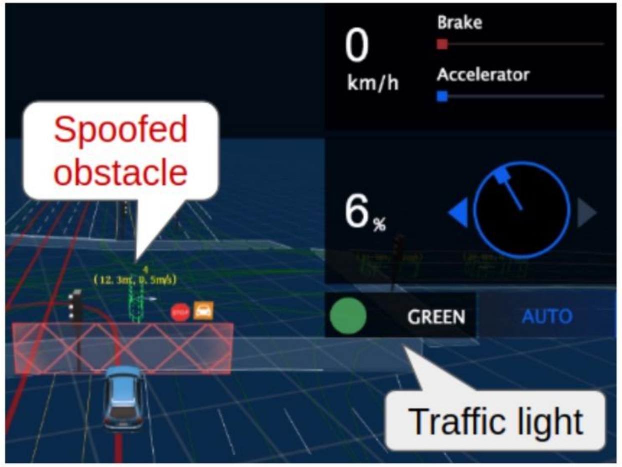

3D Point Cloud X +
Spoofed 3D Point Cloud T
Global sampling with
Spoofed Input Feature Matrix t
Global Spatial Transformation
$G_t(\theta, \tau_x, s_h; t)$
Adversarial 3D Point Cloud X'
with Spoofed Obstacle
Spoofed 3D Point Cloud T
τ
θ
Height scale
s_h
Real Car
Adversarial 3D Point Cloud T'

Figure 8: Overview of the adversarial example generation process.

Attack Lidar - Case Studys



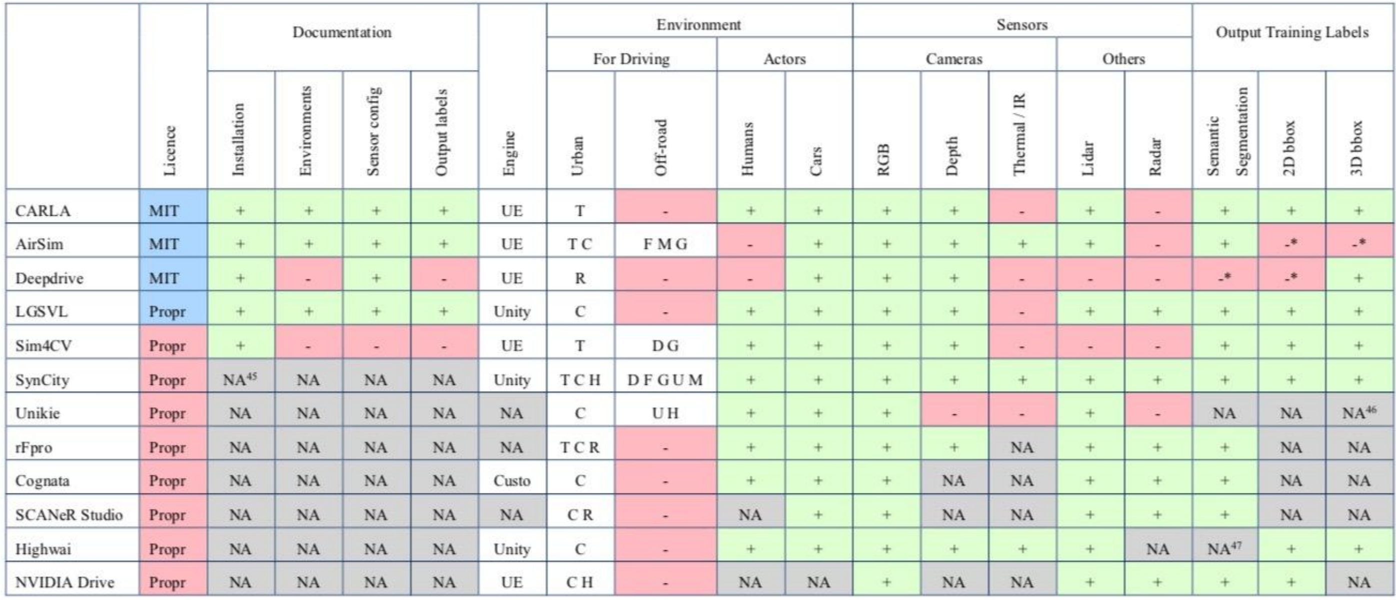


Attack Lidar - Discussion

- Limitation:
  - performing their attack on a real AV on the road requires dynamically aiming an attack device at the LiDAR on a victim car with high precision, which is difficult to prove the feasibility without road tests in the physical world
  - did not perform a comprehensive analysis on modules beyond the perception module

All popular simulators for AV

The table was made in 2019Q1. Carla now supports Radar. Both Carla and LGSVL support AutoWare. Only LGSVL supports Baidu Apollo (However, Apollo does not support Nvidia Turing Architecture (RTA20XX series and V series)).



