# Microscopic Traffic Simulation by Cooperative Multi-agent Deep Reinforcement Learning

Giulio Bacchiani
VisLab - University of Parma
Parma, Italy
giulio.bacchiani@studenti.unipr.it

Daniele Molinari
VisLab
Parma, Italy
dmolina@vislab.it

Marco Patander
VisLab
Parma, Italy
patander@vislab.it

## ABSTRACT

Expert human drivers perform actions relying on traffic laws and their previous experience. While traffic laws are easily embedded into an artificial brain, modeling human complex behaviors which come from past experience is a more challenging task. One of these behaviors is the capability of communicating intentions and negotiating the right of way through driving actions, as when a driver is entering a crowded roundabout and observes other cars movements to guess the best time to merge in. In addition, each driver has its own unique driving style, which is conditioned by both its personal characteristics, such as age and quality of sight, and external factors, such as being late or in a bad mood. For these reasons, the interaction between different drivers is not trivial to simulate in a realistic manner. In this paper, this problem is addressed by developing a microscopic simulator using a *Deep Reinforcement Learning Algorithm* based on a combination of visual frames, representing the perception around the vehicle, and a vector of numerical parameters. In particular, the algorithm called *Asynchronous Advantage Actor-Critic* has been extended to a multi-agent scenario in which every agent needs to learn to interact with other similar agents. Moreover, the model includes a novel architecture such that the driving style of each vehicle is adjustable by tuning some of its input parameters, permitting to simulate drivers with different levels of aggressiveness and desired cruising speeds.

## KEYWORDS

multi-agent systems; microscopic traffic simulation; agent cooperation and negotiation; deep reinforcement learning

## 1 INTRODUCTION

The development of autonomous vehicles is a topic of great interest in recent years, gaining more and more attention both from academy and industry. An interesting challenge is teaching the autonomous car to interact and thus implicitly communicate with human drivers about the execution of particular maneuvers, such as entering a roundabout or an intersection. This is a mandatory request since the introduction of self-driving cars onto public roads is

going to be gradual, hence human- and self-driving vehicles have to cohabit the same spaces. In addition, it would be desirable that this communication would resembles the one which takes place every day on the streets between human beings, so that human drivers does not need to care if the other vehicles are autonomous or not. Achieving this goal would improve the efficiency in traffic scenarios with both artificial and human players, as studied in [9] in case of vehicle-pedestrian negotiation, as well as increase people's trust in autonomous vehicles. We believe that seeing self-driving cars hesitating and interpreting the situation, as human usually do to negotiate, would help in breaking the diffidence of the community and support a seamless integration in regular traffic.

Typical solutions ([28], [4]) for handling those particular maneuvers consist on rule-based methods which use some notion of the *time-to-collision* ([29]), so that they will be executed only if there is enough time in the worst case scenario. These solutions lead to excessively cautious behaviors due to the lack of interpretation of the situation, and suggested the use of machine learning approaches, such as Partially Observable Markov Decision Processes ([17]) or Deep Learning techniques ([12]), in order to infer intentions of other drivers. However, training machine learning algorithms of this kind typically requires simulated environments, and so the behavioral simulation of other drivers plays an important role.

Popular microscopic traffic simulators, such as *Vissim* [6] and *Sumo* [13], use specific hard-coded rules to control vehicles dynamic based on common traffic laws. For example, in uncontrolled intersections where vehicles have to give way to traffic on the right, each vehicle will wait the freeing of the right lane before entering the intersection. At the same time, a vehicle will yield to all cars within a roundabout before entering.

Even if this conduct seems normal, the capability of negotiation between vehicles is missing: an artificial driver will not try to convince other cars of letting it squeeze in, or it will wait forever if another vehicle with the right of way is yielding. In a nutshell, simulated vehicles are not able to break the behavioral rules, a thing that men and women usually do in normal driving scenarios.

Recent advances in machine learning suggest the use of *Deep Neural Networks* ([7]) to achieve complex behaviors, inducing the agent to learn from representations instead of manually writing rules. In particular, the use of *Reinforcement Learning* (RL, [25]) techniques to train such agents appears to be a proficient path ([20]). RL deals with how an agent should act in a given environment in order to maximize the expected cumulative sum of rewards it will receives, with the idea of behaving optimally in the long run instead of chasing the highest immediate reward. This framework is appealing because it does not need explicit supervision at every decision instant, but the agent learns through a scalar reward signal

that reflects the overall result of the actions taken. This reward can also be given once it is known if the whole series of actions led to a good or bad result, like when the car succeed in completing the maneuver or it collides with another vehicle.

A well-known downside of Deep Learning is that it is data-hungry ([10]), namely many examples are needed to robustly train the model. Moreover RL requires both positive and negative experiences, making it unsuitable to be used in real world where negative experiences are generally very expensive to get. These two limitations make the use of a synthetic environment fundamental when using *Deep Reinforcement Learning* (*DRL*).

The aim of this work is to develop a microscopic traffic simulator in which vehicles are endowed with interaction capabilities. This simulator has been thought to be used to train modules able to perform high-level maneuvers with implicit negotiation skills; however it may turn useful for traffic safety and flow analyses as well. Longitudinal accelerations of each vehicle are modeled by a neural network.

Our contributions to reach the goal is twofold: firstly we extend the *Asynchronous Advantage Actor-Critic* (*A3C*, [19]) to operate in a multi-agent setting; secondly, we propose a novel architecture which permits to modify the behavior of every agent, even if they all share parameters of the same network as required by the A3C scenario. Agents are collectively trained in a multi-agent fashion so that cooperative behaviors can emerge, gradually inducing agents to learn to interact with each other.

Each agent receives as input a sequence of images representing the scene portions that can be seen at different times translated as top views, taking a cue from the ATARI environment. Using a visual input makes the system ready to be applicable to different scenarios without the need of having a situation-dependent input. However, this kind of input alone does not give the possibility of differentiate the behavior between different drivers.

