Cooperative Adaptive Cruise Control: a Reinforcement Learning Approach*

Julien Laumonier, Charles Desjardins and Brahim Chaib-draa
DAMAS Laboratory,
Department of Computer Science
and Software Engineering,
Laval University, Canada
{jlaumoni;desjardins;chaib}@damas.ift.ulaval.ca

ABSTRACT

As a part of Intelligent Transport Systems (ITS), Cooperative Adaptive Cruise Control (CACC) systems have been introduced for finding solutions to the modern problems of automotive transportation such as traffic efficiency, passenger comfort and security. To achieve cooperation, actors on the road must use internal sensors and communication. Designing such a controller is not an easy task when the problem is considered in its entirety, since the interactions taking place in the environment (from vehicle physics and dynamics to multi-vehicle interaction) are extremely complex and hard to model formally. That is why many ITS approaches consider many levels of functionnalities. In this article, we will show our work toward the design of a multiple-level architecture using reinforcement learning techniques. We explain our work on the design of a longitudinal ACC controller, which is the first step toward a fully functionnal CACC low-level controller. We describe the design of our high-level controller used for vehicles coordination. Preliminary results show that, in some situations, the vehicle-following controller is stable. We also show that the coordination controller allows to have an efficient lane allocation for vehicles. At last, we present some future improvements that will integrate both approaches in a general architecture centered on the design of a CACC system using reinforcement learning.

1. INTRODUCTION

More and more sensors are used today to gather information related to a number of components inside vehicles. Even though this information is used mainly for monitoring, some applications go even further and use it to gain knowledge on the environment of the vehicle. With information on the state of the environment, it becomes possible to make driving decisions that can help solve today's transportation problems, working toward an increase in the efficiency of traffic but also in the comfort and security of passengers. Intelligent Transport Systems (ITS) [19] are interested in developing technology to settle those issues.

One of ITS's main focus is on Adaptive Cruise Control (ACC), which is a good example of where the technology is headed. ACCs use sensors to detect preceding vehicles and adapt the cruising velocity of a car according to what might lie ahead. If a preceding vehicle is present, the car automatically slows down to avoid collision and keep a safe distance behind. Already, ACCs are available in high-end vehicles [2]. Still, today's ACC systems are limited as they do not provide a mean to share information between surrounding vehicles. As states Tsugawa [18], the use of inter-vehicle communication could help fulfill the goals of ITS by providing a system for vehicles to share with others sensor data representing their environment. With a communication system, ACCs become Cooperative Adaptive Cruise Control systems (CACCs), which have communication and cooperation between vehicles as primary concerns. Clearly, the ultimate goal is to design controllers in order to enhance today's traffic efficiency and passenger comfort and security.

To design ACC and CACC controllers, reinforcement learning is certainly a good solution. Indeed, some reinforcement learning algorithms allow us to learn an optimal policy for acting in an environment without knowing its exact inner workings. However, in the most general case, it is not possible to design a controller taking into account the entire problem of cooperative cruise control because of the complexity of the task. In this case, we consider a two levels approach. The first one is focusing on low-level control where a vehicle follows another vehicle at a secure distance, by acting directly with the throttle and observing the concrete results on the inter-vehicle gap. This level is car-centered and can be modeled by a Markov Decision Process (MDP) that can be solved using algorithms that learn on-line, such as Temporal-Differences (TD) algorithms. On the other hand, the second level is of higher level and focuses on vehicle coordination. In that context, a learned policy could choose the best driving lane for every vehicle located on a highway system. Because of the multiagent point of view of this subproblem, it could benefit from the use of Stochastic Games to be modeled as an extension to MDP.

In this article, we will present some results in the design of controllers working together in solving both the low-level and the high-level previously introduced. In the next section, we present the state of the art of MDP and reinforcement learning for mono and multi-agent settings. In section 3, we present the general architecture that links the low and high-level controllers. In section 4, we present the design of the low-level ACC controller. In section 5, the high-level coordination controller is introduced. Then, section 6 presents

*This research is funded by the AUTO21 Network of Centres of Excellence, an automotive research and development program focusing on issues relating to the automobile in the 21st century. AUTO21 is a member of the Networks of Centres of Excellence of Canada program. Web site: www.auto21.ca

experiments for both layers and section 7 presents a discussion on the improvements that can be done on both sub-problems. At last, section 8 contains related work and we finish with a conclusion.

2. REINFORCEMENT LEARNING AND GAME THEORY

Reinforcement learning allows an agent to learn by interacting with its environment. For a mono agent system, the basic formal model for reinforcement learning is the Markov Decision Process (MDP). A MDP is a tuple $< S, A, \mathcal{P}, \mathcal{R} >$ where

- $S = \{s_0, \cdots, s_M\}$ is the finite set of states where $|S| = M$,
- $A = \{a_0, \cdots, a_p\}$ is the finite set of actions,
- $\mathcal{P} : S \times A \times S \rightarrow \Delta(S)$ is the transition function from current state, agent action and new state to probability distribution over the states,
- $\mathcal{R} : S \times A \rightarrow \mathbb{R}$ is the immediate reward function for the agent.

Using this model, the Q-Learning algorithm calculates the optimal values of the expected reward for the agent in a state s if the action a is executed. To do this, the following update function for state-action values is used:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a \in A} Q(s',a)]$$

where r is the immediate reward, s' is the next state and α is the learning rate. An *episode* is defined by a sub-sequence of interaction between the agent and its environment.

