2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Adaptive Stress Testing for Autonomous Vehicles

Mark Koren,[1*] Saud Alsaif,[2*] Ritchie Lee,[3] and Mykel J. Kochenderfer[1]

***Abstract*— This paper presents a method for testing the decision making systems of autonomous vehicles. Our approach involves perturbing stochastic elements in the vehicle's environment until the vehicle is involved in a collision. Instead of applying direct Monte Carlo sampling to find collision scenarios, we formulate the problem as a Markov decision process and use reinforcement learning algorithms to find the most likely failure scenarios. This paper presents Monte Carlo Tree Search (MCTS) and Deep Reinforcement Learning (DRL) solutions that can scale to large environments. We show that DRL can find more likely failure scenarios than MCTS with fewer calls to the simulator. A simulation scenario involving a vehicle approaching a crosswalk is used to validate the framework. Our proposed approach is very general and can be easily applied to other scenarios given the appropriate models of the vehicle and the environment.**

I. INTRODUCTION

While major advances have been made in improving the capabilities of decision making systems for automated vehicles, validation of these systems is challenging due to the vast space of driving scenarios [1]–[3]. Establishing confidence in any automotive system will involve road tests, but road tests alone do not adequately cover the space of critical scenarios [4]–[6]. Hence, research efforts have focused on validation by simulation with a particular emphasis on building realistic models of the environment, including models of driving behavior and sensors [7]–[11].

In the validation of autonomous vehicles, it can be very valuable to know the most-likely failure scenarios as predicted by a probabilistic model of the environment. Direct sampling is inefficient due to the rarity of failure events. Adaptive stress testing (AST) has been proposed as a practical approach to finding most-likely failure scenarios by using a Markov decision process (MDP) formulation. The AST approach was first applied to test an aircraft collision avoidance system [12]. The original method uses Monte Carlo tree search (MCTS) with double progressive widening (DPW) [13] to search for the most-likely failure condition. This paper adapts this approach to the automotive setting where the system under test (SUT) is an autonomous vehicle with noisy sensors approaching a pedestrian crosswalk. This paper also proposes deep reinforcement learning (DRL) as an alternative solver for AST.

*These authors contributed equally.

[1]Mark Koren and Mykel J. Kochenderfer are with Aeronautics and Astronautics, Stanford University, Stanford, CA 94305, USA {mkoren, mykel}@stanford.edu

[2]Saud Alsaif is with Electrical Engineering, Stanford University, Stanford, CA 94305, USA salsaif@stanford.edu

[3]Ritchie Lee is with Electrical and Computer Engineering, Carnegie Mellon University Silicon Valley, Moffett Field, CA 94035, USA rititchie.lee@sv.cmu.edu

This paper makes the following contributions:

- We extend the AST methodology by introducing a new solver technique, DRL. We show that the theoretical advantages of AST still apply to the more general formulation.
- We present a simulation framework for autonomous vehicles that interfaces with our AST implementation. The framework is modular, which enables components such as decision making systems, sensor models, and simulation dynamic models to be interchanged with alternative and more sophisticated versions.
- We apply AST to a set of autonomous vehicle scenarios and show that AST can successfully find high probability failure scenarios. Our experiments show that DRL can find better solutions much more efficiently compared to MCTS.

The remainder of the paper is organized as follows. Section II provides a review of MDPs, the two reinforcement learning algorithms used in this paper, and the AST framework. Section III introduces the specific formulation of AST for autonomous vehicles as well as the problem setup and evaluation metrics. Section IV presents the experimental results and analysis of performance. Section V concludes the paper.

II. BACKGROUND AND METHODOLOGY

We present background material on the MDP formulation and the solvers we use along with the AST methodology. We also extend the AST methodology to include DRL.

A. Markov Decision Processes

In a Markov decision process (MDP), the agent chooses an action a based on a state s and receives a reward r according to the reward function $R(s, a)$ [14]. The state transitions to the next state s' stochastically according to the state transition function $T(s' \mid s, a)$. The probability of transitioning to state s' depends only on s and a, which is known as the Markov assumption. The goal of an agent is to find a policy π that specifies the action $a = \pi(s)$ at each state to maximize the expected utility. The utility of executing a policy π from state s is given recursively by the value function:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s' \mid s, \pi(s)) V^{\pi}(s') \quad (1)$$

where γ is the discount factor that controls the weight of future rewards. Reinforcement learning algorithms, such as MCTS and DRL, can be used to find the optimal policy $\pi(s)$.

