Continuous Safe Learning Based on First Principles and Constraints for Autonomous Driving

Lifeng Liu[1], **Yingxuan Zhu**[1], **Tim Tingqiu Yuan**[2], **Jian Li**[1]
[1] Futurewei Technologies, Inc.
111 Speen St., Framingham, MA, 01776, USA
{lifeng.liu, yingxuan.zhu, jian.li}@futurewei.com
[2] Huawei Technologies
ytq@huawei.com

Abstract

We propose a continuous learning system for autonomous driving based on first principles and constraints. The learning system includes a knowledge base to represent the first principles' manifestations; Initial vehicle control commands are derived from the knowledge base, and constrained exploration and sequence analysis based intelligent exploitation are used to refine the control commands and keep the learning safe. A decomposed learning curriculum is utilized to enrich the knowledge base and handle complicated scenarios. Our experiments in simulations and real test environments show the system's advantages of fast learning, smooth driving, and quick adapting. The proposed system does not require the complicated vehicle model calibration, and facilitates the easy transfer learning from simulation environments to real driving environments.

1 Introduction

Recently a number of self-driving projects use deep learning for part of the driving tasks (Bojarski and et al. 2016; Rao and Frtunikj 2018; Hecker and et al. 2018), but there are more challenges in interperability of the models, safety issues, and transfer learning problems. More recently, (Zhou, Krähenbühl, and Koltun 2019) compared supervised imitation learning of driving using raw images (from pixels to actions) and using representation of scene content produced by computer vision, and showed that intermediate representations have the benefits of modularization and abstraction (Müller and et al. 2018), and generalize better to unseen environments. More researches utilize deep reinforcement learning, which builds a reinforcement learning architecture on deep neural networks (Arulkumaran and et al. 2017). Reinforcement learning has been used in robot control and autonomous driving simulations. However, many of them are based on a try-and-error approach, and generally require longer learning time. In addition, real autonomous driving environments can not withstand the errors (accidents) in the procedure of the general reinforcement learning, and runtime safety monitoring using constraints is required to meet the safety standards (such as Automotive Safety Integrity Level (SAIL) defined in ISO26262).

One kind of deep reinforcement learning is deep deterministic policy gradients(DDPG) (Lillicrap and et al. 2016; Silver and et al. 2014), which only integrates over the state space, and can handle continuous action space. However, the difficulty of transfer learning still exists, such as transferring the learned model of autonomous driving from simulation environments to the real vehicle on the real test roads.

Guided Policy Search (GPS) (Levine and et al. 2013; 2015) is a supervised learning method for training complex nonlinear decisions. However, it requires to reset the state of the environment each time when a local trajectory is generated. Or you need to generate a large number of samples from the global policy network. Policy Learning using Adaptive Trajectory Optimization (PLATO) (Kahn and et al. 2017) is a continuous, reset-free reinforcement learning method. PLATO uses model predictive control (MPC) to generate the supervision, but requires accurate dynamic system model. Trust Region Policy Optimization (TRPO) (Schulman and et al. 2015) directly constructs stochastic neural network policies without decomposing problems into optimal control and supervised phases, but requires an initial safe policy and large amount of data, and is significantly less data efficient.

On the other hand, driving control based on accurate physical models and rules requires time-consuming and laborious system calibration for the control parameters, which may take months to get the complete calibration for a vehicle. Imitation learning (Liu and et al. 2018; Ng and Russell 2000; Song and et al. 2018) is based on the behavior of human beings. In autonomous driving, imitation learning has many disadvantages, including difficulty in training all scenes, especially dangerous scenes/corner cases. Under some scenarios, human behaviors have limitations, not necessarily are the best. (Chen and et al. 2019) decomposed the learning problem into two stages: A privileged agent cheats by observing the ground-truth layout of the state; The privileged agent acts as a teacher that trains a purely vision-based sensorimotor agent. Their trained system achieved higher performance than direct imitating learning.

