Improving the Take-Over Request in Highly Automated Vehicles through Reinforcement Learning

Boran (Pan) Zhao*, Philip S. Thomas †

October 20, 2018

1 Introduction

There is an increasing interest in autonomous driving research and development. According to SAE, the level of driving automation spans from Level 0 (no driving automation) to Level 5 (full driving automation) [2]. While Level 1 (driver assistance) and Level 2 (partial driver assistance) have already realized in commercial vehicles, Level 3 (conditional driver assistance) is still under development. In Level 3 driving automation, the vehicle can drive by itself to a large extent, during which the driver does not need to keep monitoring the environment and is allowed to perform non-driving related tasks, such as making a phone call, reading a book, playing a game on a mobile phone, etc. However, the automation systems at this level still have limitations and cannot address some special circumstances beyond their capacities such as blocked lane by a construction site on a freeway, exiting a freeway, missing lane marks, etc. Under these circumstances, the driver needs to take over control of the vehicle. A take-over request (TOR) or a warning signal must be issued in this case to alert the driver and make him/her aware of the critical situation. However, it is not a simple question when the TOR should be issued when a critical situation is encountered.

A simple solution would be a using "fixed-threshold" strategy, in which a warning will be issued whenever some monitoring variable, e.g. *time to crash*, is lower than some threshold. This kind of strategy is commonly adopted in forward collision warning systems in automobiles. However, this strategy ignores the differences between drivers e.g. in response time and driving preferences, and may lead to a high false-alarm rate. To enforce safety for a large number of drivers, the threshold may need to be chosen as a relatively large value. However, this may be an issue for aggressive drivers who tend to have a small safety margin. In the view of some aggressive driver, some "dangerous" situations from the warning system's perspective can be safe. As a result, the driver may choose not to intervene even a warning signal is issued. In this case, the warning becomes a false positive (FP) warning. According to the literature, e.g. [1], false warnings will decrease the driver' trust in the warning system and may lead to driver's ignoring of the TORs in the future. On the other hand, for some defensive drivers or drivers with long response time, a TOR early enough is desired; otherwise, the driver may not have enough time to response, which will lead to high rate of near-collisions or even worse, collisions.

Therefore, the TOR system should be able to adapt to the personal preferences and abilities of a human driver. As a result, the fixed-threshold strategy may not work well. We propose to use reinforcement learning to customize the strategy for each specific driver.

In this report, we describe the modeling of the environment including the human driver and its implementation in Python. We also show how a reinforcement learning (RL) algorithm can be applied to the environment with safety guarantee. Simulation results based on a $Q(\lambda)$ algorithm are also included to illustrate the feasiblity of using RL for this problem.

*Pan Zhao is a PhD Candidate in Control Engineering Lab at the Unversity of British Columbia. Email:panzhao@mech.ubc.ca

†Philip S. Thomas is an Assistant Professor and co-director of the Autonomous Learning Lab in the College of Information and Computer Sciences at the University of Massachusetts Amherst. Email: pthomas@cs.umass.edu.

2 Formulation of the TOR as a RL problem

The agent-environment interaction is shown in Fig. 1. The state of the environment S_t at time step t is the predicted *time to crash* (TTC) and the confidence level for the prediction. The action of agent A_t at time step t is either *warn* or *not warn*. The environment consists of a human driver model, which describes the behavior of the driver. Different from convential RL setting, we include a safety supervisor to enforce safe guarantee during the learning process. Its details are explained in Section 4.1.1

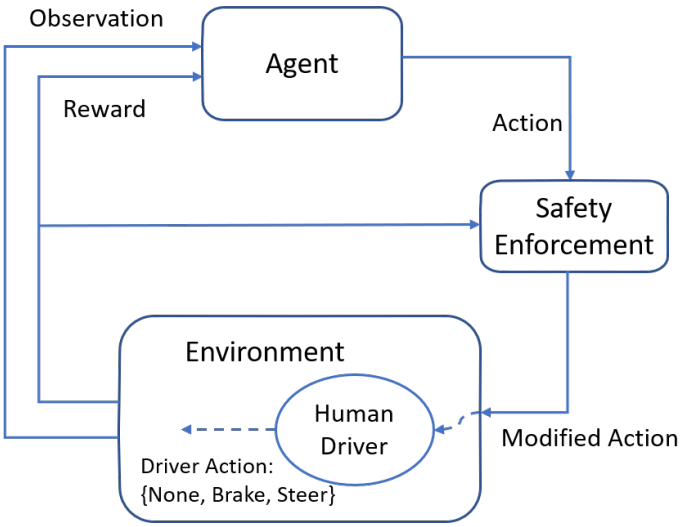


Figure 1: The agent-environment interaction in a Markov decision process

Modeling of the environment and its implementation will be explained next.

