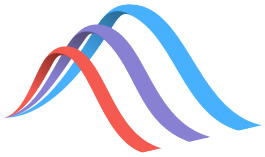
Testing ADAS/AV Algorithms with TrustworthySearch

Version 0.2

Justin Norden, Matthew O'Kelly, & Aman Sinha

{justin, matt, aman}@trustworthy.ai



Trustworthy AI
Palo Alto, CA, USA

Executive Summary

This document proposes a way to use the TrustworthySearch API to test an ADAS/AV algorithm's performance on a driving scenario. The TrustworthySearch API is an optimization layer on top of an existing simulation system; it finds coverage of failure modes and prioritizes them to enable continuous integration. We describe an example parametrization for a T-junction scenario as well as a formalism for metrics to score simulations on this scenario. Finally, we illustrate how to use the TrustworthySearch API, including ways to compare performance against current testing strategies and techniques to integrate into existing workflows.

1 Introduction

Overall Objectives:

We believe that having the ability to run millions of miles in simulation on cloud infrastructure is only half of the problem in ADAS/AV testing. The second half involves knowing (or efficiently learning) which simulations to run and digesting the output for analysis so that engineers are empowered to make improvements. The use cases outlined in Section 5 all relate to this key point.

In this document we outline a potential use-case for the TrustworthySearch API to test an ADAS/AV algorithm's performance on a driving scenario. Section 2 describes the scenario and the possible variables that can be used as simulation parameters. Section 3 describes possible ways to score simulations, including techniques to include notions of blame. Finally, Sections 4 and 5 describe techniques to perform integration of the TrustworthySearch API.

What is TrustworthySearch? The TrustworthySearch API is an efficient risk analysis service for analyzing advanced driver-assistance system (ADAS) and autonomous vehicle (AV) technologies. The service acts as an optimization layer on top of simulation by choosing which scenarios to run, providing statistical coverage over failure modes, and prioritizing failures by likelihood. Our proprietary algorithms efficiently generate useful data and actionable insights regarding safety. Most importantly, TrustworthySearch interacts with ADAS/AV systems in a black-box manner, preserving the integrity and confidentiality of the software under test. TrustworthySearch is the only third-party black-box statistical safety assessment service for ADAS/AV algorithms on the market today.

Action Items Checklist

- ☐ Determine base scenario *e.g.* unprotected left at a T-junction
- ☐ Determine whether the simulation is pre- or post-perception
- ☐ Identify if the scenario execution is deterministic
- ☐ Determine environment-vehicle parameterization
- ☐ Write scenario parameterization as a list of random variables
- ☐ Decide on a set of laws which describe the specification
- ☐ Identify any laws which have binary semantics
- ☐ Determine the metric(s) that will be used to score simulations
- ☐ Determine exit conditions and how they will affect the satisfaction of the specification
- ☐ Determine a rough estimate of the failure probability to estimate of the computational resources that will be necessary for full evaluation
- ☐ Integrate TrustworthySearch API with current simulation stack by building upon the demo code in Section 4

2 Scenario Parameterization and Description

Objectives: Scenario Description

You may already have a preferred scenario parameterization or description language within your team or company. The technical portions of the following section serve to align our language; it is not necessary that your description be identical, as the TrustworthySearch API treats the simulator as a black box. A secondary goal of this section is to nail down the specifics of the scenario that will be tested.

2.1 Proposed scenario

We reference our team's earlier research on scenario description languages (SDL's) presented in O'Kelly et al. [6]. In our view a scenario is a meta-simulation which encodes a set of possible realizations of a driving task. More specifically, a scenario $\mathbf{S}$ consists of a set of agents, A, that includes the ego-vehicle and other vehicles and the road, a set of traffic laws $\mathcal{L}$, a goal Φ to be achieved by the ego-vehicle in this scenario, exit conditions $\mathcal{E}$ that define when a scenario is over, and a set of initial states $Init$ that captures the initial states of all vehicle agents.

$$\mathbf{S} = (A, \mathcal{L}, \Phi, Init, \mathcal{E})$$

Rather than define these formally, we illustrate them using the T-Junction scenario depicted in Figure 2.1.

Discussion Point: Modify this picture and scenario description

Feel free to annotate and update this scenario example so that it is relevant to your desired test cases. Obvious potential changes could include converting this to an unprotected left or adding more environment vehicles.