On the other hand, Game Theory studies formally the interaction of multiple rational agents. In a one-stage game, each agent i has to choose an action to maximize its own utility $U^i(a^i, a^{-i})$ which depends on the others' actions a^{-i}. An action can be *mixed* if the agent chooses it with a given probability and can be *pure* if it is chosen with probability 1. In game theory, the solution concept is the notion of equilibrium. For an agent, the equilibria are mainly based on the best response to other's actions. Formally, an action a^i_{br} is a best response to actions a^{-i} of the others agents if

$$U^i(a^i_{br}, a^{-i}) \geq U^i(a'^i, a^{-i}),\ \forall a'^i.$$

The set of best responses to a^{-i} is noted $BR^i(a^{-i})$.

The Nash equilibrium is the best response for all agents. Formally, a joint action a_N, which regroups the actions for all agents, is a Nash equilibrium if

$$\forall i,\ a^i_N \in BR^i(a^{-i})$$

where a^i_N is the action of the i^{th} agent in the Nash equilibrium and a^{-i}_N is the actions of other agents at Nash equilibrium. A solution is Pareto optimal if it does not exist any other solution in which one agent can improve its reward without decreasing the reward of another.

The model which combines reinforcement learning and game theory, is called *stochastic games* [1]. This model is a tuple $< Ag, S, A^i, \mathcal{P}, \mathcal{R}^i >$ where

- Ag is the set of agents where card(Ag) = N,
- $A^i = \{a^i_0, \cdots, a^i_p\}$ is the finite set of actions for the agent i,
- $\mathcal{P} : S \times A^1 \times \cdots \times A^N \times S \rightarrow \Delta(S)$ is the transition function from current state, agents actions and new state to probability distribution over the states,
- $\mathcal{R}^i : S \times A^1 \times \cdots \times A^N \rightarrow \mathbb{R}$ is the immediate reward function of agent i. In team Markov games, $\mathcal{R}^i = \mathcal{R}$ for all agents i.

Among the algorithms which calculate an equilibrium policy for team Markov games, Friend Q-Learning algorithm, presented by algorithm 1, introduced by Littman [11], allows to build a policy which is a Nash Pareto optimal equilibrium in team games. More specifically, this algorithm, based on Q-Learning, uses the following function for updating the Q-values at each step:

$$Q(s,\vec{a}) = (1-\alpha)Q(s,\vec{a}) + \alpha[r + \gamma \max_{\vec{a} \in \vec{A}} Q(s',\vec{a})]$$

with $\vec{a}$, the joint action for all agents ($\vec{a} = (a^1, \cdots, a^N)$).

Algorithm 1 Friend Q-Learning

```
Initialize :
Q = arbitrary Q-Value function
for all episode do
  Initialize initial state s
  repeat
    Choose a⃗ which maximize Q(s, a⃗)
    each agent i carries out action a^i
    Observe r, a^{-i}⃗ and next state s'
    Q(s, a^i⃗) = (1 − α)Q(s, a⃗) + α[r + γ max_{a⃗∈A⃗} Q(s', a⃗)]
    s = s'
  until episode termination condition
end for
```

There are many techniques for choosing the joint action at each step of the learning in this algorithm. We use in this paper the ϵ-greedy policy. That means that, at each time, each agent has a probability $1 - \epsilon$ to choose the action which maximise the Q-value for the current state and a probability ϵ to choose a uniform random action. This allows the exploration of the joint action set. The size of this set is exponential in term of the number of agents. This is a real problem for real situations where the number of agents is high.

3. GENERAL ARCHITECTURE

The general architecture for CACC is described by Figure 1. The positioning and communication systems provide information to build a world model used by the action choice module to give commands to the vehicle. However, as we said before, it is not possible to solve the problem of vehicle cooperation with only one controller designed by reinforcement learning. That is why we divided the problem into two sub-problems and, consequently, two controllers. Furthermore, the action choice module is divided in two layers: the Guidance layer and the Management layer based on Hallé's approach [7] and described in Figure 2.

The Guidance layer controller takes as inputs details on the previous state of the vehicle and, by using communication, on the state of other vehicles to take secure driving decisions. Such a control loop taking into account the state of preceding vehicles could help avoid instability of a string of vehicles controlled by CACC. Inter-vehicle communication is necessary to observe the longitudinal stability of cooperating vehicles [17]. Such stability requires knowledge of the acceleration of the leader of the platoon of the preceding vehicle, which could be provided by cooperation through the communication system. Here, we present a preliminary solution, described in section 4. It is based on reinforcement learning and is used to design an ACC acting policy. We will explain the design

of a policy for the longitudinal control of a single vehicle, a follower, that can react to acceleration changes of the leader. Those acceleration changes are given, for now, by a sensor providing the inter-vehicle distance.

On the other hand, the Management layer is in charge of vehicle coordination at high-level. In our current approach, this layer has to find the best driving lane for each vehicle on an highway system according to the other vehicles' states and actions as described in section 5. The Management layer takes as input information from sensors and from the communication system according to a certain partial view of the road. Each vehicle is able to know the current state and the actions of other vehicles in a certain range. With this fixed range, a policy is designed to choose the most effective lane for each vehicle according to the accessible information for each agent. This layer sends recommended driving actions to the Guidance layer by choosing one of the many low-level controllers (for example, following a preceding vehicle, changing lane, etc.). In this article, the low-level controller we present is the vehicle-following controller.