B. Monte Carlo Tree Search

MCTS is an online sampling-based algorithm that can be used to solve MDPs [15]. MCTS builds a search tree by sampling the state space and using forward simulation to estimate the value of states and actions. This paper uses a variation of MCTS with double progressive widening (DPW) [13]. DPW regulates branching in the search tree to prevent the number of children from exploding when the number of states or actions is very large.

C. Deep Reinforcement Learning

Deep reinforcement learning (DRL) is an alternative approach to solving MDPs that uses a feed-forward neural network to represent the policy $\pi_\theta(s)$. The policy is parameterized by θ, which represents the neural network weights. In our implementation, the policy maps an input state to the mean of a Gaussian distribution. The actions are then sampled from the distribution $a \sim \mathcal{N}(\pi_\theta(s), \Sigma)$, with the diagonal covariance matrix Σ separately parametrized and independent of state [16].

To update the policy, we use Generalized Advantage Estimation (GAE) to estimate the policy-gradient from batches of simulation trajectories [17]. GAE defines an advantage function

$$A^{\pi,\gamma}(s,a) := Q^{\pi,\gamma}(s,a) - V^{\pi,\gamma}(s) \tag{2}$$

where γ is the discount factor that governs the weight of future rewards. The Q-function evaluates the value of taking an action from a state and then following the policy, and is defined as

$$\begin{aligned} Q^{\pi,\gamma}(s,a) &= R(s,a) \\ &\quad + \gamma \sum_{s'} T(s' \mid s,a)\, Q^{\pi,\gamma}(s', \pi(s')) \end{aligned} \tag{3}$$

Once the policy-gradient is known, Trust Region Policy Optimization (TRPO) is used to step the policy. TRPO generally gives monotonic increases in policy performance by constraining the KL divergence [16].

D. Adaptive Stress Testing

We formulate the problem of finding failure events as a sequential decision process, following prior work [12]. The inputs to the problem are the pair ($\mathscr{S}$, E), where $\mathscr{S}$ is a generative simulator that is treated as a black box and E is a subset of the state space where the event of interest (e.g. a collision) occurs. The simulator contains the models for the SUT, the models of the other agents in the environment, and the dynamics of the environment. The simulator exposes the following simulation control functions to the solver:

- INITIALIZE($\mathscr{S}$): Resets $\mathscr{S}$ to its initial state s_0.
- STEP($\mathscr{S}, E, a$): Steps the simulation in time by drawing the next state s' after taking action a. The function returns the probability of the transition and an indicator whether s' is in E or not.
- ISTERMINAL($\mathscr{S}, E$): Returns true if the current state of the simulation is in E, or if the horizon of the simulation T has been reached.

This formulation assumes that the state of the simulator is hidden, which in general means that the simulator is non-Markovian from the point of the view of the solver. However, because we have chosen our actions a to fix the stochastic elements of the simulator, the state transitions now become deterministic. In this setting, we can deterministically revisit a state s by replaying the history of actions that led to the state starting from the initial state. As such, we can use the sequence of actions $a_{0:t-1}$ to represent the state s_t [12]. The abstraction allows us to overcome partial observability in the simulator.

The definition of the problem is as follows: Given a simulator $\mathscr{S}$ and a subset of the state space E, find the most likely trajectory that leads to an event in E. Because a controls the stochastic elements of the simulator, the simulation does not evolve stochastically. Rather, the actions controlling the adversarial environment uniquely determine the evolution of the scenario.

The approach to solve the AST problem is shown in Figure 1. We start with the solver, which samples environment actions and passes them to the simulator through the control functions INITIALIZE, STEP, and ISTERMINAL. The simulator applies these actions, updates its internal state, and outputs an indication whether an event in E occurred and the likelihood of the latest state transition. The reward function transforms the simulator outputs into a reward to be passed back to the solver. The solver completes the loop by using the reward to choose the next action.

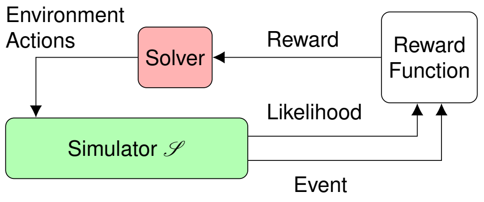


Fig. 1: The AST methodology. The simulator is treated as a black box. The solver optimizes a reward based on transition likelihood and whether an event has occurred.