To deal with the challenges of machine learning for autonomous driving, we propose a new modularized reinforcement learning (RL) framework based on first principles and constraints to control the autonomous system safely in continuous learning. The system architecture overview is shown

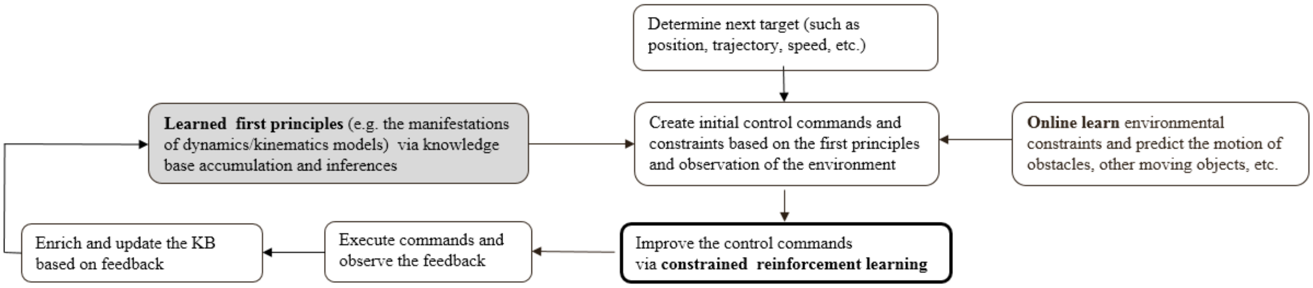

Determine next target (such as position, trajectory, speed, etc.)
Learned first principles (e.g. the manifestations of dynamics/kinematics models) via knowledge base accumulation and inferences
Create initial control commands and constraints based on the first principles and observation of the environment
Online learn environmental constraints and predict the motion of obstacles, other moving objects, etc.
Enrich and update the KB based on feedback
Execute commands and observe the feedback
Improve the control commands via constrained reinforcement learning

Figure 1: Overview of the system architecture.

in Figure 1: Learn the environmental constraints via perception and motion prediction; Determine the next driving target, such as the relative/absolute position, trajectory, speed, etc.; Query the knowledge base (KB) using the understanding of the environment and the determined driving target, and create the initial control commands and constraints; Improve the control commands via intelligent constrained reinforcement learning; Execute the commands and collect feedback for enriching and updating the knowledge base.

Our experimental evaluation shows that our learning system can significantly reduce the learn time (from month/days to hours), and make the transfer learning more efficiently and safely in adapting from simulation environments to real driving scenarios.

The following section will talk about the details of the learning system: The initial control commands and constraints are generated (Sec. 2.2) based on the knowledge base with the first principles' manifestations (Sec. 2.1); The constrained exploration and intelligent exploitation are used for improving the control commands (Sec. 2.3); The observation and feedback are used for enriching the knowledge base via following a learning curriculum (Sec. 2.4).

2 System Design

2.1 Creating Knowledge Base for Deriving Control Commands

One example table in the knowledge base is shown in Table 1. The content of the initial knowledge base can be derived from the car manufacture's specifications, or can be safely obtained by driving the vehicle in the parking lot or a training site as follows:

- Set the vehicle to the expected speed, then send the corresponding deceleration commands and measure the distance and time to stop.
- Set the vehicle to the desired speed, then send the appropriate steering wheel angle command and measure the curvature of the path the vehicle is passing.

The collected driving samples can be used to estimate the structural parameters (mass, wheelbase, center of gravity) of the vehicle. The knowledge base keeps the manifestations of the first principles of driving, and provides the functions of reasoning, fitting, and querying, such as creating the initial control commands. Knowledge base query functions have states and KB as inputs, and output the suggested candidates for vehicle control commands. For example, when driving at different speeds, query the appropriate steering angles under different road curvatures:
$expectedSteeringAngle = QuerySteeringAngleFromKB$
$(roadCurvature, currentSpeed)$

2.2 Generating Constraints for Safe Learning

Constraints are used to avoid dangerous situations:

- Hard constraints:
  - Maintain safety distance/safe response time from the obstacles/other moving objects in the environment.
  - Prevent slipping and derailment.
  - Keep operating commands within the operating range of the system (acceleration/braking limits, matching between steering angle and speed, etc.)
- Soft constraints:
  - Maintain comfort and stability.
  - Maintain fuel efficiency.

Constraints can be expressed as boundaries of control commands and are used to limit the exploration space in the RL, as shown in Table 2. The generation of constraints can be independent of the learning algorithm, and the constraints will be updated in real time according to changes in the environment and state. Therefore, constrained exploration avoids the limitations of the predetermined constraints and loss functions in terms of flexibility and scalability. As shown in Figure 3, if the action will result in driving out of lane, the action will not be explored.