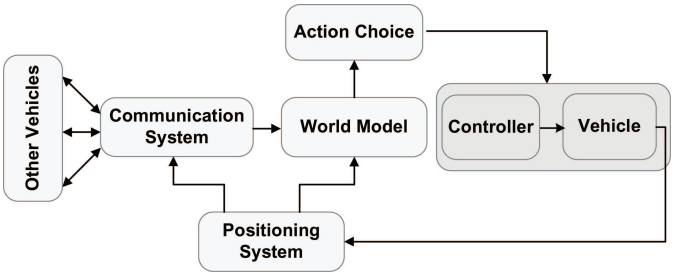

Figure 1: CACC control loop using reinforcement learning

4. REINFORCEMENT FOR GUIDANCE LAYER

Reinforcement Learning (RL) is an interesting technique for the design of a longitudinal controller because it enables us to abstract from the complexity of car physics and dynamics that have an important computing cost. With algorithms such as Q-Learning, one can learn by choosing the actions and observing their results directly in the environment. Put into an ACC context, it is possible to learn an acting policy in a simulated highway system by taking actions on the cars' brakes and throttle, and observing the results. The policy obtained can be used as a longitudinal controller to safely follow a preceding vehicle.

To apply this RL framework, we first had to model the problem by defining the states, actions, goals and rewards. Our first approach was to use variables such as the position of a leading car and of a follower, their velocities and accelerations, etc. Clearly, this state definition put us up against the curse of dimensionality, and it became impossible to have a discrete state space precise enough to learn a valuable policy. We modified our state definition by consolidating numerous state variables. This allowed us to use a smaller discretization and to reach a better precision with only two variables. Since driving can be seen as a sequential decision problem, there is no problem in modelling it using a MDP and discrete state variables. As seen in section 7, part of our future works will be to implement techniques to better approximate the continuous aspects of the problem. For now, our discrete state space was built around a state definition containing variables similar to those used in [14] for a fuzzy logic controller, as we defined our states by the relative distance in time between two vehicles and by the difference between those distances at two consecutive steps.

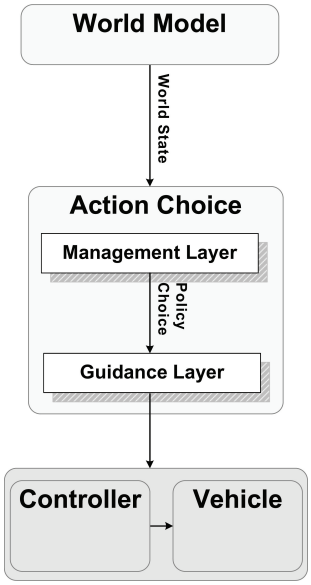

Figure 2: CACC Architecture

$$Dist_{Time} = \frac{(Position_{Leader} - Position_{Follower})}{Velocity_{Follower}} \quad (1)$$

$$\Delta Dist = Dist_t - Dist_{t-1} \quad (2)$$

As seen in Eq. (1) and Eq. (2), the time distance takes into account the relative position between the two vehicles and also the velocity of the follower, while the differences of the time distance between two consecutive time steps gives a signal about the movement of the vehicles relative to each other (whether they are closing up since last step, or getting farther). The time distance is the main variable for identifying the follower's position related to the secure distance, while the difference in time completes the Markovian signal, as it adds to the state definition an evaluation of the relative acceleration or deceleration. This relative movement between vehicles is needed to take an informed decision on the action to take at the next time step. Those actions were taken directly on the brakes or throttle (only one action per time step is chosen), closely simulating human interaction. The actions were discretized, according to a percentage of pressure on the pedal, from 0 to 100 by increments of 20.

The goal was defined as a secure distance to reach behind a preceding vehicle. That distance was specified as a time range and was defined as 2 seconds ($\pm$ 0.1 sec.), as it is a value often used as a secure distance in today's ACC systems [2]. To reach the goal, we set the rewards accordingly, with a positive reward given when the vehicle was located in the specified time range. We also set negative rewards when wandering too far or too close from the time ratio we were looking for. The behaviour the agent was supposed to learn was to reach the secure distance specified as the goal, and to stay in that range for as long as possible.

Those elements were put together in a RL framework, and the policy obtained, learned in a simulated environment, formed the core of our longitudinal controller. The environment, a simulated highway system built in previous work, featured complex car physics

and dynamics as described in [9]. Since the simulation environment was using continuous time, we had to define the time interval at which action decisions would be taken. The action chosen at the specified time frame would be taken for the whole frame. To observe an accurate behaviour of the vehicle, we had to set the time step between each action decision to a small value (50 milliseconds). But in such conditions, the observation of real vehicle acceleration needed many consecutive acceleration actions, a behaviour that could not be learned in a decent time with normal state space exploration. To overcome this problem, we had to use a heuristic to speed up learning. The heuristic specified that every time the car was behind the desired time ratio, the best acceleration action known from experience was taken. By ignoring in that case the braking actions, this action selection technique directed rapidly the agent towards more rewarding locations of the state space.

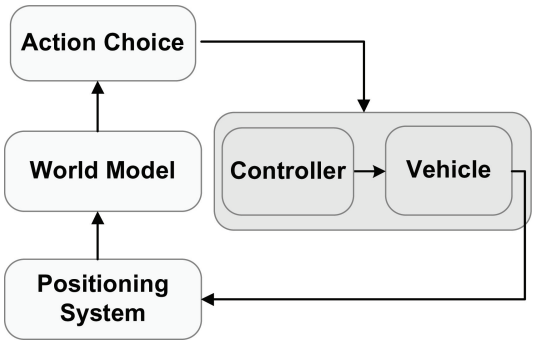

Figure 3: ACC control loop based on reinforcement learning

Put into context, Figure 3 shows that using RL simplifies the design of a longitudinal controller. The closed-loop controller takes as inputs the vehicle's state as described earlier, and selects the appropriate action according to the policy that was learned. Such a technique is obviously simpler than the complex mathematical analysis needed to predict precise car physics and dynamics for acting, as our controller basically hides in a black box vehicle physics and dynamics. It is possible for the agent to learn the optimal behaviour by taking driving actions and observing their results on the time distance and its difference between two time steps. In the next section, we will show results obtained by using this policy for longitudinal vehicle control.