III. AUTONOMOUS VEHICLES APPLICATION

This section defines the set of scenarios and metrics that we will use to evaluate the performance of the methods as well as the reward function used by the solvers.

A. Simulator Design

The driving algorithm, the sensors, the tracker, the solver, and the scenario definition are separate components in the framework. We use a modified version of the Intelligent Driver Model (IDM) as the SUT [18]. If multiple IDM implementations were to be compared, it would be easy to swap them out and compare the results. The modularity gives AST the potential to be a useful benchmarking tool, or a batch testing method for autonomous systems.

The simulator architecture is shown in Figure 2. The solver outputs the environment actions to the simulator, which is used to update non-SUT agents controlled by AST, called

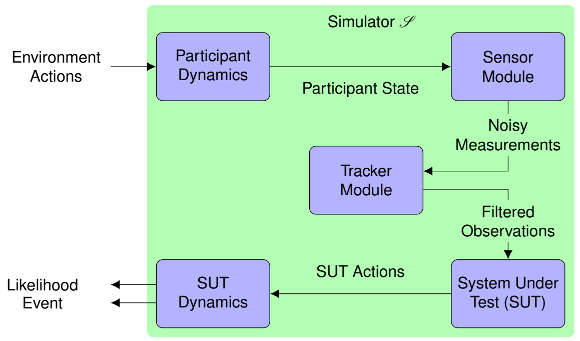
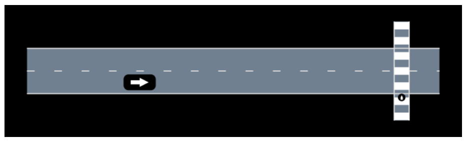
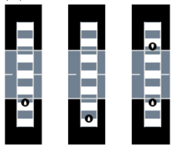


Fig. 2: The modular implementation of AST. The modules of the simulator can be easily swapped to test different scenarios, SUTs, or sensor configurations.



(a) First scenario.



(b) Initial pedestrian configurations for each scenario.

Fig. 4: Three variations of initial configurations in the crosswalk scenario.

participants. In our experiments, the only participants are pedestrians. The sensors receive the new participant states and output measurements augmented with the noise from the environment actions. The measurements are filtered by the tracker and passed to the SUT. The SUT, which is the driving model, decides how to maneuver the vehicle based on its observations.

The SUT actions are used to update the state of the vehicle. The simulator outputs the transition probability and event indicator to the reward function. The current state of the simulator can be represented in different ways. If the state of the simulator is fully observable, the simulator can provide it or an autoencoder processed version of it [19]. Otherwise, the history of previous actions can be used to represent the current state. The state representation, along with the reward of the previous step, are then input to the solver.

Solvers use different procedures to generate the environment actions as shown in Figure 3. MCTS outputs pseudorandom seeds, which are used to seed random number generators. The environment actions are then sampled using these random number generators. DRL outputs a mean and standard deviation that characterize a multivariate Gaussian distribution. Environment actions are sampled from the distribution.

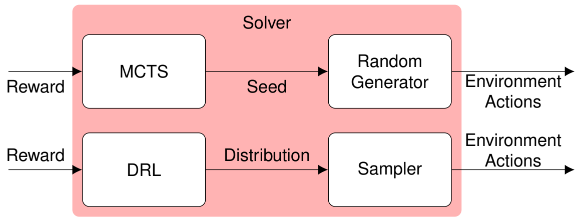


Fig. 3: A comparison of the solver methods. MCTS uses a seed to control a random number generator. DRL outputs a distribution, which is then sampled. Both of these methods produce the environment actions.

B. Problem Formulation

To evaluate the effectiveness of AST as applied to autonomous vehicles, we stress test a vehicle in a set of scenarios at a pedestrian crosswalk, shown in Figure 4. The scenario is defined by a single autonomous vehicle approaching a crosswalk. The road has two lanes to model a regular neighborhood road, although there is no traffic in either direction for this specific example. We intentionally chose the simplest possible representative scenario for pedagogical reasons to illustrate the fundamental difference in scalability between DRL and MCTS. The road and crosswalk are sized according to California state regulations [20]. The Cartesian origin is set at the intersection of the central vertical axis of the crosswalk and the central horizontal axis of the bottom lane, with the positive x direction follows the direction of the arrow in Figure 4, and positive y motion being towards the top side of the street. We test with different numbers of pedestrians, as well as with different starting states. The state of the ith pedestrian is $\mathrm{s}_{\text{ped}}^{(i)} = [v_x^{(i)}, v_y^{(i)}, x^{(i)}, y^{(i)}]$ where

- $v_x^{(i)}, v_y^{(i)}$ are the x and y components of the velocity of the ith pedestrian.
- $x^{(i)}, y^{(i)}$ are the x and y components of the position of the ith pedestrian.

