Automated Testing of Autonomous Driving Assistance Systems

Lionel Briand

Vector Testing Symposium, Stuttgart, 2018
SnT Centre

Top level research in Information & Communication Technologies

Created to fuel the national innovation system
Collaborative Research @ SnT

- Research in industrial context
- Addresses actual needs
- Well-defined problem
- Long-term collaborations
- Our lab is the industry
Strategic Research Areas

- Secure and Compliant Data Management
- FinTech
- Cybersecurity
- Space Systems and Resources
- Autonomous Vehicles
- Internet of Things
Introduction
Cyber-Physical Systems

- A system of collaborating computational elements controlling physical entities
Advanced Driver Assistance Systems (ADAS)

- Automated Emergency Braking (AEB)
- Pedestrian Protection (PP)
- Lane Departure Warning (LDW)
- Traffic Sign Recognition (TSR)
Advanced Driver Assistance Systems (ADAS)

Decisions are made over time based on sensor data

Diagram showing the interaction between sensors, actuators, controller, and environment in the context of ADAS.
A General and Fundamental Shift

• Increasingly so, it is easier to learn behavior from data using machine learning, rather than specify and code

• **Example: Neural networks (deep learning)**

• Millions of weights learned

• No explicit code, no specifications

• **Verification, testing?**
Testing Implications

- **Test oracles/verdicts?** No explicit, expected test behavior
- **Test completeness?** No source code, no specification
CPS Development Process

Model in the Loop

- Function modeling (Matlab/Simulink)
  - Controller
  - Plant/Environment

Software in the Loop

- Architecture modeling (SysML/C-Code)
  - Real-time analysis
  - Integration

Hardware in the Loop

- Deployment (embedded-C)
  - Testing (Expensive)
Opportunities and Challenges

• Early functional models (MiL) offer opportunities for early functional verification and testing

• But a challenge for constraint solvers and model checkers:
  • Continuous mathematical models, e.g., differential equations
  • Discrete software models for code generation, but with complex operations
  • Library functions in binary code
Automotive Environment

- Highly varied environments, e.g., road topology, weather, building and pedestrians …

- Huge number of possible scenarios, e.g., determined by trajectories of pedestrians and cars

- ADAS play an increasingly critical role

- A challenge for testing
Testing Advanced Driver Assistance Systems
Objective

- Testing ADAS
  - Identify and characterize most critical/risky scenarios
  - Test oracle: Safety properties
  - Need scalable test strategy due to large input space
Automated Emergency Braking System (AEB)

Objects’ position/speed → Sensor → Vision (Camera) → Decision making → Brake Controller

“Brake-request” when braking is needed to avoid collisions
Example Critical Situation

“AEB detects a pedestrian in front of the car with a high degree of certainty, but an accident happens where the car hits the pedestrian with a relatively high speed”
Testing ADAS

On-road testing

Time-consuming
Expensive

Simulation-based (model) testing

A simulator based on physical/mathematical models
Model Testing ADAS

Simulator (Matlab/Simulink)

- Physical plant (vehicle / sensors / actuators)
- Other cars
- Pedestrians
- Environment (weather / roads / traffic signs)

Matlab/Simulink Model

ADAS (SUT)

Test input

Test output

Time-stamped output
Our Goal

• Developing an automated testing technique for ADAS

• To help engineers efficiently and effectively explore the complex test input space of ADAS

• To identify critical (failure-revealing) test scenarios

• Characterization of input conditions that lead to most critical situations
ADAS Testing Challenges

• Test input space is **large, complex** and **multidimensional**

• **Explaining failures and fault localization** are difficult

• Execution of **physics-based simulation models** is computationally expensive
Test Inputs/Outputs

Environment inputs
Mobile object inputs
Outputs

Weather
- weatherType: Condition

Road
- roadType: RT
- intensity: Real

SceneLight
- intensity: Real

Condition
- fog
- rain
- snow
- normal

<enumeration>

RoadSide Object

Camera Sensor
- field of view: Real

Parked Cars

Trees

Test Scenario
- simulationTime: Real
- timeStep: Real

Vehicle
- v0: Real

<<positioned>>

Collision
- state: Boolean

Detection
- certainty: Real

<positioned>

Pedestrian
- x0: Real
- y0: Real
- ð: Real
- v0: Real

<uses>

AEB

Position
- x: Real
- y: Real

Output Trajectory
- AWA

Dynamic Object

<enumeration>
- curved
- straight
- ramped
Learnable Evolutionary Algorithms

Learn regions likely to contain most critical (failure) test scenarios

Search for critical test scenarios in the critical regions, and help refine classification models
Search-Based Software Testing

• **Definition:** The application of meta-heuristic, search-based optimization techniques to find near-optimal solutions in software testing problems.