2.3 Intelligent Constrained Reinforcement Learning

In real environments of autonomous driving, we cannot afford errors/accidents from continuous learning. Incomplete training/learning will lead to potentially unsafe strategies and cannot handle scenarios that are not learned or unseen. Including corner cases can greatly increase the variance of the reward function and make converging more difficult. To solve these problems, We propose an efficient exclusion and search algorithm, which is capable of transfer learning. The flow chart of the proposed reinforcement learning is illustrated in Figure 2 , and including:

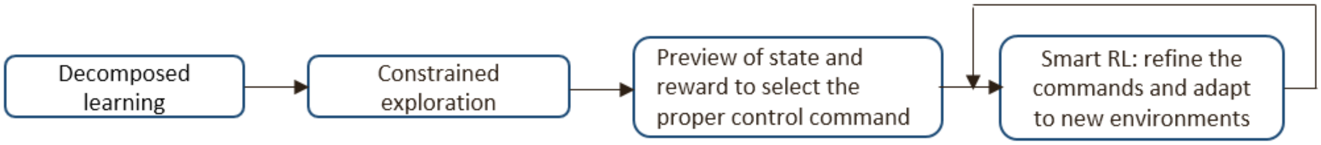

Decomposed learning
Constrained exploration
Preview of state and reward to select the proper control command
Smart RL: refine the commands and adapt to new environments

Figure 2: Flow chart of smart RL

Table 1: Example content of knowledge base

Speed	Steering angle	Curve radius of the lane	Vehicle orientation changing rate
50km/h	2 degrees	100m	0.2 degrees/s

Table 2: Examples of representation of state based constraints

State	Constraint	Based on
Lane curvature	Upper limit of speed	Road friction coefficient and first principles of vehicle motion
Distance to the ahead vehicle		
Speed of the ahead vehicle		Safety distance / response time (e.g. 3 second rule)
Distance to the behind vehicle	Lower limit of speed	
Speed of the behind vehicle		
Speed limits of the lane		Safety distance / response time
Road curvature	The average steering angle.	First principles of vehicle motion

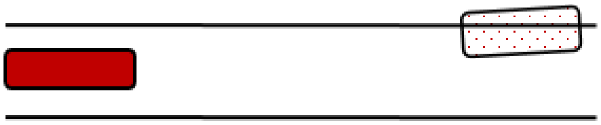


Figure 3: An example of constraints of following lane.

- Decomposed learning: For example, learn speed control and steering control separately (e.g. learning the steering control at a fixed speed). Furthermore, the steering control may include the steering control of following the curved lane, the steering control of the lateral speed (to approach the next target from current position), and the steering control of vehicle orientation (to keep the vehicle orientation the same as the lane direction). These steering controls can be learned separately and used with the proper combinations.
- Preview of state and reward for optimization: Based on the first principles' manifestations in the knowledge base, for each operation command candidate, preview the state/reward of the host vehicle at next step(s). Estimated reward is based on the result of *computeNextCarPose(KB,currentActions,currentState)*. See details in Algorithm 2.
- Constrained exploration and intelligent exploitation: Constraints are used to reduce the exploration space; Control command selection must be within the boundaries represented by the constraints (such as the speed limits derived from the steering angles). Based on the tried operation commands and the observed effects, sequence analysis is used to compute the gradient direction to generate better operation command candidates.

The detailed description of the intelligent constrained RL is in Algorithm 1, where Step 1 is for generating control commands candidates; Step 2 and Step 3 are for constrained exploration (using preview of state and reward, i.e. looking ahead). For example, if a steering action will result in running off the road in a few seconds, that action will be ruled out from the exploration; Step 4 is for transfer learning and adapting to new environment; Step 5 and Step 6 are for updating the real rewards and adjusting the actions in the correct direction for next try based on feedback; Step 7 is for generating better commands based on the results from step 6.

An enhancement of Step 4 (When KB accumulates to a certain extent): Preview the state and reward of the commands based on KB, and the algorithm is described in Algorithm 2.