5. COORDINATION BY REINFORCEMENT LEARNING: THE MANAGEMENT LAYER

In this section, we describe the design of the Management layer and, more precisely, the design of the policy to select the most efficient and safest lane for each vehicle according to their current state and action.

5.1 Problem Description

Coordination of vehicles is a real world problem with all the difficulties that can be encountered: the environment is partially observable, multi-criteria, has complex dynamic, and is continuous. Consequently, we establish many assumptions to simplify the problem and apply multiagent reinforcement learning algorithms to solve it.

The vehicle coordination problem presented here is adapted from Moriarty and Langley [12]. More precisely, three vehicles, each

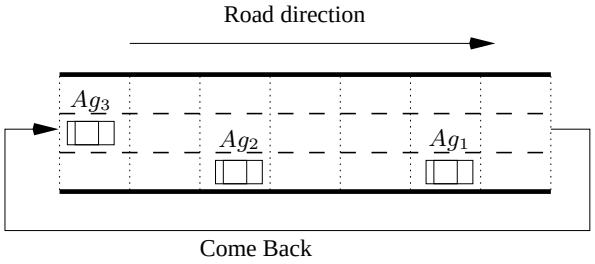

Figure 4: Initial state for 3×7 problem

represented by an agent, have to coordinate themselves to maintain velocity and avoid collisions. Each vehicle is represented by a position and a velocity. The goal of the learning algorithm is to find the best policy for each agent in order to maximize the common reward and also to avoid collision. The common reward is defined as the average velocity at each turn.

Figure 4 represents the problem's initial state. The environment's dynamics, states and actions are sampled in the easiest way: each case represents one meter and we assume that each vehicle can enter in a case. This simple discretization is repaired by the guidance layer, which will effectivelly calculate and apply the real control on the vehicle. This allows to handle uncertainty on position and velocity at this level of decision. The vehicles' dynamics are simplified to the following first order equation with only velocity $y(t) = v \times t + y_0$. For this example, we simulate the road as a ring meaning that a vehicle is returned to the left side when it quits through the right. The state of the environment is described by the position x^i, y^i and the velocity v^i of each agent i. Collisions occur when two agents are located in the same tile. The agents do not know the transitions between states. Those transitions are calculated according to the velocities of the agents and their actions. At every step, each vehicle tries to accelerate until a maximum of $5\ m/s$ is reached. If another vehicle is in front of him, the agent in charge of the vehicle sets its velocity to the front vehicle's velocity. At each step, a vehicle can choose three actions: stay on the same lane, change to the right lane and change to the left lane. Each episode has a maximum of 10 steps. The reward at each step is set to the average velocity among all vehicles. If a collision occurs, the episode stops. The size of the set of states is in $O((X \times Y \times |V|)^N)$ with X the number of lanes, Y the length of the road, V the set of possible velocities and N the number of agents. We assume, in this problem, that each agent controlling one vehicle is able to see only its own local state (position, velocity). To obtain the states of other agents, we assume that communication is needed.

5.2 Partial Observability

In this section, we introduce our approach by describing the Friend Q-learning algorithm with a local view for the agents. Then, we introduce the same algorithm but using a partial local view of distance d. This partial local view allows the reduction of the set of states and/or the set of joint actions. If no reduction is done, the exact algorithm associated is Friend Q-learning. When only the set of states is reduced, we propose Total Joint Actions Q-learning (TJA). From this algorithm, we reduce the set of joint actions and we propose another algorithm: Partial Joint Actions Q-learning (PJA). In this article, we do not consider the reduction of joint actions alone, because this reduction is lower than the reduction of the set of states.

5.2.1 FriendQ with a local view

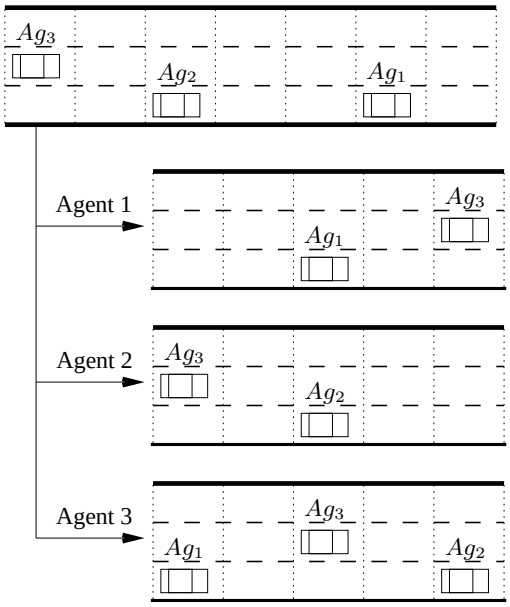

Figure 5: State and Partial States for $d = 2$

To introduce partial observability, we define the notion of local Q-Value and local state. Each agent uses the same algorithm but on different states. A local state is defined from the real state of the multiagent system for a center agent. All other agents positions are defined relatively to this central agent. This means that the same real state belonging to the set S will give different local states. For an agent i, the set of possible local state is S^i. We introduce a function f^i which transforms the real state s to a local state s^i for agent i. Formally, $\forall s \in S, \exists s^i \in S^i$ such that $f^i(s) = s^i$ for all agents i. In this version of the algorithm, each agent uses Friend Q-learning algorithm as described in section 2 but updates its Q-values for the local states and not for the real state.

5.2.2 FriendQ with a partial local view

To measure the effect of partial observability on the performance we define the partial state centered on one agent by introducing a distance of observability d. Consequently, the initial problem becomes a d-partial problem. The distance d can be viewed as an influence area for the agent. Increasing this distance increases the degree of observability. Moreover, from a communication point of view, in real world problems, the communication cost between two agents depends on the distance between them. Communicating with a remote agent is costlier than with a close agent. We define d_{total} as the maximal possible distance of observability for a given problem.