• **Problem Reformulation:** reformulating typical software testing problems as optimization problems

• **Fitness Function:** definition of functions to optimize

• **Search Algorithms:** applying search algorithm to optimise such functions
  - Hill climbing
  - Genetic Algorithms
  - Simulated Annealing
  - Tabu Search
  - Particle Swarm Optimization
  - …
Genetic Algorithms (GAs)

Genetic Algorithm: search algorithm inspired by evolution theory

- **Natural selection:** Individuals that best fit the natural environment survive
- **Reproduction:** surviving individuals generate offsprings (next generation)
- **Mutation:** offsprings inherits properties of their parents (with some mutations)
- **Iteration:** generation after generation the new offspring fit better the environment than their parents
Search-Based Test Generation

- Search for **test input data** with certain properties
- Search driven by **fitness function**
- **Examples**: Coverage source code branch, requirements conditions …
- **Non-linearity** of software (if, loops, …): complex, discontinuous, non-linear search spaces (Baresel)

“Search-Based Software Testing: Past, Present and Future”
Phil McMinn
Example: Unit Testing

Multiple objectives
Large search space: All possible test cases!
Multiple Objectives: Pareto Front

Individual A Pareto dominates individual B if A is at least as good as B in every objective and better than B in at least one objective.
Multiple Objectives: Pareto Front

A multi-objective optimization algorithm (e.g., NSGA II) must:
- Guide the search towards the global Pareto-Optimal front.
- Maintain solution diversity in the Pareto-Optimal front.
Decision Trees

Partition the input space into homogeneous regions

RoadTopology (CR = [10 – 40] (m))

RoadTopology (CR = 5, Straight, RH = [4 – 12] (m))

Count 1200
“non-critical” 79%
“critical” 21%

Count 564
“non-critical” 59%
“critical” 41%

\[ \theta^p_0 < 218.6^\circ \]

Count 412
“non-critical” 49%
“critical” 51%

\[ \theta^p_0 \geq 218.6^\circ \]

Count 636
“non-critical” 98%
“critical” 2%

\[ v^p_0 = 7.2 \text{km/h} \]

Count 230
“non-critical” 31%
“critical” 69%

\[ v^p_0 < 7.2 \text{km/h} \]

Count 182
“non-critical” 72%
“critical” 28%

“non-critical” 31%
“critical” 69%

v^p_0 > 7.2 \text{km/h}
Our ADAS Testing

- We use decision tree classification models

- We use multi-objective search algorithm (NSGAII)

- **Objective Functions:**
  1. Minimum distance between the pedestrian and the field of view
  2. The car speed at the time of collision
  3. The probability that the object detected is a pedestrian

- Each search iteration **calls simulation** to compute objective functions

- Input values required to perform the simulation:

  - Precipitation
  - Fogginess
  - Road shape
  - Visibility range
  - Car-speed
  - Person-speed
  - Person-position
  - Person-orientation
Genetic Evolution Guided by Classification

Initial input
Genetic Evolution Guided by Classification

Initial input
Fitness computation
Genetic Evolution Guided by Classification

Initial input
Fitness computation
Classification
Genetic Evolution Guided by Classification

Initial input
Fitness computation
Classification
Selection
Genetic Evolution Guided by Classification

Initial input ✓
Fitness computation ✓
Classification ✓
Selection ✓
Breeding ✓
We focus on generating more scenarios in the critical region, respecting the conditions that lead to that region.

We get a more refined decision tree with more critical regions and more homogeneous areas.
Research Questions

• RQ1: Does the decision tree technique help guide the evolutionary search and make it more effective?

• RQ2: Does our approach help characterize and converge towards homogeneous critical regions?