Because of that, the introduced architecture permits to incorporate the visual input with a vector of parameters, which are used to tune the behavior of the driver. Indeed, every human has its own peculiar way of driving affected by its temper and external factors resulting in impatience or calmness depending on the situation: let's think for this purpose on how we drive when we are in a hurry or when we are carrying a baby to the kindergarten. Encapsulating drivers with different styles inside the simulator makes the simulation more realistic and would lead to the development of more robust applications. Moreover, the parameter input can be used to give additional details which are not perceivable from images, such as the absolute speed of the agent, enriching the incoming information to the network.

The proposed solution is evaluated on a synthetic single-lane roundabout scenario shown in Figure 1b, but the same approach can be adopted in other scenarios as well. The advantage of abstract representations of this kind is their ease of reproduction with both synthetic and real data, so that high-level maneuver modules trained upon the simulator can be directly used in real world with little effort. A similar approach has been taken concurrently to this work in [1].



**(a) Real world**  **(b) Synthetic representation**

**(c) Navigable**  **(d) Obstacles**  **(e) Path**
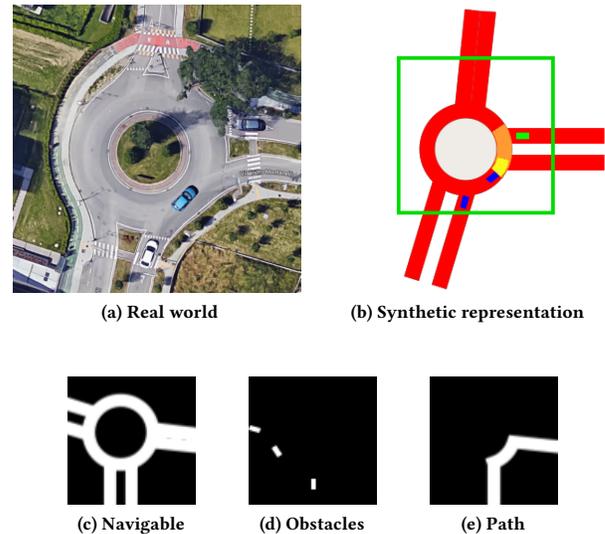
**Figure 1: Synthetic representation of a real world scenario. The top-view of the real scene depicted in (a) is translated in the synthetic representation shown in (b). (c), (d) and (e) are the semantic layers of the region inside the green square used as visual input for the green agent.**

## 2 RELATED WORKS

Road users simulation is essential to the development of maneuver decision making modules for automated vehicles. In [12] a system able to enter in an intersection is trained while other vehicles followed a deterministic model called *Intelligent Driver Model* (IDM, [27]). In [16] a lane change maneuver module is learned using DRL in a scenario where other vehicles follow a simple lane keeping behavior with collision avoidance, while in [14] they are also able to overtake relying on hard-coded rules. In [22] both high level (maneuver decision) and low level policies (acceleration and braking) are learned for addressing an intersection in which other vehicles behave aggressively following some preprogrammed rules.

In all these cases, vehicles populating the environment have a similar conduct and their driving styles are undifferentiated, weakening the realism of the environment.

In [11] the driving styles diversity is increased by assigning randomly generated speed trajectories to the IDM model; in [17] the behavior variation is enforced by adding a finite number of motion intentions which can be adopted by the drivers.

However, none of the aforementioned works explicitly consider agents with complex behaviors such as negotiation, since their motion depends on some specific features, e.g. the distance from the vehicle ahead.

Non-trivial vehicles' motion is obtained in [24] by imitation learning ([2]) using data collected from human drivers; this solution is clearly expensive and directed to a specific situation.

Having a visual input enhance the system capability of modeling complex behaviors and makes it adaptable to a varieties of different problems and road scenarios without the need of shaping the input for each situation. This was initially proved in [20], where the

same algorithm, network architecture and hyperparameters were used to learn how to play many different *ATARI 2600* games. The difficulty posed by the correlation between subsequent states in RL was overcome by storing past experiences and updating the agent with mini-batches of old and new examples, a mechanism called *experience replay*.

The system has been improved in [19] increasing its time and memory efficiency thanks to the parallel execution of multiple agents in place of experience replay. In particular, we extended the *Asynchronous Advantage Actor Critic* (*A3C*) so that it can operate in a multi-agent scenarios where many agents are allowed to interact.

In [8] the algorithm called Trust Region Policy Optimization (TRPO [23]) and A3C are extended to multi-agent systems where all agents share parameters of a single policy and contribute to its update simultaneously. In the same work this approach has been evaluated for other two popular DRL algorithms, DQN and Deep Deterministic Policy Gradient (*DDPG* [15]), which both rely on experience replay; the results were not satisfying, probably due to the incompatibility between experience replay and the non-stationarity of the system coming from the constantly changing policies of agents populating the environment. This suggests that using parallel actor-learners as in A3C has a stabilizing effect which holds in case of a multi-agent setting, since it removes the dependency on the experience replay mechanism. A different approach in case of independent learners with no parameter sharing is taken in [18], where the environment is made stationary by feeding the critic of the DDPG algorithm at training time with actions and observations of all agents.

Parameter sharing is appealing but forces the agents to behave similarly. For this reason we coupled the visual input with some parameters whose values influence the agent policy, inducing different and tunable behaviors. The idea of mixing a high-dimensional visual input and a low-dimensional stream was taken from [5], in which the low-dimensional vector, called *measurements*, is used to define the goal for the agent and to drive the setting toward supervised learning by learning how actions modify future values of the *measurements*. However, this solution fits those problems that have a natural concept of relevant and observable set of measurements (such as video-game playing), but fails on problems in which only a sparse and delayed feedback is available. Hence, we used the low-dimensional input just as an additional input, and followed standard temporal-difference reinforcement learning for training our agents.