		Documentation					Environment				Sensors					Output Training Labels		
							For Driving		Actors		Cameras			Others				
	Licence	Installation	Environments	Sensor config	Output labels	Engine	Urban	Off-road	Humans	Cars	RGB	Depth	Thermal / IR	Lidar	Radar	Semantic Segmentation	2D bbox	3D bbox
CARLA	MIT	+	+	+	+	UE	T	-	+	+	+	+	-	+	-	+	+	+
AirSim	MIT	+	+	+	+	UE	T C	F M G	-	+	+	+	+	+	-	+	-*	-*
Deepdrive	MIT	+	-	+	-	UE	R	-	-	+	+	+	-	-	-	-*	-*	+
LGSVL	Propr	+	+	+	+	Unity	C	-	+	+	+	+	-	+	+	+	+	+
Sim4CV	Propr	+	-	-	-	UE	T	D G	+	+	+	+	-	-	-	+	+	+
SynCity	Propr	NA[45]	NA	NA	NA	Unity	T C H	D F G U M	+	+	+	+	+	+	+	+	+	+
Unikie	Propr	NA	NA	NA	NA	NA	C	U H	+	+	+	-	-	+	-	NA	NA	NA[46]
rFpro	Propr	NA	NA	NA	NA	NA	T C R	-	+	+	+	+	NA	+	+	+	NA	NA
Cognata	Propr	NA	NA	NA	NA	Custo	C	-	+	+	+	NA	NA	+	+	+	NA	NA
SCANeR Studio	Propr	NA	NA	NA	NA	NA	C R	-	NA	+	+	NA	NA	+	+	+	NA	NA
Highwai	Propr	NA	NA	NA	NA	Unity	C	-	+	+	+	+	+	+	NA	NA[47]	+	+
NVIDIA Drive	Propr	NA	NA	NA	NA	UE	C H	-	NA	NA	+	NA	NA	+	+	+	+	NA

Figure 29.

Table legend - Urban and Off-road: Urban: T - town, C - city, R - road track, H - highway. Off-road: F - forest, D - desert, M - mountains, G - grassy field, U - underground mine, H - harbor

Licence: blue – open source, red – not open (commercial)

Discussion - Steps to Improve an AV System

1. Detection: finding errors/weaknesses of existing self-driving car systems.
2. Analysis: understanding the causes of such errors/weaknesses.
3. Fixing: developing a method to improve existing systems to fix such errors/weaknesses.

Discussion - Potential Directions

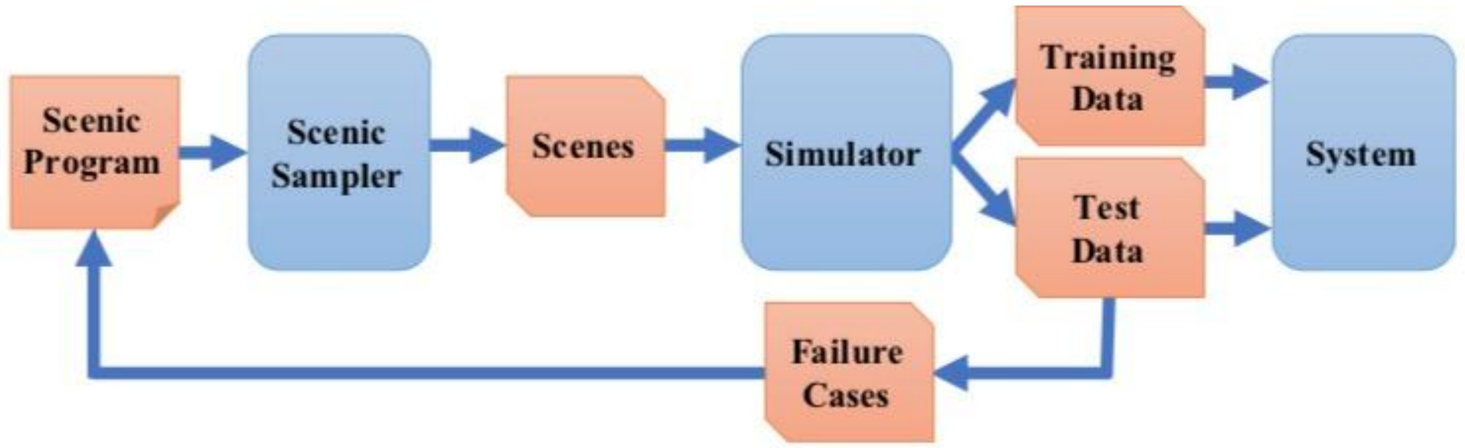
- detection of misbehavior and testing in more settings in a scalable manner
    - more realistic settings
        - image/video analysis -> 3d simulations -> physical setting
        - artificial attack(adding patches) -> more realistic attack (the behavior of cars/pedestrians, dysfunction of some units)
        - limited settings (limited number of objects) -> more complex settings (cars/people/background)
    - more components
        - perception -> other components (e.g. localization, planning)
            - camera -> LiDAR -> others (map, radar)
                - detection -> tracking
        - a single component -> fusion
- Understanding the causes of AV errors
- (offline/online) fixing detected errors through targeted augmenting or human in loop
- Understanding the interaction between the learning models and traditional software system