We present data from each pedestrian from the three different variations of the scenario:

- 1 pedestrian, with initial state
$$\mathrm{s}_{\text{ped}}^{(1)} = [0.0\,\mathrm{m/s}, 1.4\,\mathrm{m/s}, 0.0\,\mathrm{m}, -2.0\,\mathrm{m}]$$
- 1 pedestrian, with initial state
$$\mathrm{s}_{\text{ped}}^{(1)} = [0.0\,\mathrm{m/s}, 1.4\,\mathrm{m/s}, 0.0\,\mathrm{m}, -4.0\,\mathrm{m}]$$
- 2 pedestrians, with initial state
$$\mathrm{s}_{\text{ped}}^{(1)} = [0.0\,\mathrm{m/s}, 1.4\,\mathrm{m/s}, 0.0\,\mathrm{m}, -2.0\,\mathrm{m}]$$
$$\mathrm{s}_{\text{ped}}^{(2)} = [0.0\,\mathrm{m/s}, -1.4\,\mathrm{m/s}, 0.0\,\mathrm{m}, 5.0\,\mathrm{m}]$$

shown in Figure 4b. The first scenario was chosen as a basic example to demonstrate AST. The second scenario was chosen to show that a different initial condition leads to different collision trajectories found. The third scenario shows the scalability of AST by including more participants in the scenario.

1) Environment Models: Both solvers use the same representation for environment actions. The environment action vector at each time step is $\mathbf{a}_{\text{env}} = [\mathbf{a}^{(1)}, \mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(n)}]$, where n is the number of pedestrians. For the ith pedestrian, $\mathbf{a}^{(i)} = [a_x^{(i)}, a_y^{(i)}, \epsilon_{v_x}^{(i)}, \epsilon_{v_y}^{(i)}, \epsilon_x^{(i)}, \epsilon_y^{(i)}]$ where

- $a_x^{(i)}, a_y^{(i)}$ are the x and y components of the ith pedestrian's acceleration.
- $\epsilon_{v_x}^{(i)}, \epsilon_{v_y}^{(i)}$ are the noise injected into the SUT measurement of the components of the ith pedestrian's velocity $v_x^{(i)}$ and $v_y^{(i)}$.
- $\epsilon_x^{(i)}, \epsilon_y^{(i)}$ are the noise injected into the SUT measurement of x and y components of the ith pedestrian's position.

AST controls both the pedestrian motion and the sensor noise, allowing it to search over both pedestrian actions and hardware failures to find the most likely collision.

The inputs to the solvers vary slightly. MCTS does not make use of the simulator's internal state, treating it entirely as a black box. Instead, the AST implementation of MCTS differentiates states using a history of previous pseudorandom seeds [12]. In contrast, DRL takes the simulation state as input. The simulation state is $\mathbf{s}_{\text{sim}} = [s_{\text{sim}}^{(1)}, s_{\text{sim}}^{(2)}, \ldots, s_{\text{sim}}^{(n)}]$. For the ith pedestrian, $\mathbf{s}_{\text{sim}}^{(i)} = [\hat{v}_x^{(i)}, \hat{v}_y^{(i)}, \hat{x}^{(i)}, \hat{y}^{(i)}]$ where

- $\hat{v}_x^{(i)}, \hat{v}_y^{(i)}$ are the x and y components of the relative velocity between the SUT and the ith pedestrian.
- $\hat{x}^{(i)}, \hat{y}^{(i)}$ are the x and y components of the relative position between the SUT and the ith pedestrian.