In d-partial problem, the new state is defined as the observation of the center agent for a range d. More precisely, an agent j is in the partial state of a central agent i if its distance is lower or equal than d from the central agent i. Formally, the function f_d^i uses the parameter d to calculate the new local state. Figure 5 provides an example of the application of f_d^i on a state s and gives the resulting partial states for each agent with a distance $d = 2$. Agent 1 sees only Agent 3 but Agent 3 sees both Agent 1 and 2. The new size of the set of states is $O(((2d+1)^2 \times V)^N)$. The number of states is divided by approximately $(Y/(2d+1))^N$, if we neglect the number of lanes which is often small compared to the length of the road.

5.2.2.1 TJA Q-Learning.

In a first step, as in classical Friend Q-learning, we consider an algorithm that takes into account the complete joint actions. This assumption implies that all agents are able to communicate their actions to others at each step without cost. The Q-value update function is now :

$$Q(f_d^i(s), \vec{a}) = (1-\alpha)Q(f_d^i(s), \vec{a}) + \alpha[r + \gamma \max_{\vec{a} \in \vec{A}} Q(f_d^i(s'), \vec{a})]$$

for agent i. When $d = d_{total}$, we have a small reduction factor on the state set of XY, because we do not take into account, in our specific problem, the absolute position of the center agent.

5.2.2.2 PJA Q-learning.

In a second step, the algorithm takes into account only the actions where agents are in the partial local view as specified by d. This reduces dramatically the number of joint actions which has to be tested during the learning. This partial local observability allow us to consider a variable number of agents in the multiagent system.

Formally, we define a function g^i which transforms the joint action $\vec{a}$ into a partial joint action $g_d^i(\vec{a}, s)$. This partial joint action contains all actions of agents in the distance d of agent i. The Q-value update function is now :

$$\begin{aligned} Q(f_d^i(s), g_d^i(\vec{a}, s)) \quad = \quad & (1-\alpha)Q(f_d^i(s), g_d^i(\vec{a}, s)) \\ & +\alpha[r + \gamma \max_{\vec{a_d} \in G_d^i(\vec{A}, S)} Q(f_d^i(s'), \vec{a_d})] \end{aligned}$$

for agent i where $G_d^i(\vec{A}, S)$ returns the set of joint actions with a central agent i and a distance d. We can see that the result of the partial joint action depends on the current state.

6. EXPERIMENTS

6.1 Experiments at Guidance Layer

To learn the control policy, we used the Q-Learning algorithm, as described in section 2, with the reinforcement learning framework described in section 4. The learning task was defined as episodic, with each episode composed of 1000 steps of 50 milliseconds and we considered the task as semi-continuous, since it did not end when the agent reached the goal. The agent's optimal behaviour was to reach the goal, the secure following distance, and stay inside that range for as long as possible. The policy shown here was obtained after running 10 000 episodes.

The learning scenario used two vehicles: an automatically controlled leading car and an agent car trying to learn the control policy to follow the preceding vehicle by using our RL framework. The leading vehicle started 3 meters in front of the learner, and accelerated to reach the velocity of 20 m/s. The agent had to learn the policy by receiving rewards for trying different acceleration or braking actions.

Afterwards, we tested the resulting longitudinal controller based on the policy learned with the same scenario. Notice that the policy was run on-line once, and, as shown in Figure 6 and Figure 7, we obtained good results according to the specified time distance.

Figure 7 shows the time distance between the vehicles during the simulation. Again, we see that the agent was able to learn to stabilize around the goal that the agent was looking after. Those results illustrate that the time distance is much more variable than the distance in meters. This is in part from the fact that slight differences in relative positions or velocities of the vehicles can modify the ratio (see Eq. (1)), although the relative distance in meters between

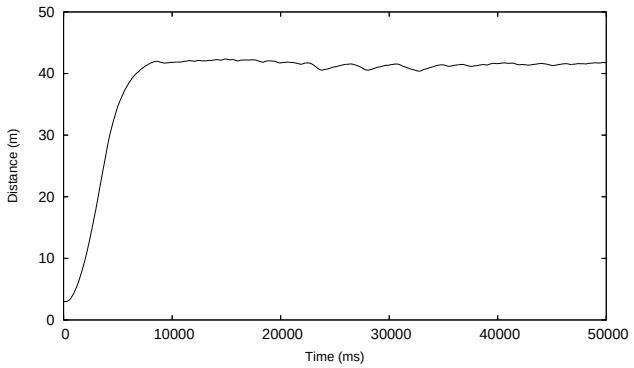

Figure 6: Distance (in m) from the preceding vehicle

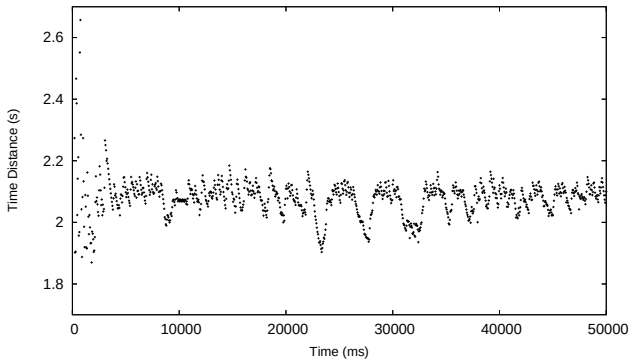

Figure 7: Time distance (in seconds) between vehicles

the vehicles might not change much. Another reason that could explain those results is the fact that our actions were not highly discretised. Hence, choosing at two consecutive steps different actions (for example, a braking action of 60%, followed by an accelerating action of 100%) might in the end affect velocity, causing important modifications on the ratio between two steps. Finally, the ratio used for the calculation of the distance in time (Eq. (1)) again explains the highly unstable values of the first seconds of the simulation (Figure 7), as the time distance is undefined when the velocity of the following vehicle is null.