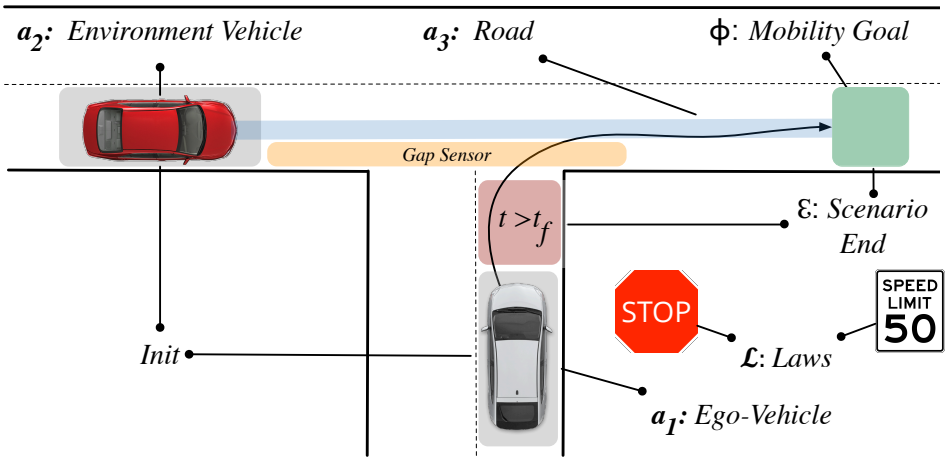


Figure 2.1: The T-Junction Scenario.

Agents (A): There are 3 agents: the `ego-vehicle` (a_1), `environment-vehicle` (a_2), and the `road` (a_3). It is unrealistic to assume perfect knowledge of the environment vehicle's intentions and dynamics. Thus, the latter should be modeled non-deterministically, *e.g.* $\dot{x} = v_x, \dot{v}_x \in [0, 1]\frac{m}{s^2}$ or probabilistically, *e.g.* $\dot{x} = v_x, \dot{v}_x \sim P$ for some distribution P. Roads can be described as a graph of lanes where each lane is a finitely-parameterized curve or sequence of curves: *e.g.* straight lines (with parameter: *length*), cubic splines (parameters: *length*, *curvatures*), or polylines. In this sense even the road geometry can be made a searchable parameter of the scenario.

Law Set ($\mathcal{L}$): The laws are fixed in a given scenario, common encodings could include Metric Temporal Logic (MTL) [2] or Signal Temporal Logic (STL) [3]. **Note that the use of a temporal logic is not required; most reinforcement-learning reward functions work. See Section 3.1 for an alternate example.** In Figure 2.1, one law imposes a speed limit of 50 (in whatever desired units of length and time you prefer) for the duration T of the scenario: $l = \text{Always}_{[0,T]}(v < 50)$.

Goal (Φ): The goal Φ of the ego vehicle is of the form "Ego must reach some region within N time units". In Figure 2.1, the goal region is the green rectangle, expressing that the vehicle should turn onto the main road within some bounded time horizon.

Initialization ($Init$): An ADAS/AV a estimates its state x_0 to within some bounded error (due to measurement noise), so it only knows that x_0 is in some set $Init_a$. Thus, it is necessary at simulation time to verify that whatever actual value x_0 has in $Init_a$, will not lead trivially to a collision or to a requirement violation. The set $Init$ is the product of all vehicles' initial sets: $Init = \Pi_{a \in A} Init_a$. The initial sets of the 2 cars are shown in light grey in Figure 2.1.

Exit Condition ($\mathcal{E}$): Finally, the scenario ends either when the ego vehicle leaves the region defined by the T-Junction (the green region in Figure 2.1) and proceeds to the next navigation task, or a timeout occurs (the red region in Figure 2.1). Other exit conditions could include a "handover" signal initiated by any part of the stack.

Discussion Point: How are different types of exit conditions evaluated?

Simulations can "complete" for a variety of reasons. In some cases the simulation may timeout or have a crash. Determining how such issues are reported back to the module responsible for proposing the next simulations to run (*e.g.* the TrustworthySearch API) is important. Should those samples be rejected? Should they receive a large penalty? Should they include a binary indicator function? Any of the decisions may be valid for your specific use case, and these choices must be made judiciously before starting a test.

In summary, the goal of a SDL is to enable the user to describe these elements of a scenario in a consistent and structured manner. Structuring scenario descriptions in this manner automatically handles low-level details like ensuring compile-time correctness (*e.g.* the type and dimensionality of variables), as well as run-time issues like maintaining a global clock, and monitoring for important events like scenario-end.

2.2 Simulation parameters and their distributions

Below we describe a possible parametrization for a highway scenario, including vehicle variables for vehicle poses/behaviors and weather. This description is an excerpt from Norden et al. [5], where we use the open-source CARLA simulator to test Comma AI's OpenPilot on highway scenarios.