An enhancement of Step 6 is listed in the following: Check overshoot/undershoot based on a sequence of ¡action, result¿ so as to reduce the influence of delayed execution of the control commands, make better use of the delayed rewards, accelerate the learning, and improve the smoothness of driving. For example, Algorithm 3 describes how to learn

Algorithm 1 Algorithm of intelligent constrained RL

```
Inputs:state,Constraints,KB
output:Refined control command.
Step 1: InitialCommand =KB.findInitialCommand(state)); Extend InitialCommand to a group of command candidates:$S_{commands}$. For example, use a group of scale factors (e.g. 0.8, 0.85, 0.9, ......,1.2) to scale InitialCommand.
Step 2: Check the command candidates, only keep the command candidates satisfying the constraints.
  for all $command$ within $S_{commands}$ do
    for all $constraint$ within Constratints do
      if KB.satisfy($command$, $constraint$) is false then
        Remove $command$ from $S_{commands}$
      end if
    end for
  end for
Where KB.satisfy($command$, $constraint$) might be implemented based on preview of the states for the next couple of timestamps, described in the following:
  for all $t$ within [1, n] do
    hostCarPoseTimeT = KB.computeHostCarPose($t$, $command$)
    obstalePoseTimeT = KB.computeObstaclePose($t$, $constraint$)
    if notSafe(hostcarPoseTimeT, obstalePoseTimeT) is true then
      return false;
    end if
  end for
  return true;
Step 3: If state is a learned state:
  Find the $bestCommand$ from $S_{commands}$ with the best estimated reward;
  if $bestCommand$ is marked with how to refine for further trials (see Step 6) then
    Adjust $bestCommand$ along the associated gradient direction;
    Output the adjusted command;
  else
    Output $bestCommand$;
  end if
Step 4: If state is a new state (not learned before), find a learned state nearby, and make use of its learning result:
  for all dimension within $state$'s Dimensions do
    $testState$ = $state$
    $testState$.dimension.value +/-= threshold
    if islearned($testState$) is true then
      return ($bestActionIndex$) of $testState$
    end if
  end for
Based on the returned index, output the corresponding (or the nearest) command in $S_{commands}$.
Step 5: Observe the execution result of the chosen command, and update the corresponding rewards related with the command (e.g. update Q learning table or SARSA table).
Step 6: Based on the observation result, adjust the actions.
  if the result is undershoot(e.g. reward is negative) then
    mark the selected action to increase in next trial.
  end if
  if the result is overshoot (e.g. from ”left to center line” to ”right to center line”, as shown in Figure 4) then
    mark the selected action to decrease in next trial.
  end if
Step 7: If needed, adjust or refine the generation of candidate control commands (to be more centered around the best action).
```

following lane with a specified speed in a curved lane: Using steering control as an example, if the selected steering angle causes overshoot, then next time the adjustment will be reduced (and vice versa). Therefore, the benefits of intelligent RL include adapting to the new environments: Learn from the previous states to help with the selection of operational commands in the new states (avoid random command selection). For example, the learning results at low speed are

Algorithm 2 Enhancement of Step 4 in Algorithm 1

```
for all $command$ within $S_{commands}$ do
    reward= KB.computeNextReward($state$, $command$)
    if reward > otherRewards then
        keep $command$
    end if
end for
Output the command with the best reward.
```

Algorithm 3 Enhancement of Step 6 in Algorithm 1

```
For a sequence of $(steeringAngle, nextState)$ on a lane with the same curvature, find the variance of steering angles:
if $variance > varianceThreshold$ then
    Calculate the medium (or average) value of the steering angles (expectedValue)
    for $steeringAngle$ within the sequence of $(steeringAngle, nextState)$ do
        if $steeringAngle < expectedValue$ then
            mark as increasing for next trial
        end if
        if $steeringAngle > expectedValue$ then
            mark as decreasing for next trial
        end if
    end for
end if
```

applied to the command generation at high speed; Adjustments when turning a sharp curve are based on experiences of turning smooth curves.

After the knowledge base is rich enough in information, the algorithm can be used as a "teacher" policy for supervised deep learning or "teacher" based self-play reinforcement learning (to replace or further optimize the knowledge base's fitting function: from linear fit to non-linear fit). This self-play based learning is described in Algorithm 4.

Algorithm 4 Self-play learning algorithm

```
Initialize training data D to be empty
for $i = 0$ to $N$ do
    for $t = 1$ to $T$ do
        Sample action $a_t$ for state $s_t$ based on the teacher's policy
        Append the new ($s_t$, $a_t$) to D
        State evolves to $s_{t+1}$ according to KB
    end for
    Train learner's policy on D
end for
```

2.4 Continuous Online Learning and KB Accumulation/Optimization

The study course of establishing/improving the knowledge base is illustrated in Figure 5:

- Initialize KB from samples of human driver's driving data.