At each time step, the pedestrian samples $\mathbf{a}^{(i)}$ (the procedure of this sampling differs slightly between solvers, but the representation of the action vector $\mathbf{a}^{(i)}$ is the same). To find the likelihood of $\mathbf{a}^{(i)}$, a model of the expected pedestrian action vector is needed. The model of the pedestrian is a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_a, \boldsymbol{\Sigma})$ where $\boldsymbol{\mu}_a$ is a zero-vector, and $\boldsymbol{\Sigma}$ is diagonal. Our pedestrian model is parameterized by σ_{aLat}, σ_{aLon}, and σ_{noise}, which are the diagonal elements of the covariance matrix and correspond to lateral acceleration, longitudinal acceleration, and sensor noise respectively. The values we use are: $\sigma_{\text{aLat}} = 0.01$, $\sigma_{\text{aLon}} = 0.1$, and $\sigma_{\text{noise}} = 0.1$. The acceleration parameters are designed to encourage the pedestrians to move across the street with some lateral movement. The assumption of the mean action being the zero-vector implies that, on average, a pedestrian holds course. In reality, this distribution could depend on the location of the pedestrian, where the vehicle is, the attitude or attention of the pedestrian, or other factors. Applying a more realistic pedestrian model is an avenue for future work. The initial speed of the pedestrian is set to $1.5\,\mathrm{m/s}$, which is the average human walking speed.

2) Sensor and Tracker Models: The sensors of the SUT receive a vector of the participant state and output a vector of noisy measurements $\mathbf{m} = [m^{(1)}, m^{(2)}, \ldots, m^{(n)}]$. For the ith pedestrian, $\mathbf{m}^{(i)} = \mathbf{s}_{ped}^{(i)} + \boldsymbol{\epsilon}^{(i)}$ where $\boldsymbol{\epsilon}^{(i)} = [\epsilon_{v_x}^{(i)}, \epsilon_{v_y}^{(i)}, \epsilon_x^{(i)}, \epsilon_y^{(i)}]$. The measurements are passed to an alpha-beta tracker [21], parameterized by α_{tracker} and β_{tracker}, which returns the filtered versions of the measurements as the SUT's observations. We use the values $\alpha_{\text{tracker}} = 0.85$ and $\beta_{\text{tracker}} = 0.005$.

3) System Under Test Model: The SUT is based on the Intelligent Driver Model [18]. The IDM is designed to stay in one lane and safely follow traffic. To follow the rules around crosswalks, we set the desired velocity at 25 miles per hour ($11.17\,\mathrm{m/s}$). If no lead vehicle is available to follow, the model maintains a desired velocity. We adapted the IDM for interacting with pedestrians by modifying it to treat the closest pedestrian in the road as the target vehicle. The IDM then tries to follow a safe distance behind the pedestrian based on their relative velocity, which results in the vehicle stopping at the crosswalk since the pedestrian's velocity along the x-axis is negligible. Our modified IDM is not a safe model; as we will show, ignoring any pedestrian outside of the road makes the vehicle vulnerable to being blindsided by people moving quickly from the curb. The goal of this paper, however, is to show that AST can effectively induce poor behavior in an autonomous driving algorithm, not to develop a safe algorithm ourselves. The SUT model receives a series of filtered observations $\mathbf{o} = [o^{(1)}, o^{(2)}, \ldots, o^{(n)}]$. If there are pedestrians in the road, the SUT model uses the closest pedestrian to find $\mathbf{s}_{\text{SUT}} = [v_{\text{oth}}, s_{\text{headway}}]$ where

- v_{oth} is the relative x velocity between the SUT and the closest pedestrian.
- s_{headway} is the relative x distance between the SUT and the closest pedestrian.

These factors determine the acceleration of the SUT in the next time step.

4) Modified Reward Function: As a proxy for the probability of an action, we use the Mahalanobis distance [22], which is a measure of distance from the mean generalized for multivariate continuous distributions. The penalty for failing to find a collision is not actually $-\infty$, but instead a very large negative number. In addition, the penalty at the end of a no-collision case includes a component that is scaled by the distance (DIST) between the pedestrian and the vehicle. The penalty encourages the pedestrian to end early trials closer to the vehicle, and leads to faster convergence. The reward function is modified from the previous version of AST [12] as follows:

$$R(s) = \begin{cases} 0 & s \in E \\ -10000 - 1000 \times \textsc{dist}\left(\mathbf{p}_v, \mathbf{p}_p\right) & s \notin E, t \geq T \\ -\log\left(1 + M\left(a, \mu_a \mid s\right)\right) & s \notin E, t < T \end{cases} \tag{4}$$

where $M(a, \mu_a \mid s)$ is the Mahalanobis distance between the action a and the expected action μ_a given the current state s. The distance between the vehicle position $\mathbf{p}_v$ and the closest pedestrian position $\mathbf{p}_p$ is given by the function $\textsc{dist}(\mathbf{p}_v, \mathbf{p}_p)$.