• Failure explanation

• Usefulness (feedback from engineers)
Usefulness

• The characterizations of the different critical regions can help with:

  (1) **Debugging** the system model (or the simulator)

  (2) **Identifying possible hardware changes** to increase ADAS safety

  (3) **Providing proper warnings** to drivers
Automated Testing of Feature Interactions Using Many Objective Search
System Integration

System Under Test (SUT)

- sensors
- cameras
- feature 1
- feature 2
  ...
- feature n

Integration component

actuators
Case Study: SafeDrive

• Our case study describes an automotive system consisting of four advanced driver assistance features:

  • Advanced Cruise Control (ACC)
  • Traffic Sign Recognition (TSR)
  • Pedestrian Protection (PP)
  • Automated Emergency Breaking (AEB)
Simulation

Simulator

Ego Vehicle (physical plant)
- Dynamic models
- actuators
- sensors
- cameras

Environment
- mobile objects
- Pedestrians
- Other Vehicles
- static aspects
  - Road
  - Traffic sign
  - Weather

Inputs
- the initial state of the physical plant and the mobile environment objects
- the static environment aspects

Outputs
Time-stamped vectors for:
- the SUT outputs
- the states of the physical plant and the mobile environment objects

Feedback loop
Actuator Command Vectors

PP

AEB

TSR

ACC

IntC

If (condition)

b

40%(b_{AEB}(0))

40%(b_{AEB}(1))

80%(b_{PP}(2))

80%(b_{PP}(3))

...}

80%(b_{PP}(T/\delta))

b : braking
a : acceleration
s : steering

b^{PP}

0
60%
80%
80%
80%
...
T
\delta
80%

b^{AEB}

0
40%
40%
40%
20%
...
T
\delta
0%

b^{TSR}

0
T
\delta

b^{ACC}

0
T
\delta

39
## Safety Requirements

<table>
<thead>
<tr>
<th>Feature</th>
<th>Requirement</th>
<th>Failure distance functions ($FD_1, \ldots, FD_5$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PP$</td>
<td>No collision with pedestrians</td>
<td>$FD_1(i)$ is the distance between the ego car and the pedestrian at step $i$.</td>
</tr>
<tr>
<td>$AEB$</td>
<td>No collision with cars</td>
<td>$FD_2(i)$ is the distance between the ego car and the leading car at step $i$.</td>
</tr>
<tr>
<td>$TSR$</td>
<td>Stop at a stop sign</td>
<td>Let $u(i)$ be the speed of the ego car at time step $i$, once it reaches a stop sign. If there is no stop sign, then $u(i) = 0$. We define $FD_3(i) = 0$ if $u(i) \geq 20km/h$. Otherwise, we define $FD_3(i) = \frac{1}{u(i)}$. If there is no stop sign, we have $FD_3(i) = 1$.</td>
</tr>
<tr>
<td>$TSR$</td>
<td>Respect the speed limit</td>
<td>Let $u'(i)$ be the difference between the speed of the ego car and the speed limit at step $i$ if a speed limit sign is detected. If there is no speed limit sign $u'(i) = 0$. We define $FD_4(i) = 0$ if $u(i) \geq 20km/h$. Otherwise, we define $FD_4(i) = \frac{1}{u'(i)}$. If there is no speed limit sign, we have $FD_4(i) = 1$.</td>
</tr>
<tr>
<td>$ACC$</td>
<td>Respect the safety distance</td>
<td>$FD_5(i)$ is the absolute difference between the safety distance $sd$ and $FD_2(i)$.</td>
</tr>
</tbody>
</table>
Features & Interactions

- Behavior of features based on **machine learning algorithms processing sensor and camera data**

- **Interactions** between features may lead to **violating safety requirements**, even if features are correct

- **Example**: ACC is controlling the car by ordering it to accelerate since the leading car is far away, while a pedestrian starts crossing the road. PP starts sending braking commands to avoid hitting the pedestrian.

- **Complex**: predict and analyze possible interactions at the requirements level

- **Resolution strategies** cannot always be determined statically and may depend on the state of the environment
Objective

- **Automated and scalable testing** to help ensure that resolution strategies are safe
- Detect **undesired feature interactions**
- **Assumptions:** IntC is white-box (integrator is testing), features were previously tested
Input Variables

Test input vector $X = (x_0^p, y_0^p, \theta^p, \varepsilon_0^p, \varepsilon_0^l, \theta_0^l, \theta_0^l, x_0^l, x_0^{ts}, f_i)$
Search

• Input space is very large

• **Dedicated search algorithm** directed/guided by many objectives (fitness functions)

• **Fitness (distance) functions**: reward test cases that are more likely to reveal integration failures leading to safety violations

• Combine **three types of functions**: (1) safety violations, (2) unsafe overriding by integration component (IntC), (3) coverage of the decision structure of IntC

• Many test objectives to be satisfied by the test suite
Failure Distance

• Goal: Reveal safety requirements violations

• Fitness functions based on the trajectory vectors for the ego car, the leading car and the pedestrian, generated by the simulator

• **PP fitness**: Minimum distance between the car and the pedestrian during the simulation time.