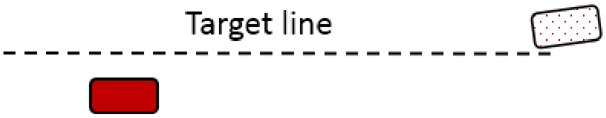


Figure 4: Example of overshooting.

- Learn basic tasks first before learning combination tasks; learn simple tasks before learning complex tasks; and learn the prerequisites for tasks before learning the tasks themselves.
- New control commands are based on experiences learned from old behaviors; explore in the state/action space, and gradually improve the knowledge base (no sudden jumps in states/actions during learning).
- Online learning: observed actions/effects are consistently used to improve the reasoning power of the knowledge base.

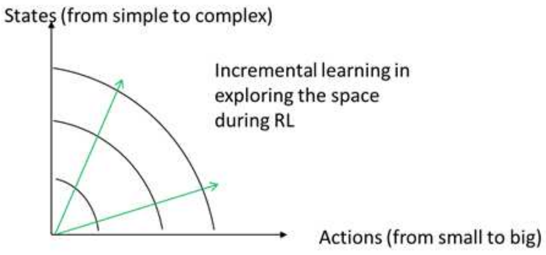

States (from simple to complex)
Incremental learning in exploring the space during RL
Actions (from small to big)

Figure 5: Learning and knowledge accumulation

Online learning will help adapting to the environmental changes and the vehicle's performance changes, and once the KB becomes more matured, online learning may only occur occasionally for rare/corner cases.

3 Implementation of the Learning Using KB

In our implementation, the sensors on the vehicles get the perception of the surrounding environment (safety and performance validation of the perception algorithms are beyond the scope of this paper), and then the system learns how to generate appropriate driving control commands (such as the steering wheel angle, acceleration/deceleration, etc.) according to the observed environments and the determined driving target which is decided by a planning module (a separate module which is beyond the scope of this paper).

To learn the commands of controlling the steering wheel angle and acceleration/deceleration in following lanes (such as high-speed cruise), the calculation of the initial steering angle is based on the following components:

- $angle0$ based on the distance from the host vehicle to the lane center and a look ahead distance. For example, $angle0 = atan(\frac{distanceToLaneCenter}{lookaheadDistance})$.

Table 3: Model simplification for car acceleration and braking

Simplified	Original	
$F_r = c_1 * m * g$	$F_r = r_r * m * g$	Rolling resistance force depends on the resistant parameter (r_r), mass (m), and gravity coefficient(g).
$F_d = c_2 * v^2$	$F_d = \frac{1}{2} * C_d * A * \rho * v^2$	Air resistance force depends on the coefficient of friction (C_d), car shape/cross sectional area (A), air density (ρ), and speed (v).
$F_w = c3 * \%\ throttle$	$F_w = \frac{T_B\ *\ R *\ g_k}{d\ /2}$	Forward/drive force controlled via throttle: the engine torque (T_B), the gear ratio (g_k), the differential ratio (R), and the wheel diameter (d).
$a = \frac{F_w - F_r - F_d}{m}$		Acceleration is based on the above forces.
$F_b = c4\ *\ BF_w$	$T = \frac{BF_w\ *\ R}{r}$	Brake torque depends on the braking force for the wheel (BF_w), the wheel radius (R), and the speed ratio between the wheel and the brake.

- $angle1$ based on the Lane's curvature. It can be queried from the knowledge base, or calculated from simplified formula. For example, $angle1 = asin(\frac{distanceFrontWheelToMassCenter}{roadCurveRadius})$.
- $angle2$ which is the host vehicle's orientation relative to the lane direction.

The calculation of initial acceleration/deceleration is based on steering constraints caused by lane curvature, speed limit from speed limit sign, and limitations for passengers' comfortness.

To learn the acceleration/deceleration control commands when following another vehicle (e.g. low speed following) so as to maintain the desired distance (as shown in Figure 6), the following factors are considered:

- The difference between the current distance and the ideal distance;
- The speed difference between the two vehicles;
- The difference in acceleration between the two vehicles;

Via data mining in KB, the relationship between vehicle model parameters and vehicle control parameters are established. Based on the data accumulation, the vehicle model parameters are calculated using simplified dynamics/kinematics formulas, and used for generating the control commands. Table 3 displays the simplification of the formulas for acceleration and braking computation. Therefore, the number of unknown parameters (e.g. $c1$ to $c4$ in Table 3) to learn is reduced.