Agent Variables For the behaviors of the environment vehicles we follow the parametrization of O'Kelly et al. [7]. Namely, we search over the last layer of a neural network GAIL generator model. We use $\xi \in \mathbb{R}^{404}$ to denote the weights of the last layer of the neural network trained to imitate human-like driving behavior with $\xi \sim \mathcal{N}(\mu_0, \Sigma_0)$, with the mean and covariance matrices learned via empirical bootstrap. The input for this GAIL generator is described in O'Kelly et al. [7]. The single network defined by ξ drives the behavior for all environment agents in a particular simulation rollout.

Initialization Variables Initial positions for each vehicle are independently given by the pair (S, T), where $S \sim 500\text{Beta}(2, 2) + 200$ denotes the longitudinal position along the stretch of road with respect to the start of the road, and $T \sim 0.5\text{Beta}(2, 2) - 0.25$ denotes the lateral distance from the lane center with left being

positive. Both have units in meters. Initial velocities are independently given by $V \sim 10\text{Beta}(2,2) + 15$ with units in meters/second.

Perception variables If the scenario uses a rendering engine or sensor simulation tool in the loop there will be additional factors which can be used to parameterize the scenario. In what follows we describe parameters for the CARLA simulator; for other proprietary simulators a similar methodology may be applied. In order to define the weather and lighting conditions 4 parameters are defined: precipitation on the ground (P_g), the altitude angle of the sun (A), cloudiness (C), and precipitation in the air (P_a). Precipitation on the ground follows the distribution $P_g \sim 50\text{Uniform}(0,1)$, where the units are proprietary for the CARLA simulator. Sun altitude follows the distribution $A \sim 90\text{Uniform}(0,1)$, whose units are in degrees. Letting $C_b \sim \text{Beta}(2,2)$ and $C_u \sim \text{Uniform}(0,1)$, cloudiness has a distribution given by the following mixture:

$$C \sim 30C_b\mathbf{1}\{C_u < 0.5\} + (40C_b + 60)\mathbf{1}\{C_u \geq 0.5\},$$

where $\mathbf{1}\{\cdot\}$ is the indicator function and the units of C are in proprietary units for CARLA. Precipitation in the air is a deterministic function of cloudiness, $P_a = C\mathbf{1}\{C \geq 70\}$, where the units are again proprietary CARLA units.

2.3 Simulator execution semantics and tricks

Discussion Point: Determinism?

Knowing whether your simulator is deterministic or not can change the way you search for failures (or use the TrustworthySearch API). If the simulator is deterministic but the real system is not, is there value in supporting non-deterministic hardware-in-the-loop executions? In our opinion, determinism is better, but we can support non-deterministic simulations if the proper variable is set in the job request.

Deploying the planning and control elements together with the perception pipeline in a simulation environment requires further refinements. The perception and planning elements may use a real-time clock to stamp vehicle-state messages. However, the simulation clock does not necessarily match wall-clock time. To address this problem, the simulator should supply the system under test with a clock signal. Furthermore, the system under test may utilize asynchronous message passing; this induces non-determinism in the execution order of the control stack. In order to improve the repeatability of the experiments, it may be desirable to force sequential execution of the control components. Lastly, there are often low-level PID controllers operating at, say, 100Hz to actuate steering and throttle mechanisms and track the desired state (updated by the planning module at, say, 20Hz). In order to avoid significant time costs and execute the simulator at 20Hz, it is possible to send the planning module's desired state to the simulator and perform tracking on physics substeps within the simulator.

3 Metrics for Evaluating Vehicle Performance

We motivate the notion of metrics for evaluating vehicle performance by first defining a notion of risk. Following Norden et al. [5], let P_0 denote the base distribution that models standard traffic behavior and $X \sim P_0$ be a realization of the simulation (*e.g.* weather conditions and driving policies of other agents). For a continuous objective function $f : \mathcal{X} \to \mathbb{R}$ that measures "safety"—so that low values of $f(x)$ correspond to dangerous scenarios—our goal is to evaluate the probability of a dangerous event

$$p_\gamma := \mathbb{P}_0(f(X) < \gamma), \tag{3.1}$$

for some threshold γ. Examples of this objective function f include simple performance measures such as the (negative) maximum magnitude acceleration or jerk during a simulation. More sophisticated measures include the minimum distance to any other object during a simulation or the minimum time-to-collision (TTC) during a simulation. The objective can even target specific aspects of system performance. For example, to stress-test a perception module, the objective can include the (negative) maximum norm of off-diagonal elements of the perception system's confusion matrix.