## 3 BACKGROUND

### 3.1 Reinforcement Learning

In this work we used Reinforcement Learning methods applied to the *Markov Decision Process* framework, in which an agent takes actions at discrete time steps based on the state of the environment. At every step the agent receives a reward signal together with the new state of the updated environment, which is modified by both its natural progress and the action of the agent itself. The reward signal is generally a simple scalar value and has a central role in the learning process, since it is the only way for the agent to evaluate its actions. This introduces a well-known difficulty of RL: the feedback

is generally the result of many actions, possibly taken several steps in the past.

Describing the process in mathematical terms, the action at time $t$ is represented by $a_t$ and is taken from a discrete or continuous range of actions referred to as $A$; the state at time $t$ is represented as $s_t$ while the reward as $r_t$. The goal of the agent is to learn a policy $\pi(a|s)$, namely the probability of taking action $a$ given the state $s$, in order to maximize the expectation of a function of future rewards called *return*. The return is generally defined as $R_t = \sum_t^T r_t + \gamma r_{t+1} + \dots + \gamma^{T-t} r_{T-t}$, where T is the terminal time step and $\gamma$ is a discount factor used to modulate the importance of future rewards.

Useful for the estimation of a policy are the *value functions*. The *state-value function* (1) of a policy $\pi$ gives the expected return when, starting from state $s$, the policy $\pi$ is followed; the *action-value function* (2), similarly, gives the expected return if we follow policy $\pi$ after taking action $a$ in state $s$.

$$V_\pi(s_t) = \mathbb{E}(R_t|s_t) \qquad (1)$$

$$Q_\pi(s_t, a_t) = \mathbb{E}(R_t|s_t, a_t) \qquad (2)$$

If the agent was able to find the *optimal action-value function* $Q^*(s, a)$, namely the action-value function of the best policy, the problem would be solved since it could pick the best action from a generic state $s_t$ simply by choosing $a^* = \max_{a \in A} Q^*(s, a)$.

Trying to estimate this function is the goal of those algorithms which fall under the category of *action-value methods* such as Q-*learning* ([30]). If we approximate $Q^*(s, a)$ with a function $q(s, a; \theta)$ modeled by the vector of parameters $\theta$, in *1-step Q-learning* the parameters are updated as:

$$\theta_{t+1} = \theta_t + \alpha \cdot K \frac{\partial q(s_t, a_t; \theta_t)}{\partial \theta_t} \qquad (3)$$

where $\alpha$ is a parameter dependent on the optimization algorithm while $K$ tells if the approximated action-value function should be increased or decreased around the pair $\{s_t, a_t\}$ and is defined as:

$$K = r_t + max_{a^*} q(s_{t+1}, a^*; \theta_t) - q(s_t, a_t; \theta_t) \qquad (4)$$

It is worth noting that the target value $r_t + max_{a^*} q(s_{t+1}, a^*; \theta_t)$ is an estimate itself because it uses the estimated action-value function of the next state as better approximation of the real optimal action-value. This operation of learning "a guess from a guess" is called *bootstrapping*.

A different approach is taken in *policy-based methods*, which directly estimate the policy instead of a value function. In this category falls the algorithm called *Reinforce* ([31]) in which parameters are updated such that probabilities of actions that yielded a better cumulative reward at the end of an episode are increased with respect to the lower-return actions. Approximating the optimal policy $\pi^*(a|s)$ with a function $\pi(a|s; \theta)$, the updates are:

$$\theta_{t+1} = \theta_t + \alpha \cdot R_t \frac{\partial \log \pi(a|s; \theta)}{\partial \theta} \qquad (5)$$

In this algorithm the updates can be computed only at the end of an episode, since the true cumulative reward is needed.

Between action-value and policy-based methods there is an hybrid family of algorithms called *actor-critic methods*.

In this setting the policy is directly optimized as in policy-based solutions; however, the value function is also estimated, giving two

benefits: firstly the value of a state is used to reduce the variance of updates, as explained in [25]; furthermore the estimated state-value function permits bootstrapping, thus avoiding the need to wait for the end of an episode before performing a parameters update. The probability of actions can now be modulated with the following *Advantage* $A_b$ instead of the full reward $R_t$:

$$A_b = r_t + \gamma r_{t+1} + ... + \gamma^{b-1} r_{t+b-1} + v(s_{t+b}; \theta^v) - v(s_t; \theta^v) \quad (6)$$

where $b$ is the number of real rewards used before bootstrapping, called *bootstrapping interval*.

Calling the approximated optimal policy $\pi(a|s; \theta^\pi)$ and the state-value function $v(s; \theta^v)$, the updates of this two functions can now be defined as:

$$\theta_{t+1}^\pi = \theta_t^\pi + \alpha_\pi \cdot A_b \frac{\partial \log \pi(a|s; \theta^\pi)}{\partial \theta^\pi} \quad (7)$$

$$\theta_{t+1}^v = \theta_t^v + \alpha_v \cdot A_b \frac{\partial v(s_t; \theta^v)}{\partial \theta^v} \quad (8)$$

If the bootstrapping interval $b$ equals 1, only the reward following an action is used to directly evaluate that action. This setting is ideal in scenarios where the complete consequences of an action are known immediately (e.g. face correctly recognized by a face detection agent); however, it is not optimal in situation where the reward is delayed (e.g. agent regulating the water level of a dam). Increasing $b$, a longer series of rewards will be accumulated before estimating the value function; in this way delayed rewards are propagated faster but their merit is divided among all the actions performed in the time window.