3 Environment Modeling and Implementation

We consider a scenario that a vehicle should drive on the middle lane of a three-lane freeway whenever possible . However, there are several blocked intervals due to construction sites or accidents, so the driver has to take over control of the vehicle and serve to the left lane before these blocked intervals. Once the vehicle passes a blocked interval, the driver will change to the middle lane and let the vehicle drive by itself again on it. According to the experiments reported in [3], drivers may choose to brake to a complete stop when confronting an obstacle, when the situation is extremely urgent. Therefore, we also allow the driver to choose to brake the vehicle to a complete stop in response to the TOR. The intervention way, i.e. swerve to the left or emergency brake, that a driver chooses depends on the current TTC. The larger is the TTC, the more probable it will be for the driver to choose to swerve to the left.

The environment named `TrafficEnv` is written to follow the style of OpenAI Gym. For instance, similar to the environments in Gym, `TrafficEnv` has a function `step` that simulates the execution of an action on the environment, and a function `render` to visualize the current state of the environment.

The main functions of `TrafficEnv` is listed in Table 1.

Table 1: Member functions of `TrafficEnv`

Function name	Purpose
`rest(self)`	Reset the environment to start a new episode
`new_state,reward,done,game_over = step(self,action)`	Simulate exection of `action` on the environment. `done` is `True` if one of the critical situations has been addressed or `game_over` is `True`. `game_over` is `True` if all critical situations have been addressed or a crash happens.
`render(self)`	Visualize the current state of the environment

3.1 Reward function

The reward value $r(t)$ at each step t is defined to be

$$r(t) = r_1(S_t) + r_2(a_{t-1}), \tag{1}$$

where $r_1(S_t)$ depends on the state S_t, and $r_2(a_{t-1})$ depends on a_{t-1}, i.e. the action of the agent at $t-1$. The mapping between S_t and $r_1(S_t)$ is given in Table 2. The numerical value in Table 2 can be further tuned.

Table 2: Mapping between S_t and $r_1(S_t)$

S_t	$r_1(S_t)$	Note
near-crash	-1	
crash	-10	
circumvent a blocked interval	10	Only evaluated when `done` is `True`
otherwise	0	

A near-crash happens when the longitudinal deceleration (due to brake) is over some pre-specified threshold, or the lateral acceleration (due to emergent steering) is over some pre-specified threshold, or the TTC is below some pre-specified threshold (such that it would be impossible for the driver to respond properly to the critical situation).

For $r_2(a_{t-1})$, we simplify penalize each TOR by setting $r_2(a_{t-1})$ to be a negative value (e.g. -0.4) whenever a_{t-1} is `WARN` and to be zero otherwise. The purpose of doing this is to minimize the false positive warnings. Due to the penalty on each warning, the agent should learn to use as few warnings as possible while still minimizing the near-crashes and crashes.

3.2 Driver model

The driver model describes the behavior of the driver in the whole driving process, including the time when the car is autonomous driving mode. We designed several modes for the driver, as listed in Table 3.

Table 3: Driver modes and their meanings

Driver Mode	Driver Behavior
STOP	Brake the vehicle until it stops. Keep the brake afterward.
DRIVE TO SPEED	Control the brake and acceleration intensities (with a P controller) to drive the car towards a target speed.
PERCEIVE AND DECIDE	Perceive the driving environment after acknowledging a TOR and make a decision at the end whether and how to intervene.
BE DISTRACTED	Perform some non-driving related tasks while the car is in autonomous driving mode.
FOLLOW WITH SAFE DISTANCE	Control the brake and acceleration intensities (with a PD controller) to follow a leading vehicle with a pre-specified distance. Currently not in use.
SWERVE TO LEFT	Control the steer intensity (with a PD controller) to make a lane change to the left.
CHANGE TO RIGHT	Control the steer intensity (with a PD controller) to make a lane change to the right.
EMERGENCY BRAKE	Apply an emergency brake (with a constant brake intensity).