Objectives: Blame and Vehicle Performance Metrics

The goal of this section is to describe different ways to measure the performance of an autonomous vehicle within a scenario. Particular weight is given to mechanisms which encode (1) "fault" or (2) map binary measures to a continuous metric. We discuss the use of "responsibility-sensitive safety" (RSS) or similar fault-based systems, how hyper-parameters of such methods are defined, and ways in which current methodology is unsatisfactory. Another key question is whether "blame" will be applied post-hoc to tests or as a search metric itself.

3.1 Time-to-collision (TTC)

As described in the appendix of Norden et al. [5], let $T_i(t)$ be the instantaneous time-to-collision between the ego vehicle and the i-th environment vehicle at time step t. For a given simulation rollout X, the TTC metric is given by

$$f(X) := \min_t \left(\min_i T_i(t) \right).$$

The value $T_i(t)$ can be defined in multiple ways (see *e.g.* Sontges et al. [10]). In the aforementioned study [5], we define it as the amount of time that would elapse before the two vehicles' bounding boxes intersect assuming that they travel at constant fixed velocities from the snapshot at time t. Time-to-collision captures directly whether or not the ego-vehicle was involved in a crash. If it is positive no crash occurred, and if it is 0 or negative there was a collision.

3.2 Incorporating blame

It's not strictly necessary to use RSS or blame-based formalization. We have had good results using TTC and even some binary metrics.

3.2.1 Basic traffic-rules when no collision is present

As anyone who has ever recieved a ticket for speeding or rolling through a stop sign can assert, it is possible to violate regional traffic laws and conventions without crashing. In particular, laws such as stopping or turning from the proper lane are more naturally expressed as true or false than typical highway regulations regarding maintaining a safe speed of travel. The Vienna Traffic Convention is one source of such rules,

although the presentation is in natural language and insufficient to be formally evaluated. Several researchers have proposed encoding the rules in higher-order logic, temporal logic, and other formalisms (*e.g.* Rizaldi and Althoff [8]). The logical encoding is not particularly interesting or relevant, but the behavior of the simulation when such rules are violated is important.

Discussion Point: Traffic laws in urban driving can be handled in multiple ways

Violation of traffic rules could be considered an exit condition for the simulator, a component of the objective function returned by the simulator to the TrustworthySearch API, or a means of filtering executions after a set of simulation runs has been completed.

3.2.2 Responsibility in the event of an accident

Beyond simple moving violations there is still more to say about crashes (instances where $TTC \leq 0$); when a crash occurs between two or more human driven vehicles, law enforcement generally assesses which drivers are responsible for causing the accident. While laws vary between jurisdictions the majority of laws can be derived from a few invariant "common-sense" rules [9]:

1. Do not hit someone from behind
2. Do not cut-in recklessly
3. Right-of-way is given not taken
4. Use caution in areas of limited visibility
5. If you can avoid an accident without causing another one, you must do it

The arguments presented in the RSS framework of Shalev-Shwartz et al. [9] (and the nearly identical presentation of Nistér et al. [4]) attempt to formalize the common-sense paradigm above. However, the ideas are hardly new or unique to the RSS framework (see the 1938 article by Gibson and Crooks [1]). In particular, RSS and its variants describe notions of a safe-following distance as well as "proper" responses to dangerous situations. While RSS is not a sound verification framework due to the existence of time-varying and situation-specific acceleration bounds (see Remark 2 in Shalev-Shwartz et al. [9]), it can be useful for filtering or guiding test case generation.

Discussion Point: Handling RSS parameters

How are maximum acceleration parameters assigned in any current RSS-based filtering of simulation traces? Have you considered leaving them as a parameter with a distribution and searching over them?

Discussion Point: To guide or filter?

You can use RSS-like safety-metrics to guide the search for interesting scenarios or use the binary results in a post-hoc manner to triage and filter simulation traces prior to human inspection. Both of these strategies can be useful to automate large-scale testing, either separately or in tandem.

4 Mechanics of API Integration

For code and detailed instructions on the mechanics of integrating TrustworthySearch with your simulation system, please go to the following Github repository:
`https://github.com/trustworthy-ai/trustworthysearch-demo`

The code at the above repository provides an example of a client which acts as a broker to local simulation workers. The broker receives jobs from the search server, load balances amongst workers, and then sends results back to the search server asynchronously. If you have trouble, email `support@trustworthy.ai`.