For steering control, steering wheel angle(δf) is set properly according to $normalizedSteeringAngle$ (based on $angle0$ to $angle2$ mentioned earlier), and the derived knowledge in KB is two-way mapping:
$f(normalizedSteeringAngle) = \delta f$
$normalizedSteeringAngle = f^{-1}(\delta f)$

4 Experimental Results

Intelligent constrained RL optimizes the control commands for the vehicle, and is capable of compensating the inaccuracy in estimation of the first principles of vehicle motion, and conducts constrained reinforcement learning through self-guided exploration and exploitation.

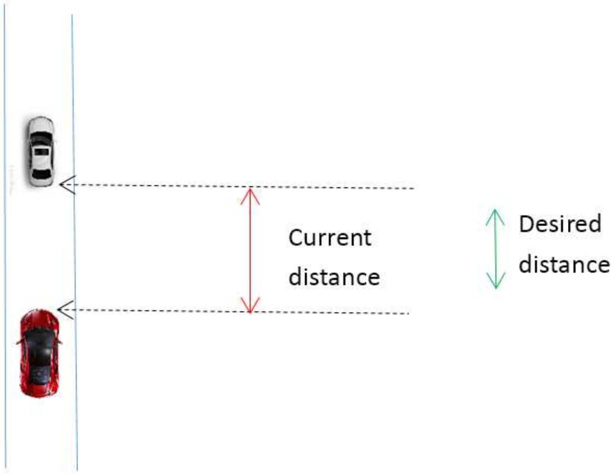


Figure 6: Illustration of following car.

We tested the learning of following lane in TORCS simulation environment (Wymann and et al. 2014) (an open racing car simulator). As shown in Figure 7, after a short period of learning (less than an hour), the vehicle can remain on the centerline of the road even under high speed (70km/h) and on curved roads. In contrast, DDPG based learning takes days to correct its mistakes (Lau 2016). We also tested learning of following lane and following car in the SCANeR Studio simulation environment (AVSimulation), which has more powerful simulation capability for different road types, traffic landmarks, and vehicle types, etc. Users of SCANeR can also create new scenarios based on the real road situation, and modify the vehicle settings to match the real test vehicles. Figure 8 shows the learned car follows the traffics in SCANeR. Our sequence analysis algorithm (Algorithm 3) achieved more smooth driving in curved roads (curve radius varies from 30 meters to 100 meters) under higher speed (70 -80km/h) after about half an hour of learning. Our supple-

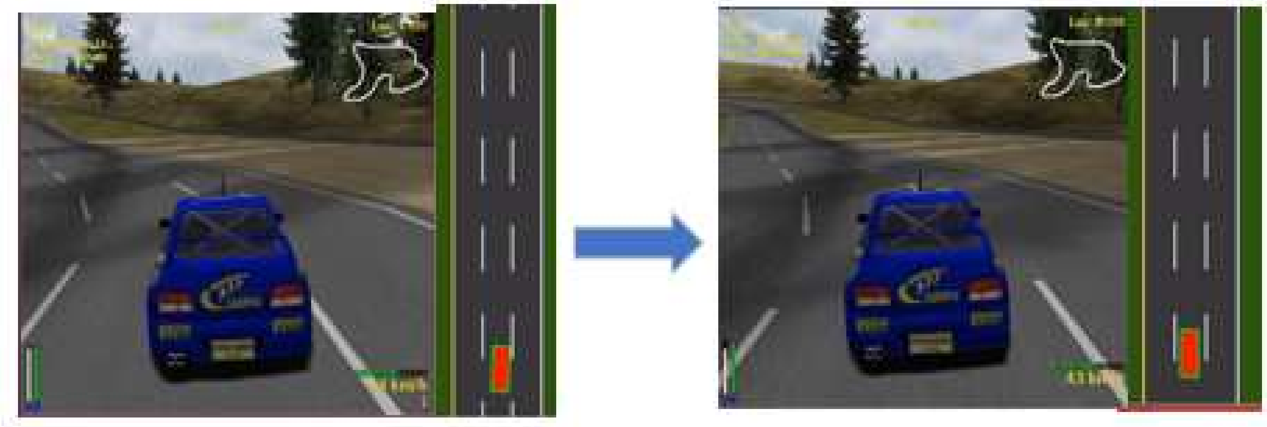
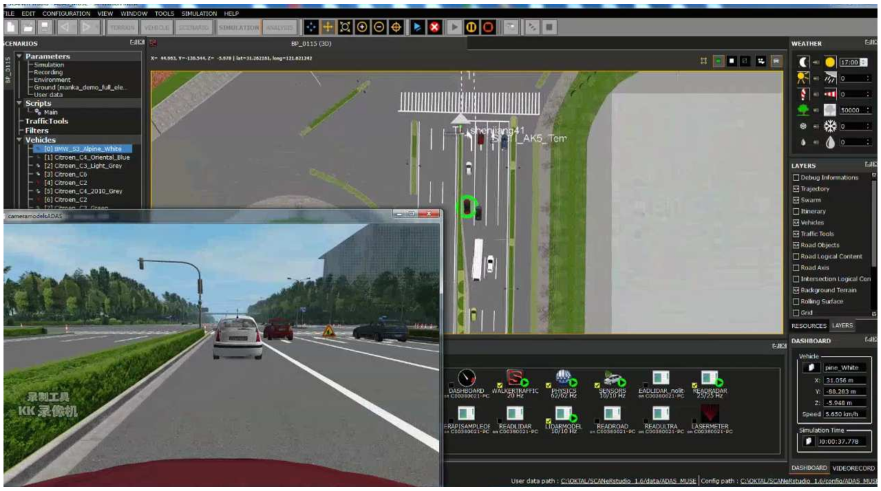