The flow between different modes is shown in Fig. 2.

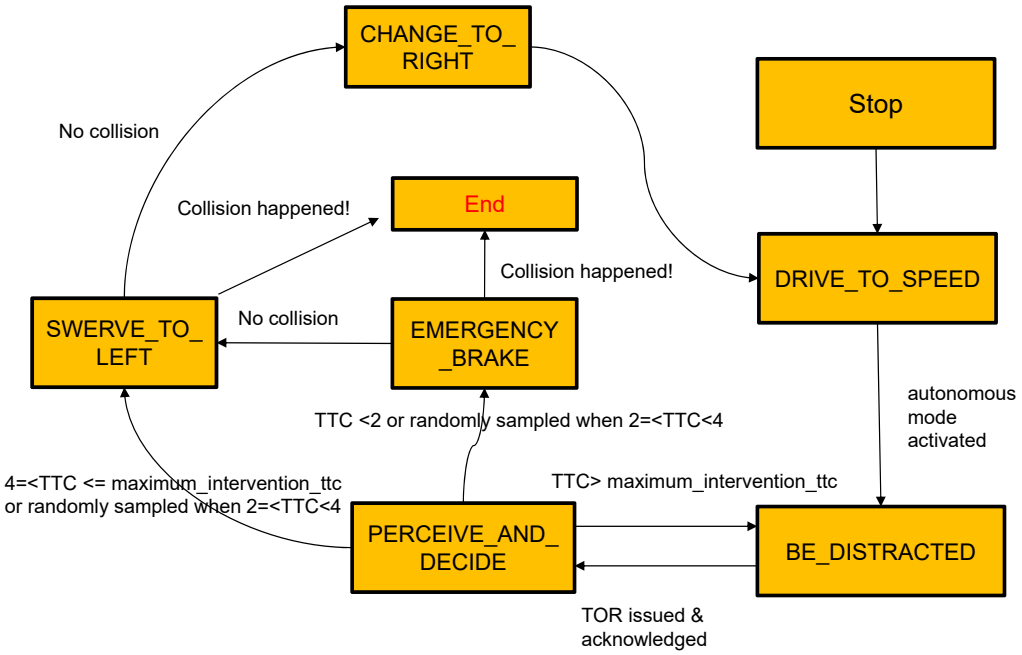


Figure 2: Flow of driver modes

To represent the effect of false warning on the driver's trust in the TOR system, we model the probability of the driver to acknowledge a TOR as

$$p_f(t) = 1 - \frac{\#\text{ false positive warnings}}{\#\text{ total warnings}+1}. \tag{2}$$

Once the driver acknowledges a TOR, he/she will start to perceive the environment and then decide whether to take over control of the vehicle; otherwise, the driver ignores the TOR and stays distracted.

Other properties specific to each driver include the `response_time_bounds` and `maximum_intervention_ttc`. The former will be used to sample a response time, which is the time from acknowledging a TOR to

initiating an intervention action. The latter represents the aggressive level of a driver. The smaller is the `maximum_intervention_ttc`, the more aggressive the driver is.

4 Learning with safety guarantee

In this section, we will explain how a RL algorithm can be applied to the environment described above with safety guarantee.

4.1 Deal with the semi-MDP problem by episode cutting

Note that the car can be in autonomous mode and driver-intervention model. The agent's actions are only selected in autonomous mode; however, the effect of its actions will span the driver intervention mode. To apply the regular RL algorithms, we cut one episode at the end of driver intervention into sub-episodes, as shown in Fig. 3. Furthermore, we treat the whole process of driver intervention in each sub-episode as a terminal state for this episode. The reward at this terminal state is the summation of the discounted rewards in the whole driver-intervention process. Assume the driver intervention starts at time $t+1$ as shown in Fig. 3, the reward at the terminal state will be

$$\tilde{r}(t+1) = r(t+1) + \gamma r(t+2) + \cdots + \gamma^{N-1} r(t+N), \tag{3}$$

where γ is the discount factor and $r(t+i)$ is defined in (1). Note that when using (1), $r_2(a_{t+i-1}) = 0$ for $1 < i <= N$, where $t+N$ is the last time step for driver intervention.