5 TrustworthySearch and Strategies for Integration

Objectives: Using TrustworthySearch

In this section, we describe possible ways to test TrustworthySearch in your continuous-integration and QA systems. We understand that different teams within your organization may have different needs and pain points when it comes to using simulation to drive improvements, and we believe TrustworthySearch can be used effectively in a variety of workflows. Overall, we want to decrease the time it takes from starting a test to receiving valuable feedback on what to work on next.

5.1 Integration strategies for testing and continuous integration (CI)

Discussion Point: Where do you think we best fit in to your system?

There are multiple ways to use the TrustworthySearch API. Here are a few sample integration strategies. Please reach out to us if you have another use case in mind.

Here are the most common ways our customers use TrustworthySearch:

1. Test nightly builds by evaluating risk on certain scenarios
2. A/B test for differences between two ADAS/AV stacks on a given scenario
3. Build an importance-sampling library of scenarios to use in fixed regression tests
4. Build importance samplers for performing enhanced or "'fuzzed" log replay with perturbations that are learnt to be dangerous for a certain score function (or functions).
5. Make the ego-vehicle policy part of the search space to perform sensitivity analysis to changes with respect to certain parameters

You are likely already testing your system using one of these strategies. Next we'll discuss how you might go about determining whether TrustworthySearch provides value on top of your current system.

5.2 Measuring the benefits of TrustworthySearch

Discussion Point: What is the best way to evaluate?

There are many ways to test whether TrustworthySearch is making improvements to an existing testing or continuous integration system. We outline a few ways to do so below, but we welcome your suggestions based on your comprehensive understanding of how your current testing framework operates.

1. Consider a scenario you wish to test. For a given number of simulations, determine if TrustworthySearch provides more useful failures and failure analysis than an existing testing approach. This includes
   - Coverage: which algorithm discovers more/all failure modes?
   - Prioritization: TrustworthySearch outputs a ranking of which failure modes are more important to fix first. Is this better or worse than the current prioritization method?
   - Failure sampler: TrustworthySearch builds a parametric importance sampler for the discovered failure modes that can be used to generate more "hard" test cases as a warm-start to test future versions of the ADAS/AV stack (the sampler can be thought of as generator for a library of scenarios similar to the discovered failure modes). Is this sampler of failure modes better or worse than the current way that previous failures are used to inform generation of similar scenarios?
2. Consider an ADAS/AV stack that has been tested before and has a known problem on a certain scenario. Run TrustworthySearch on the stack to see if it can find the problem and perform risk analysis on it more efficiently and/or completely than current techniques (including coverage and prioritization of failure modes).
3. Consider two ADAS/AV stacks, one of which has a known problem and the other which claims to have fixed this problem without introducing further problems. Run TrustworthySearch on the second stack to see if it is truly a Pareto improvement on the first stack. Determine if this test is faster or more complete (with respect to coverage and prioritization) than current evaluation techniques.
4. Consider two ADAS/AV stacks, but you do not know which of the two is better. Rather than perform risk analysis on each of them independently, run TrustworthySearch to determine regions where the two stacks maximally differ in performance (conditioned on both of their performances being below some global threshold as well). This means finding regions where the first stack has worse performance than the second stack and vice versa. Determine whether this process occurs more efficiently and completely (with respect to coverage and prioritization of the discovered regions) than current approaches to A/B testing.

References

[1] J. J. Gibson and L. E. Crooks. A theoretical field-analysis of automobile-driving. *The American journal of psychology*, 51(3):453–471, 1938.

[2] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[3] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proceedings of FORMATS-FTRTFT*, volume 3253 of *LNCS*, pages 152–166, 2004.

[4] D. Nistér, H.-L. Lee, and Y. Wang. The safety force field, Mar 2019. URL `https://www.nvidia.com/content/dam/en-zz/Solutions/self-driving-cars/safety-force-field/the-safety-force-field.pdf`.

[5] J. Norden, M. O'Kelly, and A. Sinha. Efficient black-box assessment of autonomous vehicle safety. *arXiv preprint arXiv:1912.03618*, 2019.

[6] M. O'Kelly, H. Abbas, and R. Mangharam. Computer-aided design for safe autonomous vehicles. In *2017 Resilience Week (RWS)*, pages 90–96. IEEE, 2017.

[7] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi. Scalable end-to-end autonomous vehicle testing via rare-event simulation. In *Advances in Neural Information Processing Systems*, pages 9827–9838, 2018.

[8] A. Rizaldi and M. Althoff. Formalising traffic rules for accountability of autonomous vehicles. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 1658–1665. IEEE, 2015.

[9] S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.

[10] S. Sontges, M. Koschi, and M. Althoff. Worst-case analysis of the time-to-react using reachable sets. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1891–1897. IEEE, 2018.