In the *n-step actor critic* algorithm used in A3C a mixed approach is adopted. In this solution both long and short-term bootstrapping take place: every *n* time-steps (or if a terminal state is reached) a sequence of updates like those in equations (7) and (8) are performed, one for each time-step. In each update the longest possible series of rewards is used for estimating the value of the state, ranging from a 1-step to n-step updates.

## 3.2 Asynchronous Advantage Actor Critic - A3C

A problem of using Deep Neural Networks as function approximators in RL is the correlation between updates that comes from the sequentiality of data in the process.

This correlation was initially broken by using the so called experience replay ([20]): *tuples* $\{s_t, a_t, s_{t+1}, r_t\}$ were stored instead of being used immediately, while the parameters updates were computed using randomly sampled tuples from the replay buffer. In this setting, updates are not based on sequential data anymore, but the replay buffer makes the process expensive in terms of memory occupation and it forces the use of old data.

An improved solution, adopted in [19], consists in running several actor-critic agents in parallel, each one with its own copy of the parameters of a global neural network. Every agent acts in a different instance of the environment and sends its updates asynchronously from the other agents. Since each agent is seeing a different environment configuration, their updates are not correlated and contribute in augmenting the stability of the process. A visual representation of this system is shown in Figure 2a.

# 4 MICROSCOPIC TRAFFIC SIMULATION

## 4.1 Multi-agent A3C

In the normal A3C setting, multiple agents act in parallel in independent instances of the same environment. However, it is not possible for them to learn the interaction with each other, since each agent is sensing a state which is independent from that of the others.

In order to make this interaction possible we let several agents share the same playground, so that they will be able to sense each other. For instance, let's consider agents $a$ and $b$ acting on the same environment: $a$, by taking action $a_t^a$ at time step $t$ based on policy $\pi(a_t^a|s_t^a)$, could affect state $s_{t+1}^b$ of the agent $b$. If this happens, $b$ will react accordingly taking action $\pi(a_{t+1}^b|s_{t+1}^b)$, potentially affecting in turn $s_{t+2}^a$. Therefore, learning the optimal policy $\pi$ for an agent means taking into account not only how the environment is affected by its actions, but also what behavior they induce to other agents.

In practice, this implies that agents have to wait that the others took their actions before receiving the subsequent state. This affects negatively the performance of the algorithm, since, due to the asynchronous nature of the process, some of the agents could be computing their updates, that is executing a time consuming backpropagation. Moreover, this solution jeopardizes the stability of the system because parameter updates of agents acting in the same environment are not independent anymore. For this reason, in multi-agent A3C we run several environment instances as in traditional A3C, each one populated with several active agents.

Each agent accumulates updates every *n* frames, and sends them to a global copy of the network only at the end of the episode in order to reduce the synchronization burden, updating at the same time its local copy with the updates coming from the other agents. Since some of those agents are accumulating experience in different environment instances, the correlation between updates is diminished, making the process more stable. Algorithm 1 shows the process of multi-agent A3C, which visual representation is given in Figure 2b.

In our experiment we set $n = 20$ and $\gamma = 0.99$; we used *RMSProp* optimizer with a decay factor of $\alpha = 0.99$ and initial learning rate of $7e^{-4}$.

Depending on how the reward function is shaped, the learning process may be cooperative or competitive ([26]). The former is the way we followed to simulate behavior of road vehicles: a positive reward is given to the artificial driver if it reaches its goal while a negative reward is given to those drivers involved in a crash. Agents are stimulated to reach their goals avoiding other agents, leading to an implicit negotiation when the paths of different vehicles intersect.

We think that this multi-agent learning setting is captivating for many applications that require a simulation environment with intelligent agents, because it learns the joint interactive behavior eliminating the need for hand-designed behavioral rules.

## 4.2 Input architecture

Having a visual input organized as images is appealing since it frees from the need of defining a case-specific design for each different

---

**Algorithm 1** multi-agent *n-step* Advantage Actor Critic

---

1: initialize parameters $\theta_\pi^g$ of $\pi(a|s;\theta_\pi)$
2: initialize parameters $\theta_v^g$ of $V(a|s;\theta_v)$
3: get $n_{env}$ as the number of environment instances
4: get $n_{ag}$ as the number of agents for each environment
5: **run** $n_{env}$ environment instances concurrently
6: **for each** *environment instance* **do**
7:    **run** $n_{ag}$ threads concurrently
8: **end for**
9: **for each** *agent a* **do**          *# executed concurrently*
10:    $t \leftarrow 1$
11:    **while** *training is active* **do**
12:       copy parameters $\theta_\pi^a = \theta_\pi^g$
13:       copy parameters $\theta_v^a = \theta_v^g$
14:       $\Delta_\pi^a = 0, \ \Delta_v^a = 0$
15:       **while** $s_t$ *is not terminal* **do**
16:          get $s_t$
17:          $t_{last\_up} = t$
18:          **while** $t - t_{last\_up} < n$ **and** $s_t$ *is not terminal* **do**
19:             execute action $a_t$ from $\pi(a_t|s_t;\theta_\pi^a)$
20:             wait other agents to finish their actions
21:             get reward $r_t$, state $s_{t+1}$
22:             $t \leftarrow t + 1$
23:          **end while**
24:          $R = \begin{cases} 0 & \text{if } s_t \text{ is terminal} \\ V(s_t, \theta_v^a) & \text{otherwise} \end{cases}$
25:          **for each** $i = t - 1, .., t - t_{last\_up}$ **do**
26:             $R = r_i + \gamma r$
27:             $\Delta_\pi^a = \Delta_\pi^a + \nabla_{\theta_\pi^a} \log \pi(a_i|s_i;\theta_\pi^a)(R - V(a|s_i;\theta_v^a))$
28:             $\Delta_v^a = \Delta_v^a + \nabla_{\theta_v^a}(R - V(a|s_i;\theta_v^a))^2$
29:          **end for**
30:       **end while**
31:       **update** $\theta_\pi^g$ using $\Delta_\pi^a$
32:       **update** $\theta_v^g$ using $\Delta_v^a$
33:    **end while**
34: **end for**

---

scenario. Moreover, it frees from the need of considering a fixed amount of vehicles (as in [11]).