5) Metrics: We use two metrics to evaluate the AST algorithms. The first is the likelihood of the final collision trajectory output by the system. The second metric is the number of calls to the step function. The goal of the second metric is to compare the efficiency of the two AST solvers. The separate implementations render both wall clock time and iterations inappropriate. The simulator update function

(STEP), which was the computational bottleneck, is used instead. This metric is agnostic to the implementation hardware, the algorithm used, and to the run-time of updating the simulation.

6) Solvers: For MCTS, the parameters that control how much of the state space is explored are the depth, the horizon T, and the number of iterations. The depth and horizon are chosen to be equal so that the search and rollout stages explore the same scenario. We experimented with different values for the horizon $(50, 75, 100)$, and found that 100 was the minimum horizon that is sufficiently long to cover the scenario of interest. We used 2000 iterations. For additional detail on MCTS and DPW, see the paper by Lee et al. [12].

For DRL, the results shown are obtained using a batch size of 4000. Experimentation showed that reducing the batch size any further resulted in too much variance during the trials. We use a step size of 0.1, and a discount factor of 0.99. The DRL approach is implemented using RLLAB [23].

IV. RESULTS

The results show that both solvers are able to identify failure trajectories in an autonomous vehicle scenario. MCTS and DRL produce several situations where the vehicle collides with the pedestrian. Table I shows the results for the three different scenarios. Both methods successfully converge to a solution in a tractable amount of simulator steps. AST is able to take advantage of the modified IDM's decision to ignore any pedestrian that is not in the road to find collisions. Although the likelihoods seem to vary greatly, much of this difference is due to MCTS having non-zero noise that adds up over the long horizon. The likelihood of pedestrian motion dominates that of sensor noise. As such, the noise should be very sparse, and the DRL solution reflects this. MCTS, however, has difficulty driving the noise to true zero. Instead, there are very small numbers in the noise vector throughout that can accumulate to a large probability error. We recomputed the reward as if the noise was 0 as a reference, which is also shown in Table I.

A. Performance

The number of calls to STEP for MCTS is the number of calls required to find a collision in the rollouts. The algorithm itself does not commit to the actions that cause the collision until later. The number is provided to highlight the computational ability of the algorithm to find a collision, not the termination time. In addition, the number of calls presented for MCTS is the average over 100 single runs multiplied by 100 to represent the number of calls needed to have confidence in the results.

Across all scenarios, DRL consistently requires orders of magnitude fewer calls to STEP than does MCTS. DRL converges to solutions with less than 1% of the number of calls to STEP required by MCTS despite the state and action spaces being very small. Theoretically, the scalability advantages of DRL should be even more apparent in a higher-dimensional problem. This advantage is supported by the fact that MCTS performs worse on the dual pedestrian scenario relative to both single pedestrian scenarios.