Appendix

SCENIC - Summary

motivation: a probabilistic programming language that makes testing self-driving car systems easier.

interface: GTAV **task:** detection **dataset:** generated by driving laps in GTAV **model:** squeezeDet

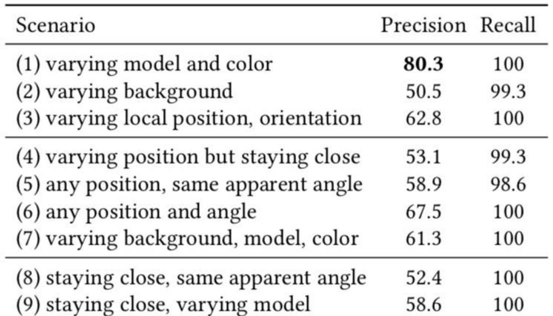


SCENIC - Experiments

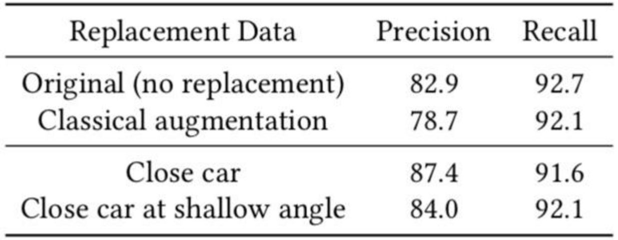
testing under different conditions: 1000*4 for training and 50*4 for testing

training on rare events (occlusion): Replace 250 images in “Driving in the Matrix” synthetic dataset with generated occlusion images and produces better performance on 200 generated occlusion images and comparable results on the sampled testing set from “Driving in the Matrix” synthetic dataset.

debugging failures: change different features in the failure scene, and then redo the training by targeting data according to the most influential features found. In this way, a better result has been achieved.



Scenario	Precision	Recall
(1) varying model and color	**80.3**	100
(2) varying background	50.5	99.3
(3) varying local position, orientation	62.8	100
(4) varying position but staying close	53.1	99.3
(5) any position, same apparent angle	58.9	98.6
(6) any position and angle	67.5	100
(7) varying background, model, color	61.3	100
(8) staying close, same apparent angle	52.4	100
(9) staying close, varying model	58.6	100



Replacement Data	Precision	Recall
Original (no replacement)	82.9	92.7
Classical augmentation	78.7	92.1
Close car	87.4	91.6
Close car at shallow angle	84.0	92.1

SCENIC - Discussion

- **Other applications:**
    - integrate with VERIFAI to generate seed inputs for temporal-logic falsification of an automated collision-avoidance system.
    - interface to X-plane flight simulator and CARLA
- SCENIC seems a bit limited to object detection/classification since it cannot be used to generate continuous frames? the detection step identifies the objects in the images and the tracking step links the objects to the trajectories (i.e., trackers)

DeepBillboard - Summary

Summary: Physical attack by adding poster to billboard along the road.