However, this kind of input has a drawback: the convolutional pipeline makes it not suitable for communicating numerical information, such as the desired behavior for the agent or its precise speed. Transferring this type of information is possible using fully-connected layers, but it becomes inefficient to directly process visual images by this mean. For this reason, we shaped our system in order to have a mixed input, made of *visual* and *numerical* information, which is processed by two different pipelines as elucidated in Section 4.3.

*4.2.1 Visual input.* The agent may not be able to sense the whole environment but just a portion of it, which we call its *view*. The visual part of the input consists on a historical sequence of the $v$ most recent views seen from the agent. We set $v = 4$ as in [20] making the estimation of relative speed and acceleration feasible. Each view is divided in semantic layers separating information with different origin. As explained in Section 5.1 we included semantic



(a) Single-agent setting



(b) Multi-agent setting
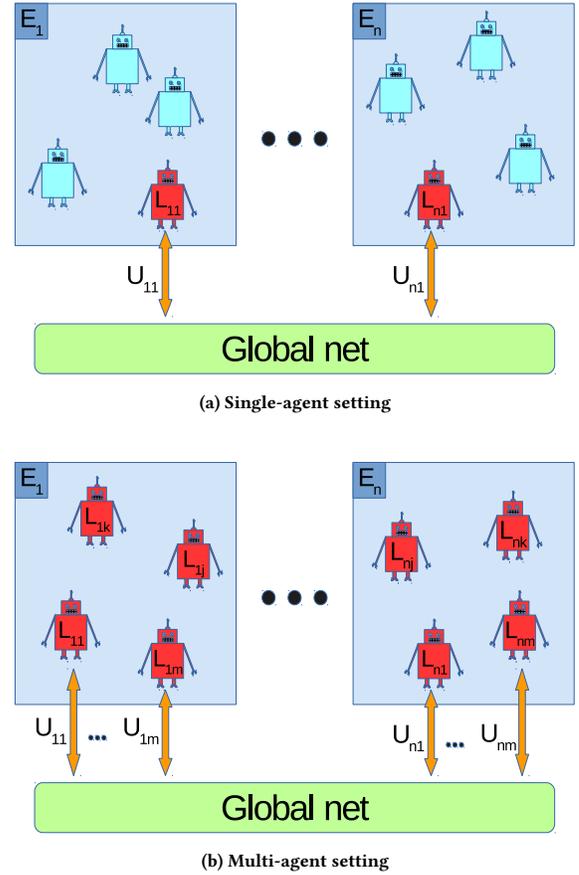
**Figure 2: Comparison between single and multi-agent A3C settings. Red robots are learning agents acting in environment $E_i$, while blue robots are passive agents. Each active agent owns a copy $L_j$ of the global network, and it contributes to its evolution by updates $U_j$. In (a) learning agents play in an environment populated only by non-active agents, while in (b) active agents can sense each other, allowing them to learn how to interact.**

layers related to the navigable space, the path the agent will follow and obstacles around the vehicle; however, several other layers could be added, such as the occluded area or the road markings, in order to enrich the input information.

*4.2.2 Numerical input.* The numerical component permits the augmentation of the input with information which are not directly perceivable from the sequence of views, such as the absolute speed of the agent or the exact distance which is still to be traveled to reach the goal. This allows a better understanding of the scene, i.e. a better estimation of the states value. More importantly, the numerical input can be used to shape agents attitude by teaching it a direct correlation between some of its tunable input and the to-be-obtained reward. Practically, they can be used to partially perturb the state of the agent at will. In this work we took advantage of
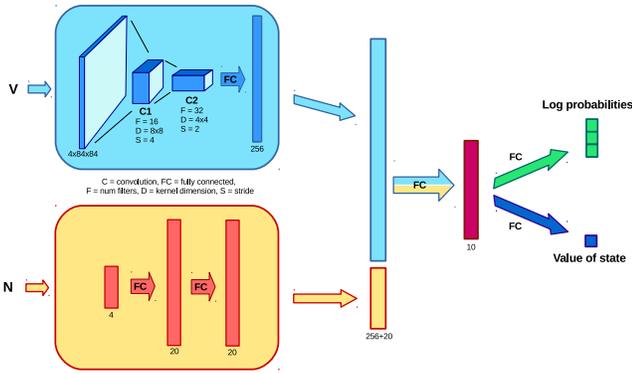
**Figure 3: Architecture of the network. Visual input V and numerical input N initially follow two different pipelines, before being merged in a single vector, which is then used to compute probabilities of actions and to estimate the value of the state.**

numerical input to the network for tuning the agent aggressiveness and to suggest a desired cruising speed, as explained in Sections 5.4 and 5.5 respectively.

### 4.3 Network architecture

Because of the different nature of the two input streams, they are initially processed by two different pipelines, aimed to produce two feature vectors.

The image processing pipeline is constituted by two convolutional and one fully connected layers, whereas the numerical input vector is processed by two fully connected layers. Each pipeline produces a feature vector as output. The two output vectors are joined together before being handled by the last fully-connected hidden layer. After every transformation a *ReLU* nonlinearity is applied to the output.

The idea of this hybrid architecture has been inspired from [5], in which an input made of a single image and two numerical vectors is processed by a network built in a similar fashion.

The output of the last hidden layer is finally used for the actions log-probabilities computation and the state-value estimation. The network architecture is represented in Figure 3.

### 5 EXPERIMENT

The model explained in Section 4 has been used to simulate the traffic of cars on a roundabout with three entries (Figure 1b), which is a representation of an existing roundabout (Figure 1a) made with the help of the *Cairo* graphics library ([21]).