Figure 7: Learning following lane in TORCS simulation environment. Left:Before the learning; Right:After the learning.



Parameters
- Simulation
- Recording
- Environment
- Ground (manka_demo_full_ele...
- User data

Scripts
- Main

TrafficTools

Filters

Vehicles
- [0] BMW_53_Alpine_White
- [1] Citroen_C4_Oriental_Blue
- [2] Citroen_C3_Light_Grey
- [3] Citroen_C6
- [4] Citroen_C2
- [5] Citroen_C4_2010_Grey
- [6] Citroen_C2

☐ Debug Informations
☒ Trajectory
☒ Swarm
☐ Itinerary
☒ Vehicles
☒ Traffic Tools
☒ Road Objects
☐ Road Logical Content
☐ Road Axis
☐ Intersection Logical Con
☒ Background Terrain
☐ Rolling Surface
☐ Grid

Vehicle: pine_White
X: 31.056 m
Y: -88.283 m
Z: -5.948 m
Speed 5.650 km/h
Simulation Time 00:00:37.778

User data path : C:\OKTAL/SCANeRstudio_1.6/data/ADAS_MUSE | Config path : C:\OKTAL/SCANeRstudio_1.6/config/ADAS_MUSE

Figure 8: SCANer test environment

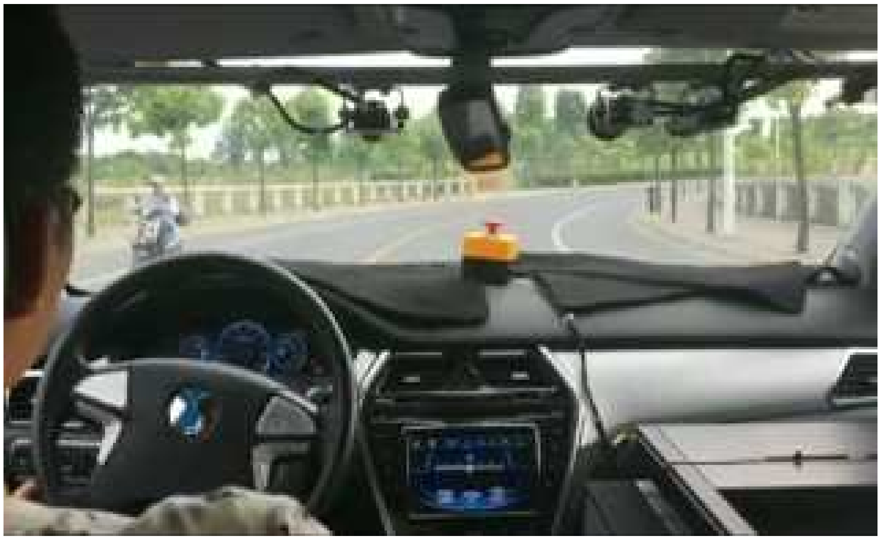


Figure 9: Following lane in real road

mental materials will provide more video clips of the testing.