In the end, we observe good stability around the safe distance which shows that we were able to learn a policy to follow the vehicle safely, without knowledge of the inner workings of the environment. Clearly, small disturbances in time ratio are not amplified but come back to the desired value.

6.2 Experiments at Management Layer

In this section, we compare empirically the performance of the totally observable problem (FriendQ) and the performance of the approximated policy (TJA and PJA). We present three kind of results: first of all, we compare the algorithms on a small problem P_1 defined by size $X = 3$, $Y = 7$, the set of velocities $V = 0, \cdots, 5$ and the number of agents $N = 3$. Consequently, in this problem, the maximal distance that we can use to approximate the total problem is $d_{total} = 3$. The 3-partial state is a local representation of the totally observable state because we are sure that all agents are visible from others in this representation. In the initial state (Figure 4), velocities of the agents are $v^1 = 1$, $v^2 = 2$ and $v^3 = 3$. We present, for all results, the average total sum reward over 25 learnings with each episode lasting 10 steps. More precisely, the reward presented on following figures uses $R = \sum_{t=1}^{10} \overline{v_t}$ where $\overline{v_t}$ is the average velocity over all vehicles at each step t of the episode. The y-axis is consequently the average of R over 25 learnings.

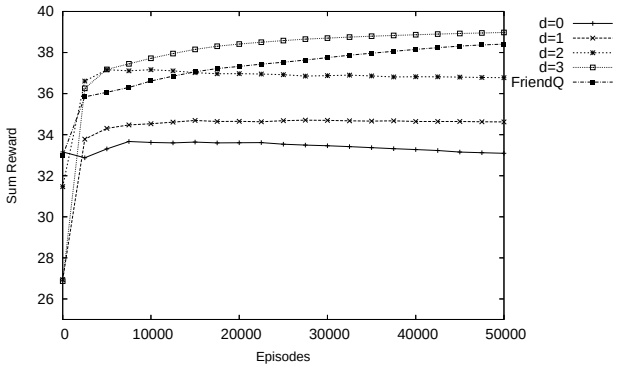

Figure 8: Rewards for Total Joint Action Q-learning for problem P_1

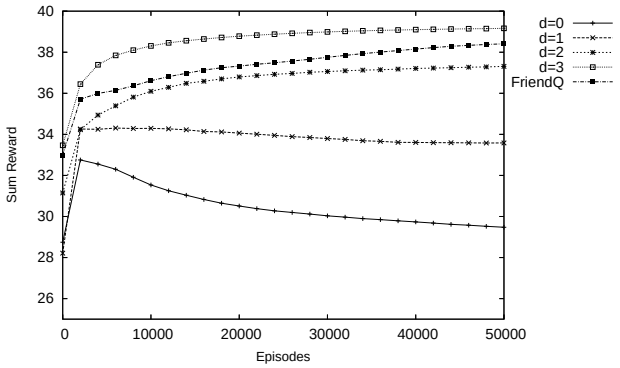

Figure 9: Rewards for Partial Joint Action Q-learning for problem P_1

Figure 8 shows the result of TJA Q-learning with distance from $d = 0$ to $d = 3$. This algorithm is compared to the total observation problem resolved by Friend Q-Learning. For $d = 0$, $d = 1$ and $d = 2$, TJA converges to a local maximum, which increases with d. In these cases, the approximated values are respectively of about 86%, 89% and 94% of the optimal value. When $d = 3$, that is, when the local view is equivalent to the totally observable view, the average sum rewards converges to the total sum rewards of Friend Q-learning. However, since we do not take into account the absolute position of the center agent, TJA converges faster than Friend Q-learning. Figure 9 shows the results of PJA Q-Learning on the same problem. As previously, for $d = 0$, $d = 1$ and $d = 2$, PJA converges to a local maxima respectively of about 76%, 86% and 97%. These values are lower than TJA's value but, for $d = 2$, the value is still close to the optimal.

For the second result, we compare PJA Q-learning for two different problems. We define a correct approximation distance d_{app} for each problem, where the associated policy is close to the optimal value. In the vehicle coordination problem presented here, the optimal value is the best lane choice. The first problem is the same as

previously (Figure 9) and we can show that $d_{app} = 3$ for this problem. In the second problem P_2, we enlarge the number of lanes and the length of the road ($X = 5$, $Y = 20$, $V = 0, \cdots, 5$ and $N = 3$). This problem increases the number of states but decreases the possible interactions between vehicles because they have more space. For the second problem P_2, Figure 10 shows the comparison between Friend Q-learning and PJA Q-learning from $d = 0$ to $d = 7$. We can see that from $d = 4$, there are only small differences between PJA and Friend Q-learning. Consequently, for this problem, we can see that $d_{app} = 4$. The difficulty of this approach is the need to calculate the optimal policy, which can be intractable, to get d_{app}.