Each vehicle populating the roundabout has its own path which simply depends on a randomly assigned exit; the goal of the agent is to reach the end of its exit lane. Furthermore, to each agent is assigned a desired cruising speed consisting to the maximum speed the agent is aiming for, which it should be reached and maintained when caution is not needed; this speed will be called from now on its target speed. A new vehicle with a random target speed is spawned from an entry lane dependently on the positions of the other cars, inducing different traffic conditions.

During the experiments the roundabout was populated by a maximum of six vehicles such that no traffic jams were created, thus making the effects of the agent's behavior tuning appreciable. However, this framework can be used to simulate vehicles' behavior even in high traffic conditions.

### 5.1 State space

As explained in Section 4.2 the input is made of two different streams, a visual and a numerical one. The visual part is a representation of the surrounding of the vehicle within a limited range of $50x50$ square meters converted to a sequence of images with $84x84$ pixels. This input is made of several semantic layers, which in this experiment are:

(1) **Navigable space**: the part of the space in which vehicles can drive, generally given from the maps and shown in Figure 1c;
(2) **Obstacles**: made of all vehicles and obstacles seen from the agent, including the agent himself; this information can be obtained from the perception module of the self-driving vehicle and it is shown in Figure 1d;
(3) **Path**: that shows to the agent its assigned path, and depends on the choices of the high-level planner; this layer is shown in Figure 1e.

The numerical part, on the other hand, consists of several scalar values which contain both additional information about the scene and clues to predict future rewards, which are used at test time to modulate the agent behavior. In our instance the numerical input is restricted to the following parameters:

(1) **Agent speed**.
(2) **Target speed**.
(3) **Elapsed time ratio**: ratio between the elapsed time from the beginning of the episode and the time limit to reach the goal.
(4) **Distance to the goal**: distance to be traveled before reaching the goal.

The network architecture, together with the hyperparameters used for the experiment, are shown in Figure 3.

### 5.2 Action space

The chosen action space is discrete and boils down to three actions: accelerate, brake or maintain the same speed. We set comfortable acceleration values, which are $1\frac{m}{s^2}$ in case of positive acceleration and $-2\frac{m}{s^2}$ in case of braking. The acceleration command will have its effect only if the speed is under a maximum accepted value.

### 5.3 Reward shaping

The reward obtained from an agent is made of three main terms:

$$r_t = r_{terminal} + r_{danger} + r_{speed} \qquad (9)$$

$r_{terminal}$ depends on how the episode ends, and so differs from zero only if $t$ is a terminal state; it equals:

- +1: if the agent reaches its goal;
- −1: if the agent crashes against another agent;
- −1: if the available time expires.
- 0: if $t$ is a non-terminal state.

$r_{danger}$ is a soft penalization given in situations which are to be avoided, either because the agent made a dangerous maneuver or because it broke some traffic laws. Calling $d_v$ the distance traveled from the vehicle $v$ in one second, $r_{danger}$ will be:

- $-k_y$: if the agent fails to yield when entering the roundabout to an already inserted vehicle $v$. This happens when the agent crosses the region in front of $v$ whose length is taken equal to $3 \cdot d_v$, as shown from the orange region in the example of Figure 1b.
- $-k_s$: if the agent $a$ violates the safety distance with the vehicle in front, unless the latter is entering the roundabout and it is cutting in front of the agent. The safety distance in this case is taken equal to $d_a$, and the associated region is depicted in yellow in Figure 1b.
- 0: if none of the above occur.

$r_{speed}$ is a positive reward which is maximized when the actual speed of the agent ($s_a$) coincides with the target speed input parameter ($s_t$):

$$r_{speed} = \begin{cases} \frac{s_a}{s_t} \cdot k_p & \textbf{if } s_a \leq s_t \\ k_p - \frac{s_a - s_t}{s_t} \cdot k_n & \textbf{if } s_a > s_t \end{cases} \quad (10)$$

In our experiment we set $k_y = k_s = 0.05$, $k_p = 0.001$, $k_n = 0.03$.

## 5.4 Aggressiveness tuning

In the training phase, agents learn to increase their speed to reach the goal within a time limit. This is possible because both time left and remaining distance to be traveled are provided as input. It was not possible to achieve this same behavior without using the distance-to-the-goal input, highlighting the agent inability to realize from images that increasing its speed the goal will be reached faster. Nevertheless, things work well when this information is enforced with the remaining distance, confirming the power of the visual-numerical coupling.

This experience has been exploited at test time to vary the agent aggressiveness by tuning its time and distance left at will, since it is not needed anymore to limit the episode length. Doing this each agent present in the simulator will have its own unique attitude, resulting in a more heterogeneous scenario.

We explored scenarios in which the elapsed time ratio was replaced by an arbitrary real value in the $[0, 1]$ interval, keeping the distance to an imaginary goal fixed. Values close to 1 will induce the agent to drive faster, in order to avoid the predicted negative return for running out of time. For the same reason, values close to 0 will tell the driver that it still has much time, and it is not a problem to yield to other vehicles. This way, the elapsed time ratio acts as an "aggressiveness level" for the agent.

An experiment has been set up in order to test the validity of this approach. We populated the roundabout with six vehicles having random aggressiveness levels, and we tested how an agent with a fixed known aggressiveness behaves by letting it play several episodes. We registered two different parameters as feedback of each driver style: the ratio of positive ending episodes, in which the agent reaches the goal, and its average speed. Results are shown in the graph of Figure 4a. It is clearly visible that, by increasing the aggressiveness level, the driver behavior will shift towards a more risky configuration, since both average speed and probability of

accidents grow. Moreover, it is interesting to see that values of aggressiveness outside the training interval $[0, 1]$ produce consistent consequences to the agent conduct, intensifying its behavior even further.