In TORCS simulation, the state information (speed, distance to lane boundaries, lane curvatures, etc.) is obtained from the simulation environments. In SCANeR studio and real environments, image and Lidar based perception system (based on proprietary software package) provides the state information.

We further tested our learning system using real car on real roads by transferring the learned knowledge base from the simulation environment to the real car, and achieved fast transfer learning without time consuming mechanical re-calibration. The self-driving on real road was smooth in its first try. Figure 9 shows the following lane in real road with speed about 45km/h. Experiments of following car, and following traffic lights were also conducted using real car in our test field.

5 Conclusion

The existing machine learning technologies have limitations in autonomous driving, such as the "trial and error" reinforcement learning method takes a long time (a few days to several months) to learn. It is not acceptable for learning from accidents in a real environment. Driving control based on accurate physical model requires time-consuming and laborious system control parameters' calibration; Incomplete training/learning will learn potentially unsafe strategies and cannot deal with unlearned or unseen scenarios; Learning from the operator/human requires the use of computationally expensive inverse-reinforcing learning. We proposed an efficient, continuous, and safe first principles-based constrained self-learning to solve the following challenges in autonomous driving: Shorten the learning time; Learn online safely; Avoid complete system control parameters' calibration; Transfer learning to deal with unlearned or unseen scenarios.

Our experiments show that the proposed learning system does not rely on big data collection for initialization; Can reduce the exploration space in reinforcement learning, and significantly improve learning efficiency; Supports continuous online safe learning; The knowledge representation and enhancement are based on first principles' manifestations, and are beneficial to transfer knowledge/skills between different vehicles and between different scenes.

We understand that autonomous driving has a lot of complexities and challenges, this work is a small step toward principled learning, and more complex tasks and situations will be needed to consider. Deep learning based methods might be combined with this first principles based learning to work together (such as the self-playing based learning described in Algorithm 4): The first principles based learning improves the interpretability of the system, and deep learning further generalizes the system capabilities.

References

Arulkumaran, K., and et al. 2017. A brief survey of deep reinforcement learning. *IEEE Signal Processing magazine*.

AVSimulation. Scaner stuio. *online resources, https://www.avsimulation.fr/solutions/*.

Bojarski, M., and et al. 2016. End to end learning for self-driving cars. *online resources, https://arxiv.org/abs/1604.07316*.

Chen, D., and et al. 2019. Learning by cheating. In *Conference on Robot learning (CoRL)*.

Hecker, S., and et al. 2018. End-to-end learning of driving models with surround-view cameras and route planners. In *European Conference on Computer Vision (ECCV)*.

Kahn, G., and et al. 2017. Plato: Policy learning using adaptive trajectory optimization. In *Proceedings of IEEE International Conference on Robotics and Automation*.

Lau, B. 2016. Using keras and deep deterministic policy gradient to play torcs. *online resources, https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html*.

Levine, S., and et al. 2013. Guided policy search. In *Proceedings of International Conference on Machine Learning*.

Levine, S., and et al. 2015. Learning contact-rich manipulation skills with guided policy search. In *Proceedings of IEEE International Conference on Robotics and Automation*.

Lillicrap, T., and et al. 2016. Continuous learning with deep reinforcement learning. In *Proceedings of International Conference on Learning Representations*.

Liu, Y., and et al. 2018. Imitation from observation: Learning to imitate behaviors from raw video via context translation. *Technical Report No. UCB/EECS-2018-37*.

Müller, M., and et al. 2018. Driving policy transfer via modularity and abstraction. In *Conference on Robot learning (CoRL)*.

Ng, A., and Russell, S. 2000. Algorithms for inverse reinforcement learning. In *Proceedings of International Conference on Machine Learning*.

Rao, Q., and Frtunikj, J. 2018. Deep learning for self-driving cars: Chances and challenges. In *2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS)*, 35–38.

Schulman, J., and et al. 2015. Trust region policy optimization. In *Proceedings of International Conference on Machine Learning*.

Silver, D., and et al. 2014. Deterministic policy gradient algorithms. In *Proceedings of International Conference on Machine Learning*.

Song, J., and et al. 2018. Multi-agent generative adversarial imitation learning. In *Proceedings of International Conference on Neural Information Processing Systems*, 7472–7483.

Wymann, B., and et al. 2014. Torcs, the open racing car simulator. *online resources, http://www.torcs.org*.

Zhou, B.; Krähenbühl, P.; and Koltun, V. 2019. Does computer vision matter for action? *Science Robotics*.