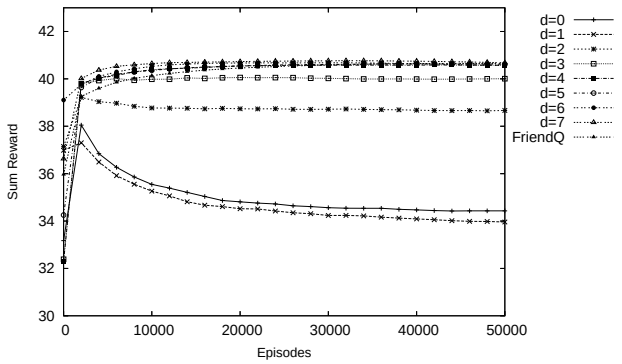

Figure 10: Rewards for Partial Joint Action Q-learning for problem P_2

As we can see, we need to generalize this result to know the d_{app} parameter without calculating the optimal policy. To present the third result, we calculate the ratio $DS = XY/N$ which represents the degree of space for each agent. Obviously, if the space (X or Y) increases, then each agent has more space for itself. As we study a problem where the team of agents has to handle only negative interaction, the higher the ratio, the more space agents have. We compare the performance of our PJA algorithm for different ratios. The ratios for the first two problems are respectively $DS_{P_1} = 7$ and $DS_{P_2} = 33$. We add two new problems P_3 ($X = 5$, $Y = 20$, $V = 0, \cdots, 5$ and $N = 5$) and P_4 ($X = 6, Y = 28, V = 0, \cdots, 5$ and $N = 4$) where the ratios are respectively of 20 and 42. Table 1 presents the results for each problem after 50000 episodes. For each problem, we define the correct approximation distance d_{app} such as $1 - (\frac{R_{d_{app}}}{R_{friendQ}}) < \epsilon$. When $\epsilon = 0.01$, $d_{app}^{P_1} = 3$, $d_{app}^{P_2} = 4$, $d_{app}^{P_3} = 2$ and $d_{app}^{P_4} = 2$.

To discover a relation between the ratio DS and the value of d_{app}, we compare in Figure 11, the link between DS and the degree of observability, defined as $\frac{d_{app}}{d_{total}}$ where d_{total} is the maximal distance for a given problem. For example, d_{total} for the problem P_1 is 3. We can see that the degree of observability decreases with the degree of space for each agent. We calculate an interpolated curve assuming that the degree of observability cannot be higher than 1 when $DS < 7$. We can see that the needed observability decreases and tends to 0 when DS increases. With this relation between both the observability and the degree of space, we can evaluate, for other problems how would be the d_{app} value.

Thus, introducing the locality of the view allows us to limit the observability of the state. More precisely, this approach allows us to use the partial version of Friend Q-learning in real world problems where the state is always partially observable. We obtain an approximation of the optimal policy without knowing the transition function. This approximation can be very close to the optimal policy.

Algorithms	P_1	ϵ_{P_1}	P_2	ϵ_{P_2}
FriendQ	38.4 ± 1.1	-	40.6 ± 0.3	-
PJA $d = 7$	-	-	40.6 ± 0.2	$\sim 0\%$
PJA $d = 6$	-	-	40.5 ± 0.2	$\sim 0\%$
PJA $d = 5$	-	-	40.6 ± 0.2	$\sim 0\%$
PJA $d = 4$	-	-	40.5 ± 0.2	$\sim 0\%$
PJA $d = 3$	39.1 ± 0.2	$\sim 0\%$	40.0 ± 0.2	$\sim 2\%$
PJA $d = 2$	37.3 ± 0.2	$\sim 3\%$	38.6 ± 0.2	$\sim 5\%$
PJA $d = 1$	33.5 ± 0.2	$\sim 14\%$	33.9 ± 0.3	$\sim 15\%$
PJA $d = 0$	29.4 ± 0.3	$\sim 24\%$	34.4 ± 0.4	$\sim 15\%$
Algorithms	P_3	ϵ_{P_3}	P_4	ϵ_{P_4}
FriendQ	37.0 ± 1.2	-	37.6 ± 0.3	-
PJA $d = 7$	37.2 ± 0.7	$\sim 0\%$	38.4 ± 0.2	$\sim 0\%$
PJA $d = 6$	37.9 ± 0.7	$\sim 0\%$	38.8 ± 0.4	$\sim 0\%$
PJA $d = 5$	37.8 ± 0.9	$\sim 0\%$	38.7 ± 0.4	$\sim 0\%$
PJA $d = 4$	38.3 ± 0.8	$\sim 0\%$	38.7 ± 0.2	$\sim 0\%$
PJA $d = 3$	38.7 ± 0.6	$\sim 0\%$	38.9 ± 0.2	$\sim 0\%$
PJA $d = 2$	37.7 ± 0.5	$\sim 0\%$	38.5 ± 0.1	$\sim 0\%$
PJA $d = 1$	35.2 ± 0.3	$\sim 5\%$	35.1 ± 0.4	$\sim 8\%$
PJA $d = 0$	33.5 ± 0.4	$\sim 10\%$	34.3 ± 0.3	$\sim 11\%$

Table 1: Average Rewards and standard deviation after 50000 episodes

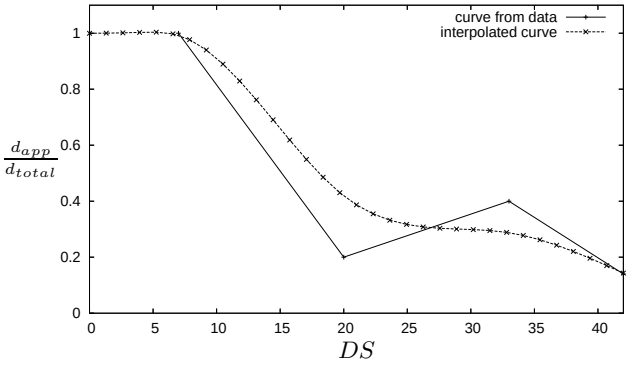

Figure 11: Link between observability and degree of space

In our approach, we do not explicitly take into account communication for many reasons. First of all, in real world problems, choosing the right communication cost is not an easy task. Furthermore, as we said previously, the communication cost depends not only on the messages sent but also on the distance between senders and receivers. This problem complicates the design of communication cost. Knowing the value of the approximated policy and of the associated communication policy (and consequently, the cost of this policy) to obtain the n-partial state, the multiagent system designer can get a good approximation for the real world problem.