## 5.5 Target speed tuning

The $r_{speed}$ reward term, presented in Section 5.3, increases until the speed of the agent gets closer to its target speed, and it starts to diminish for agent speeds above the desired one. At training time, agents were assigned a random target speed for every episode, taken from a uniform distribution between $[5, 8]\frac{m}{s}$, inducing them to learn to not surpass that tunable value and so simulating drivers with different desired speeds. Agents can still decide to surpass their target speed if this will bring advantages, such as stopping the insertion of another vehicle by a particular aggressive driver.

To evaluate how the dedicated input affects the behavior of the agent we let several agents, having different target speed values but equal aggressiveness levels of 0.5, to play several runs in scenarios similar to the one used to test the aggressiveness tuning in Section 5.4.

The graph with the results, given in Figure 4b, shows that an increment on the agent target speed leads to an higher average speed, without affecting negatively the positive ending ratio. Even in this case, values of the target speed outside the training range $[5, 8]$ induce consistent agent behaviors.

## 5.6 Environment settings comparisons

We compared agents trained with 1 and 4 environment instances in a simplified case in which the target speed given to the agent cannot be exceeded (once reached the target speed the accelerate command has no effect), in order to speed up learning for comparison purposes. We tested both with and without *action repeat* of 4 ([20]), that is repeating the last action for the following 4 frames (repeated actions are not used for computing the updates). It has been proved that in some cases action repeat improves learning ([3]) by increasing the capability to learn associations between temporally distant state-action pairs, giving to actions more time to affect the state.

The learning curves, shown in Figure 5, tell that, when using the action repeat technique, training the system with multiple instances stabilizes learning and leads to a better overall performance, even if in both cases the model converges. This confirms that reducing the correlation between updates brings a positive effect. It is interesting to note that in the multi-instance scenario agents start to improve their ability to reach the goal at a later stage but with a faster pace. Our explanation for this behavior is that, when running several environment instances, a higher rate of asynchronous updates coming from pseudo-random policies makes learning more problematic. Indeed, during the length of an episode, the local policy of the agent for which the updates are computed remains fixed, while the global policy receives asynchronous updates from several other agents whose behavior is still infant. However, when the policies of agents start to gain sense, their updates will be directed strongly toward a common goal due to a reduced search space.

Nonetheless, things change when action are not repeated. Indeed, while the model still converges when using a single environment

**(a) aggressiveness test**



**(b) speed test**
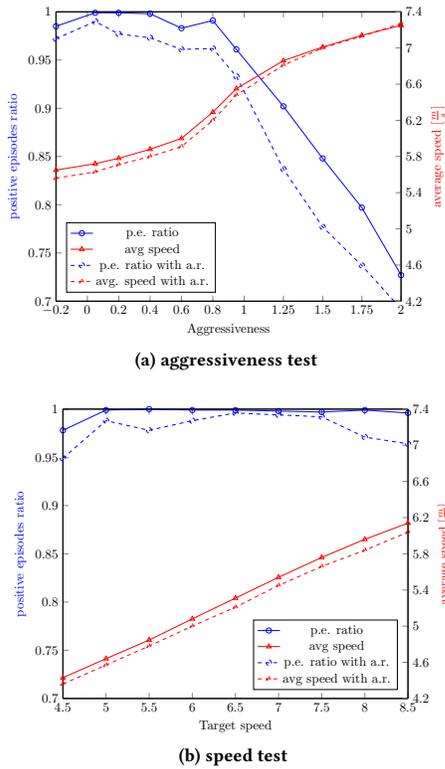
**Figure 4: Effect of aggressiveness (a) and target speed (b) levels variation on the agent behavior. Blue plots show the agent positive-ending episodes ratio while the red plots its average speed. Dashed lines refer to an agent whose action are repeated for 4 times while solid lines to an agent free to choose actions at every time step. Each data point is obtained averaging over 5000 episodes.**

instance, agents are not able to learn successfully when adopting the multi-instance setting, making the stabilizing effect of action repetition necessary.

However, the use of repeated actions brings a drawback, that is to diminish the rate at which agents interact with the environment. Indeed our empirical results, given in the graphs of Figure 4, show that the positive episode ratio at test time increases if actions are not repeated, even if the model was trained with action repetition. In this way it is possible to take advantage of the stabilizing effect of action repetition, without loosing the chance of a finer action selection resolution.

## 6 CONCLUSION

We presented an architecture able to model the longitudinal behavior of vehicles on a synthetic road scenario by using a multi-agent version of the A3C algorithm. This solution allows simulating road traffic without the need of specifying hand crafted behaviors for vehicles or collecting real data.

Agents populating the simulator sense the surrounding by a visual input made of a sequence of images representing different



**(a) 1 instance, yes action repeat**



**(b) 4 instances, yes action repeat**



**(c) 1 instance, no action repeat**



**(d) 4 instances, no action repeat**
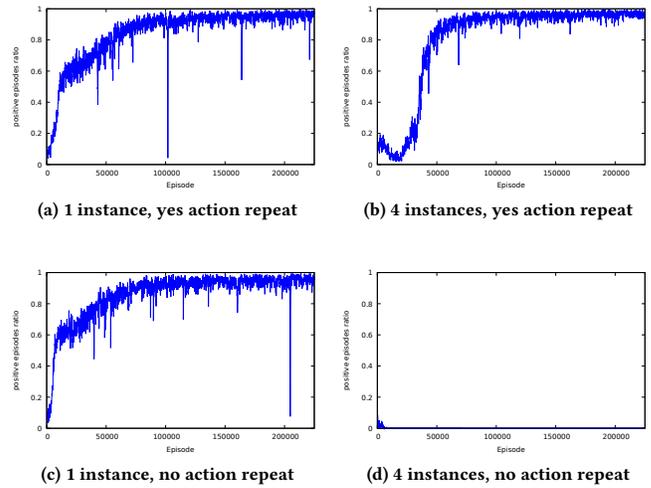
**Figure 5: Learning comparison between systems trained with single or multiple environment instances, with or without action repeat.**

semantic information across the last time steps. The proposed architecture gives also the possibility to simulate the uniqueness of each real human driver by coupling a series of scalar parameters to the visual one. We exploited this feature so that aggressiveness and desired cruising speed of each driver can be regulated.