Task: prediction of steering angle

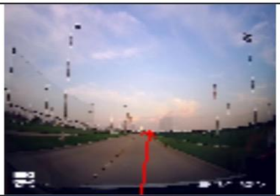
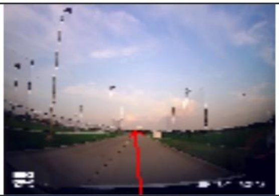
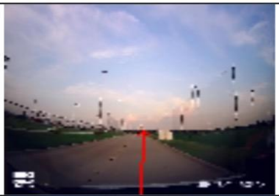
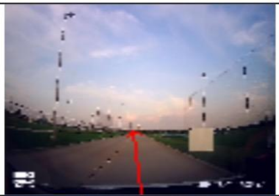
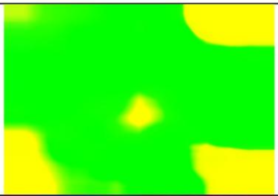
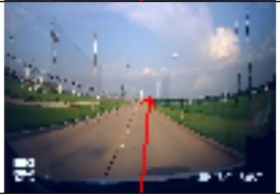
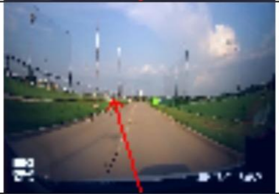
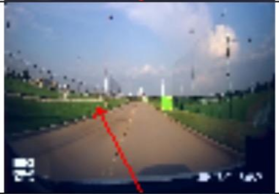
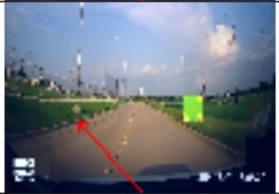
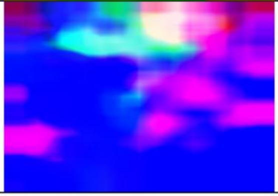
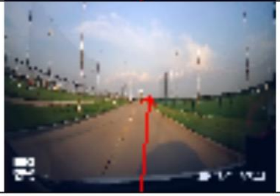
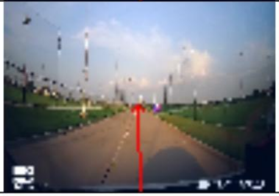
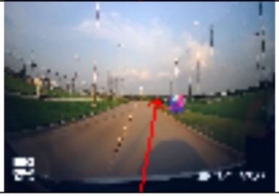
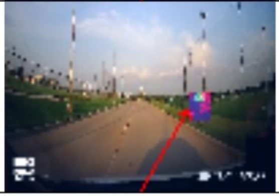
Models: DAVE_v1, DAVE_v2, DAVE_v3, EPOCH

Dataset:

- Digital: Udacity self-driving car challenge dataset(101,396), Dave testing dataset(45568), Kitti(14999)
- Physical: collect image frames by driving through a billboard (3 weather * 2 direction cases), generate adversarial poster using the images, and then finally paste it.

DeepBillboard - Illustration

Table 3: Illustration of physical billboard perturbation.

	Perturbation	100’	60’	20’	10’
White	N/A				
Adversarial (left)					
Adversarial (right)					

DeepBillboard - Methodology

Loss: average angle errors (+ perturbation printability)

Algorithm: Given a series of frames for passing a billboard, one-step gradient ascent to perturb k most impactful pixels for each frame (when overlap, taking the one that gives higher loss) and combine them.

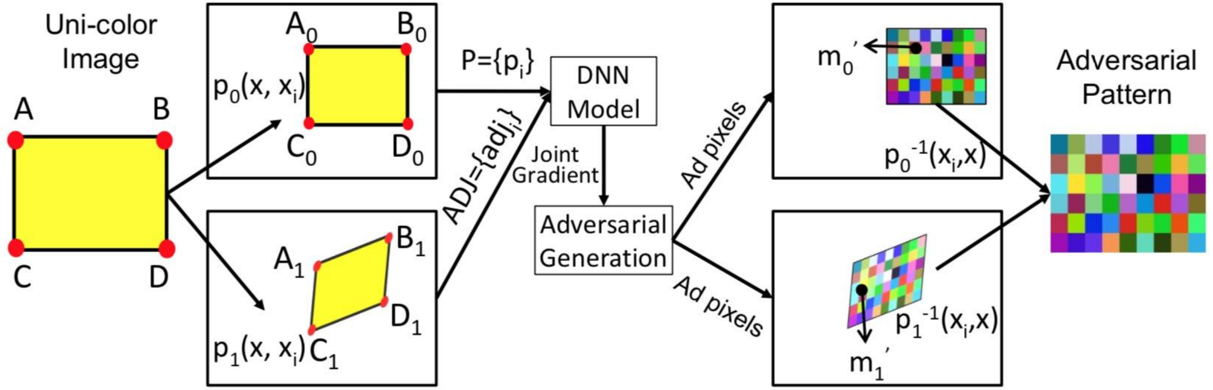


Figure 2: Work flow of DeepBillboard to generate adversarial perturbations for contiguous frames, where p_i represents the i_{th} frame.

DeepBillboard - Discussion

- The task is similar to DeepTest but have physical experiment and threat model becomes adding a poster to a billboard rather than adding effects on the whole image.
- Limitation: need to create customized adversarial poster for each billboard and for each specific model.