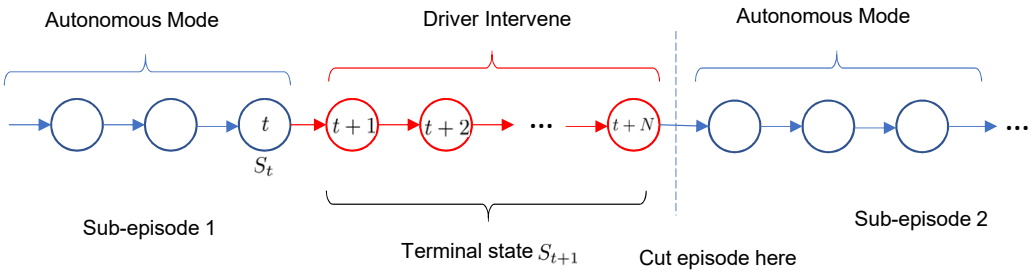


Figure 3: Handle the alternation between autonomous model and driver-intervention model by episode cutting

Although we treat the sub-episode as an episode when applying RL algorithms, we do not reset the environment at the end of each sub-episode. Instead, we only reset the environment when the whole episode terminates, i.e. when `game_over` in Table 1 is `True`. This is similar to how Google DeepMind treated the episode when they trained the deep RL algorithm for playing Atari games. In their setting, a sub-episode terminates when a life is lost while an episode terminates when all lives are lost. Furthermore, the gaming environment is reset only when the whole episode terminates.

4.1.1 A mechanism to enforce safety

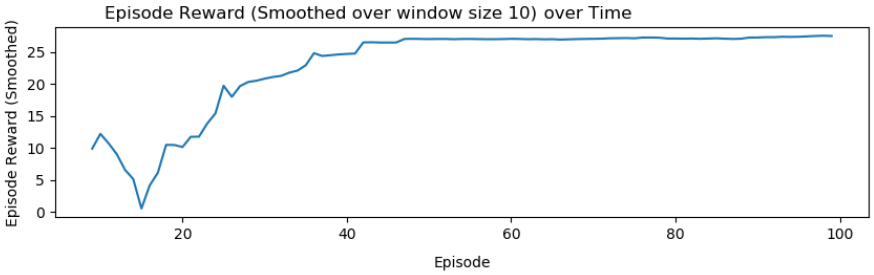
When training the agent, it can happen that even when the TTC is already critically small, the agent may still choose not to issue the TOR. This may lead to a dangerous situation, when the driver will not have enough time to respond. As a result, a crash will have a high probability to happen. To enforce the safety in the learning process, we introduced a *safety supervisor*, as shown in Fig. 1, a variable, `min_ttc_for_safety`. Here safety refers to that no crash will happen at any time. Note that near-crash is still allowed; otherwise, the variable may be too conservative that will lead to high false positive warnings for many drivers. The safety supervisor will keep monitoring the TTC and the agent's action. When TTC is smaller than `min_ttc_for_safety`, if the agent's action is `NOT WARN`, the supervisor will overwrite this action to be WARN. We assumed that a *false negative* (FN) warning happens when this overwriting happens, and a penalty will be applied to the

agent's action at the corresponding state. Eventually, the agent should learn to be away from the safety boundary defined by `min_ttc_for_safety`.

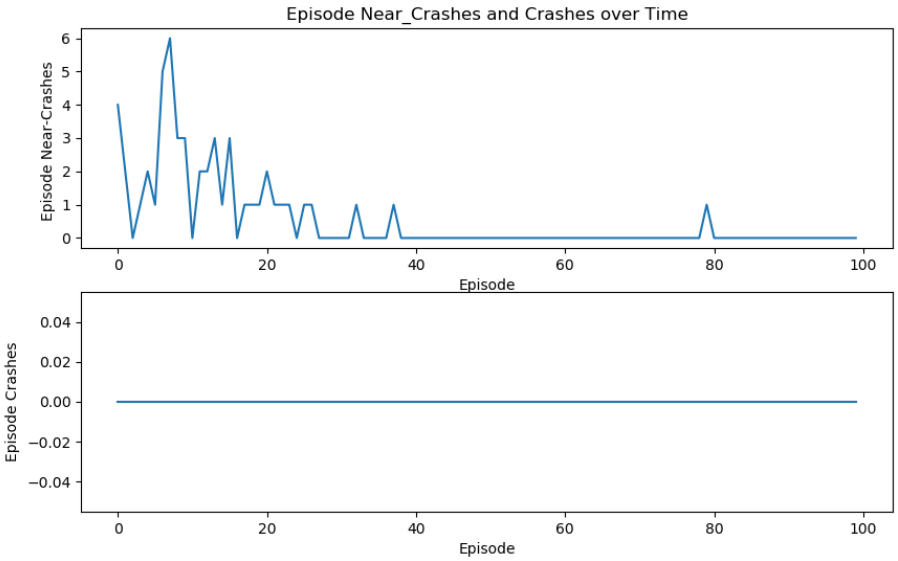
5 Simulation results using actor critic with eligibility traces