• **AEB fitness**: Minimum distance between the car and the leading car during the simulation time.
Unsafe Overriding Distance

- **Goal:** Find faults in integration component
- Reward test cases generating integration outputs *deviating from the individual feature outputs*, in such a way as to possibly lead to safety violations.
- **Example:** A feature \( f \) issues a braking command while the integration component issues no braking command or a braking command with a lower force than that of \( f \).
Branch Distance

- Many decision branches in IntC
- Branch coverage of IntC
- **Fitness:** Approach level and branch distance $d$ (standard for code coverage)
- $d(b, tc) = 0$ when tc covers b
Combining Distance Functions

- **Goal:** Execute every branch of IntC such that while executing that branch, IntC unsafely overrides every feature \( f \) and its outputs violate every safety requirement related to \( f \).

\[
\Omega_{j,l}(i) = \begin{cases} 
BB_{j}(i) + \text{Max}(UOD) + \text{Max}(FD) & \text{(1) If } j \text{ is not covered } (BB_{j}(i) > 0) \\
UOD_{j}(i) + \text{Max}(FD) & \text{(2) If } j \text{ is covered, but } f \text{ is not unsafely overridden } (BB_{j}(i) = 0 \land UOD_{j}(i) > 0) \\
FD_{j}(i) & \text{(3) Otherwise } (BB_{j}(i) = 0 \land UOD_{j}(i) = 0) 
\end{cases}
\]

\[
\Omega_{j,l} = \text{Min}_{i=0}^{\infty} \Omega_{j,l}(i)
\]

- \( \Omega_{j,l}(tc) > 2 \) Indicates that \( tc \) has not covered branch \( j \)
- \( 2 \geq \Omega_{j,l}(tc) > 1 \) Branch covered but did not cause unsafe override of \( f \)
- \( 1 \geq \Omega_{j,l}(i) > 0 \) Branch covered, unsafe override, but did not violate requirement \( I \)
Search Algorithm

- Best test suite covers all (feasible) search objectives, i.e., for all IntC branches and all safety requirements
- Not a Pareto front optimization problem
- Objectives compete with each others for each test case
- Example: We cannot have the ego car violating the speed limit after hitting the leading car in one test case
- Tailored, many-objective genetic algorithm
- Must be efficient (test case executions are very expensive)
Evaluation on SafeDrive

The graph shows the number of integration errors over time (in hours) for two conditions: FITest and Baseline. The x-axis represents the time in hours, ranging from 0 to 12 hours. The y-axis represents the number of integration errors, with values ranging from 0 to 7. The blue line indicates the FITest condition, showing an increasing trend in errors over time, while the red dotted line represents the Baseline condition, also showing an increasing trend but starting from a lower baseline.
Summary

- Machine learning plays an increasingly prominent role in autonomous systems
- No (complete) requirements, specifications, or even code
- Some safety and mission-critical requirements
- Neural networks (deep learning) with millions of weights
- How do we gain confidence, through automated testing, in such software in a scalable and cost-effective way?
- We propose solutions based on metaheuristic search and machine learning
Related Testing Research

• Testing of hybrid controllers
• Testing timeliness requirements
• Testing for deadline misses (schedulability)
• HiL acceptance testing prioritization
• Testing for security vulnerabilities
• Find publications on: svv.lu
Acknowledgements

• Raja Ben Abdessalem
• Shiva Nejati
• Annibale Panichella
• IEE, Luxembourg
References


• R. Ben Abdessalem et al., "Testing Autonomous Cars for Feature Interaction Failures using Many-Objective Search”, IEEE/ACM/IEEE ASE 2018
Automated Testing of Autonomous Driving Assistance Systems

Lionel Briand

Vector Testing Symposium, Stuttgart, 2018