B. Trajectories

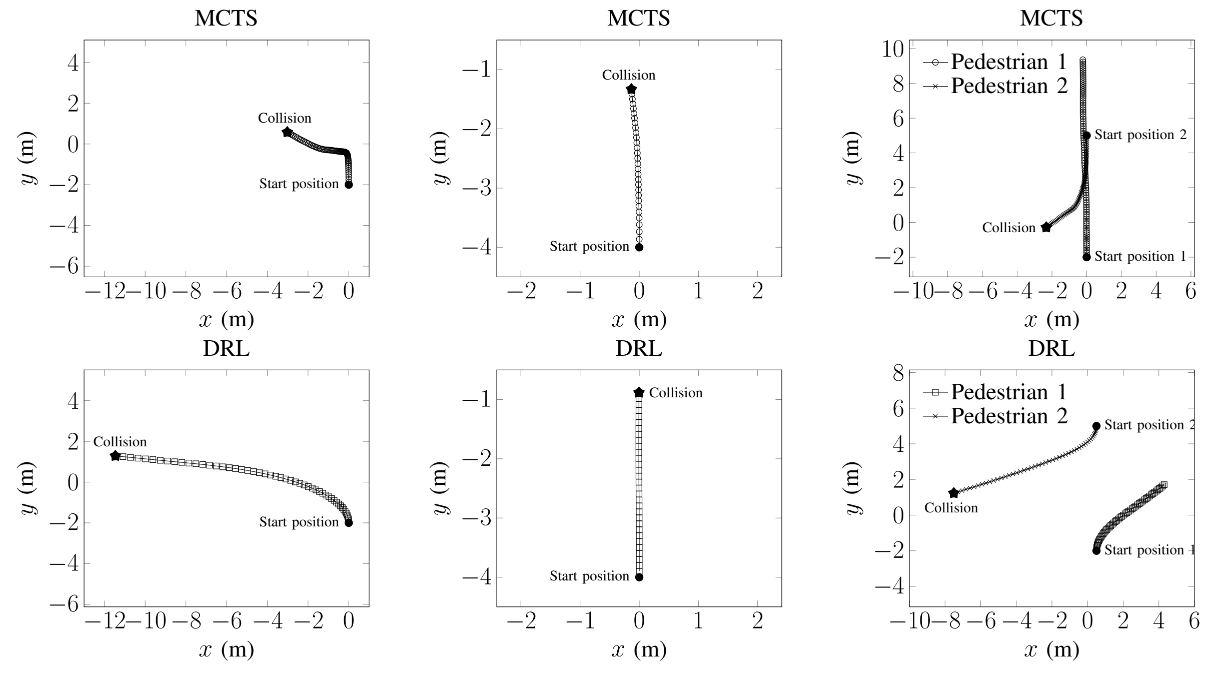
Figure 5 shows the pedestrian paths until the collision from each solver for scenarios 1, 2, and 3, respectively. The vehicle collides with the pedestrian at the collision point. In scenario 1, both solvers send a single pedestrian into the road, and have the pedestrian move towards the vehicle to create a collision. However, the turn towards the vehicle is much more pronounced in MCTS, where the pedestrian comes to a near stop, before angling hard left and into the vehicle. DRL instead settles on a smoother path. The DRL path is slightly more likely because there is less acceleration needed. Scenario 2 has similar paths from the solvers. The pedestrians start at a point from which their mean action should create a collision. Both solvers identify this path quickly and the pedestrians take very little action.

Both scenarios are relevant to scenario 3, which presents the largest difference. Because pedestrian 2 starts farther away from the vehicle, it has the more likely path to being hit by the vehicle as in scenario 2. In both solvers, the second pedestrian takes actions similar to scenario 1. However, there is a large difference in the first pedestrian. In MCTS, the pedestrian holds course for a bit before aggressively accelerating towards the other side of the road. In DRL, the first pedestrian takes a slight turn to the right, and then holds a constant path from there. MCTS has less ability to minimize the effect of the first pedestrian on the total reward since using a single seed results in coupling the actions of the pedestrians. Hence, the first pedestrian has a different and less optimal trajectory than its counterpart in DRL. In DRL, the first pedestrian had a change of direction at first, causing the second pedestrian to be closer to the vehicle. Then the first pedestrian maintained a course with very little acceleration, minimizing the pedestrian's effect on the reward.

In scenarios 1 and 3, the blame of the collision is on the pedestrian, which does not inform any modifications to the SUT. However, in scenario 2, the blame is on the vehicle, since it does not check for pedestrians approaching the crosswalk until the pedestrians are in the crosswalk, which gives very short response time for the vehicle. The suggestion for avoiding a collision like scenario 2 is to expand the sensing range of the IDM to go beyond the curb of the road. The reason AST returns situations where the blame is not on the vehicle is that we define the subset of state space that we are interested in E to be any collision. The kind of collisions reported by the examples shown in scenario 1 and 3 do not give the designer of the SUT any insight on how the SUT should be improved. The solution is to redefine the space of events of interest E to be the subset of collisions where the responsibility of the collision was on the SUT. The definition of E requires formal models of responsibility and blame in various road situations [24]. Incorporating these models in the AST framework is an area of future work.

TABLE I: Numerical results from both solvers. Reward without noise shows the reward of the MCTS path if sensor noise was set to zero, to illustrate the difficulty MCTS has with eliminating noise. DRL is able to find a more probable path than MCTS with a large reduction in calls to the STEP function.