Even if our framework is able to deal with generic junction shapes, we focused our tests on a specific scenario consisting of a single-lane roundabout with three entries: it would be interesting to see how well the system is able to generalize to different junction topologies for application in which these information are not known a priori. Moreover, it would be intriguing to extend the architecture in order to include lateral behaviors of vehicles in the simulation.

It is worth to mention, as can be seen in the previous graphs, that the proposed simulator is not collision free, but it reaches success ratios close to 1 when a safe driving is set. Even though a collision-free simulator could be desirable for some applications, the aim of this work was to simulate real driving scenarios in which errors and accidents happen: modeling them in the simulator may lead to the development of more robust high-level maneuver modules.

The time needed to train the model is not negligible and can reach several days in a desktop computer. The main issue is that before proceeding to the subsequent time step, all the agents acting in the same environment instance need to have taken their actions: since some of them might be computing gradients, the overall performance decreases. Future works can be directed toward the design of a more efficient network architecture, as well as the development of a learning setting in which the gradient computations of all the agents in the same instance are executed simultaneously.

## 7 ACKNOWLEDGMENTS

# REFERENCES

[1] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. 2018. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. (2018). arXiv:1812.03079

[2] Mariusz Bojarski et al. 2016. End to End Learning for Self-Driving Cars. (2016). arXiv:1604.07316

[3] Alex Braylan et al. 2015. Frame Skip Is a Powerful Parameter for Learning to Play Atari. In *AAAI-15 Workshop on Learning for General Competency in Video Games*.

[4] Akansel Cosgun et al. 2017. Towards Full Automated Drive in Urban Environments: A Demonstration in GoMentum Station, California. abs/1705.01187 (2017). arXiv:1705.01187

[5] Alexey Dosovitskiy and Vladlen Koltun. 2017. Learning to Act by Predicting the Future. In *International Conference on Learning Representations (ICLR)*.

[6] Vortisch P. Fellendorf M. [n. d.]. Microscopic Traffic Flow Simulator VISSIM. In *Fundamentals of Traffic Simulation*, Jaume Barcelò (Ed.).

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[8] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In *Autonomous Agents and Multiagent Systems*. 66–83.

[9] S. Gupta, M. Vasardani, and S. Winter. 2018. Negotiation Between Vehicles and Pedestrians for the Right of Way at Intersections. *IEEE Transactions on Intelligent Transportation Systems* (2018), 1–12.

[10] Alon Halevy, Peter Norvig, and Fernando Pereira. 2009. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems* 24 (2009), 8 – 12.

[11] Carl-Johan Hoel, Krister Wolff, and Leo Laine. 2018. Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning. (2018). arXiv:1803.10056

[12] David Isele, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. 2017. Navigating Occluded Intersections with Autonomous Vehicles using Deep Reinforcement Learning. (2017). arXiv:1705.01196

[13] Daniel Krajzewicz, Georg Hertkorn, Christian Feld, and Peter Wagner. 2002. SUMO (Simulation of Urban MObility); An open-source traffic simulation. In *4th Middle East Symposium on Simulation and Modelling (MESM2002)*. 183–187.

[14] X. Li, X. Xu, and L. Zuo. 2015. Reinforcement learning based overtaking decision-making for highway autonomous driving. In *International Conference on Intelligent Control and Information Processing (ICICIP)*. 336–342.

[15] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. (2015). arXiv:1509.02971

[16] Jingchu Liu, Pengfei Hou, Lisen Mu, Yinan Yu, and Chang Huang. 2018. Elements of Effective Deep Reinforcement Learning towards Tactical Driving Decision Making. (2018). arXiv:1802.00332

[17] W. Liu, S. Kim, S. Pendleton, and M. H. Ang. 2015. Situation-aware decision making for autonomous driving on urban road using online POMDP. In *IEEE Intelligent Vehicles Symposium (IV)*. 1126–1133.

[18] Ryan Lowe et al. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. (2017). arXiv:1706.02275

[19] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning (ICML)*. 1928–1937.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533.

[21] Keith Packard and Carl Worth. 2003–2018. Cairo graphics library. https://www.cairographics.org/. (2003–2018).

[22] Chris Paxton, Vasumathi Raman, Gregory D. Hager, and Marin Kobilarov. 2017. Combining neural networks and tree search for task and motion planning in challenging environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vol. 2017-September. 6059–6066.

[23] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2015. Trust Region Policy Optimization. In *International Conference on Machine Learning (ICML)*.

[24] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. 2016. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. (2016). arXiv:1610.03295

[25] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

[26] Ardi Tampuu et al. 2015. Multiagent Cooperation and Competition with Deep Reinforcement Learning. (2015). arXiv:1511.08779

[27] M Treiber, A Hennecke, and D Helbing. 2000. Congested Traffic States in Empirical Observations and Microscopic Simulations. *Phys. Rev. E* 62 (2000), 1805–1824.

[28] Chris Urmson et al. 2008. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *J. Field Robot.* 25, 8 (Aug. 2008), 425–466.

[29] Richard van der Horst and Jeroen Hogema. 1994. Time-To-Collision and Collision Avoidance Systems. In *6th ICTCT workshop*.

[30] Christopher John Cornish Hellaby Watkins. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation. King's College, Cambridge, UK.

[31] Ronald J. Williams. 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* 8, 3-4 (May 1992), 229–256.