Note that most of the RL algorithms can be applied to the TOR problem we have explained above. Here we just tested an actor critic algorithm with linear function approximation and eligibility traces. Linear function approximations were employed to describe both the value function and the policy, where Fourier basis was used to generate the feature vector. For training the RL agent, we selected $\lambda_\theta = \lambda_w = 0.8$, the discount factor $\gamma = 1$ and the learning rate $\alpha_\theta = \alpha_w = 0.002$. The order of the Fourier basis was selected to be 5. For the driver model, we selected `maximum_intervention_ttc` to be 6 seconds. The training results are shown in Fig 4. Due to the safety guarantee mechanism, there is no crash happening in the whole learning process, although there are some near-crashes at the beginning.

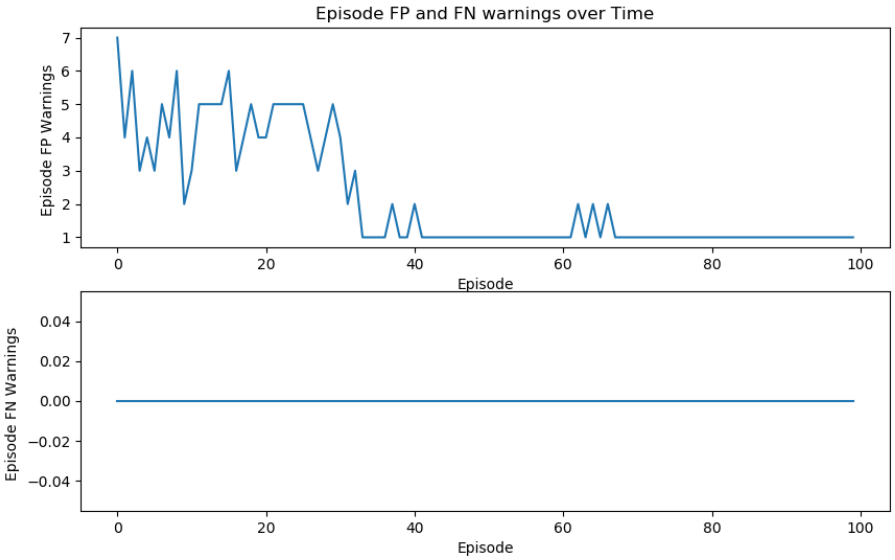
A video demo is available here.



((a)) Accumated reward at each episode



((b)) Number of crashes and near-crashes at each episode



((c)) False warnings at each episode

Figure 4: Training results

6 Future work

Currently, one of the state variables, confidence level, is constantly set to 1 in the simulation. More realistic simulation of this variable can be done in the future, by associating this variable with the distance to the critical situation or TTC. One can possibly assume that the confidence level increases with the decrease of distance to the critical situation or TTC.

More work can still be done to further reduce the FP warning rate. For instance, one can use the safety supervisor to enforce the modified action to be `NOT WARN` whenever the TTC is larger than some pre-specified value.

References

[1] J. P. Bliss and S. A. Acton. Alarm mistrust in automobiles: how collision alarm reliability affects driving. *Applied ergonomics*, 34(6):499–509, 2003.

[2] S. International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, 2016.

[3] L. Lorenz, P. Kerschbaum, and J. Schumann. Designing take over scenarios for automated driving: How does augmented reality support the driver to get back into the loop? In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 58, pages 1681–1685. SAGE Publications Sage CA: Los Angeles, CA, 2014.