	MCTS			DRL	
Scenario	Calls to STEP	Reward	Reward w/o noise	Calls to STEP	Reward
1	4.91×10^8	-131	-71	8×10^5	-62
2	1.85×10^6	-38	-15	8×10^5	-1.7
3	1.61×10^9	-161	-104	1×10^6	-52



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 5: Pedestrian motion trajectories for each scenario and algorithm. The collision point is the point of contact between the vehicle and the pedestrian. In scenario 3, pedestrian 1 does not collide with the vehicle.

V. CONCLUSIONS

This paper extended the adaptive stress testing methodology used before to test an aircraft collision avoidance system to autonomous vehicles. In addition, this paper introduced how to use deep reinforcement learning to improve the efficiency of adaptive stress testing. Deep reinforcement learning can find more-likely failure scenarios than Monte Carlo tree search, and it finds them more efficiently. Another contribution of this paper is a testing framework for autonomous vehicles that has modular components. By adapting one of the approaches, any manufacturer who wishes to test the sensor or decision system of a vehicle in simulation can use this framework. Further work will involve incorporating more realistic sensor and pedestrian models and imposing a tighter constraint on the definition of the events of interest.

ACKNOWLEDGEMENTS

The authors would like to express their gratitude towards Tim Wheeler for his helpful comments and his assistance in using the Automotive Driving Models simulator, which this work is built on.

REFERENCES

[1] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, no. Supplement C, pp. 182–193, 2016.
[2] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 2016-01-0128, pp. 15–24, 2016.
[3] W. Wachenfeld and H. Winner, "The new role of road testing for the safety validation of automated vehicles," in *Automated Driving*. Springer, 2017, pp. 419–435.
[4] W. Wachenfeld, "How stochastic can help to introduce automated driving," Ph.D. dissertation, Technical University Darmstadt, 2017.
[5] H. Winner, *Handbuch Fahrerassistenzsysteme*. Springer, 2015.
[6] D. Zhao, H. Lam, H. Peng, S. Bao, D. J. LeBlanc, K. Nobukawa, and C. S. Pan, "Accelerated evaluation of automated vehicles safety in lane-change scenarios based on importance sampling techniques," *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, vol. 18, no. 3, pp. 595–607, Mar. 2017.
[7] J. Woodrooffe, D. Blower, S. Bao, S. Bogard, C. Flannagan, P. E. Green, and D. LeBlanc, "Performance characterization and safety effectiveness estimates of forward collision avoidance and mitigation

systems for medium/heavy commercial vehicles," University of Michigan Transportation Research Institute, Tech. Rep. UMTRI-2011-36, 2012.

[8] H.-H. Yang and H. Peng, "Development and evaluation of collision warning/collision avoidance algorithms using an errable driver model," *Vehicle System Dynamics*, vol. 48, no. S1, pp. 525–535, 2010.

[9] Z. Huang, D. Zhao, H. Lam, D. J. LeBlanc, and H. Peng, "Evaluation of automated vehicles in the frontal cut-in scenario—an enhanced approach using piecewise mixture models," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 197–202.

[10] Z. Sunberg, C. Ho, and M. Kochenderfer, "The value of inferring the internal state of traffic participants for autonomous freeway driving," in *American Control Conference (ACC)*, 2017.

[11] T. A. Wheeler, M. Holder, H. Winner, and M. Kochenderfer, "Deep stochastic radar models," in *IEEE Intelligent Vehicles Symposium*, 2017.

[12] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, "Adaptive stress testing of airborne collision avoidance systems," in *Digital Avionics Systems Conference (DASC)*, 2015, pp. 6C2–1.

[13] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *Learning and Intelligent Optimization (LION)*. Springer, 2011, pp. 433–445.

[14] M. J. Kochenderfer, *Decision Making Under Uncertainty*. MIT Press, 2015.

[15] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[16] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1889–1897.

[17] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[18] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physics Review E*, vol. 62, pp. 1805–1824, Aug 2000.

[19] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *iclr*, 2013.

[20] California Department of Transportation, "California manual on uniform traffic control devices," pp. 682–685, 2014, Revision 2.

[21] J. Sklansky, "Optimizing the dynamic parameters of a track-while-scan system," *RCA Review*, vol. 18, no. 2, pp. 163–185, June 1957.

[22] P. C. Mahalanobis, "On the generalised distance in statistics," *Proceedings of the National Institute of Sciences of India*, vol. 2, no. 1, pp. 49–55, 1936.

[23] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning (ICML)*, 2016, pp. 1329–1338.

[24] 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.