7. FUTURE IMPROVEMENTS

The Guidance layer could benefit from a few ameliorations in order to raise its effectiveness and precision. First, as noted in section 4, it is clear that with our current discretization technique, it is necessary to use a heuristic to direct the state space exploration and

to observe learning in decent time. However, since the use of the heuristic cannot give us the certainty that every state will be visited, that technique leaves a lot of states unexplored, yet some of those unexplored states could eventually be visited while acting on-line. As a solution, we will be looking to discretize while learning (instead of at the initialization phase) the state variables describing the policy. We could use techniques such as those proposed by Munos [13] where the policy could be discretized according to the areas of the state space where extra precision is needed. We could also discretize actions using such a technique, where refined actions could possibly be used according to the level of discretization of the current state. The use of such a technique could help us approximate continuous actions and state variables and have a more precise control policy.

We would also like to use and take advantage of inter-vehicle communication in the learning process for low-level longitudinal controllers. Vehicle communication could provide environment information for the RL process, and the agent could learn secure longitudinal control policies to be used in environments where multiple vehicles would be involved. We hope to design a CACC longitudinal controller using reinforcement learning, where information on the state of other vehicles would be transmitted and taken into account in the decisions of a driving agent (resulting in a control loop similar to the one in Figure 1). More specifically, we would like to observe how those controllers are doing according to the string stability of a platoon of vehicles using such policies.

We would like to use our learning algorithm on different scenarios (for example, hard-braking of the leader, stop and go situations, etc.) as to obtain a controller that could react to most driving situations.

As for the Management layer, we plan to evaluate more theoretically the relationship between the degree of observability and the performance of the learned policy. To define some formal bounds, we will certainly need to use complex communication cost. Finally, introducing the physical distance for a measure of observability is basic. We plan to discover others kind of distance between agents to measure observability to generalize our approach to positive and negative interaction management problems in teams. Finally, it will be very interesting to study the effect of partial local view to non-cooperative cases.

Finally, since both of our two RL approaches seem to give good results separately, we will look to integrate them into a fully functional CACC system. Consequently, the next step of our work will be to integrate both approaches: the high-level Management Layer will be taking decisions to select the best low-level controller according to its assessment of the multiagent environment.

8. RELATED WORK

A lot of related work has been done in recent years in the design of CACC systems. Regarding the vehicle-following controller, Hallouzi *et al.* [8] did some research as part of the CarTalk 2000 project. These authors worked on the design of a longitudinal CACC controller based on vehicle-to-vehicle communication. They showed that inter-vehicle communication can help reduce instability of a platoon of vehicles. In the same vein, Naranjo and his colleague [14] worked on designing a longitudinal controller based on fuzzy logic. Their approach is similar to what we did with reinforcement learning for our low-level controller. Forbes has presented a longitudinal reinforcement learning controller [5] and compared it to a hand-coded following controller. He showed that the hand-coded controller is more precise than its RL controller but less adaptable in some situations. However, Forbes did not test explicitly communication between vehicles to improve its longitudinal controller to a multi-vehicle environment (which will be the focus of our future work). Our approach will also integrate our low-level controllers with a high-level multiagent decision making algorithm, which was not part of Forbes' work.

Regarding the reinforcement learning in a vehicle coordination problem, Ünsal, Kachroo and Bay [21] have used multiple stochastic learning automata to control the longitudinal and lateral path of a vehicle. However, the authors did not extend their approach to the multiagent problem. In his work, Pendrith [15] presented a distributed variant of Q-Learning (DQL) applied to lane change advisory system, that is close to the problem described in this paper. His approach uses a local perspective representation state which represents the relative velocities of the vehicles around. Consequently, this representation state is closely related to our 1-partial state representation. Contrary to our algorithms, DQL does not take into account the actions of the vehicles around and updates Q-Values by an average backup value over all agents at each time step. The problem of this algorithm is the lack of learning stability.

On the other hand, our high level controller model is similar to Partially Observable Stochastic Games (POSG). This model formalizes theoretically the observations for each agent. The resolution of this kind of games has been studied by Emery-Montermerlo [4]. This resolution is an approximation using Bayesian games. However, this solution is still based on the model of the environment, unlike our approach which does not take into account this information explicitly since we assume that the environment is unknown. Concerning the space search reduction, Sparse Cooperative Q-Learning [10] allows agents to coordinate their actions only on predefined set of states. In the other states, agents learn without knowing the existence of the other agents. However, unlike in our approach, the states where the agents have to coordinate themselves are selected statically before the learning process. The joint actions set reduction has been studied by Fulda and Ventura who proposed the Dynamic Joint Action Perception (DJAP) algorithm [6]. DJAP allows a multiagent Q-learning algorithm to select dynamically the useful joint actions for each agent during the learning. However, they concentrated only on joint actions and they tested only their approach on problems with few states.

Introducing communication into decision has been studied by Xuan, Lesser, and Zilberstein [20] who proposed a formal extension to Markov Decision Process with communication where each agent observes a part of the environment but all agents observe the entire state. Their approach proposes to alternate communication and action in the decentralized decision process. As the optimal policy computation is intractable, the authors proposed some heuristics to compute approximation solutions. The main differences with our approach is the implicit communication and the model-free learning. More generally, Pynadath and Tambe [16] have proposed an extension to distributed POMDP with communication called COM-MTDP, which take into account the cost of communication during the decision process. They presented complexity results for some classes of team problems. As Xuan, Lesser, and Zilberstein [20], this approach is mainly theoretical and does not present model-free learning. The locality of interactions in a MDP has been theoretically developed by Dolgov and Durfee [3]. They presented a graphical approach to represent the compact representation of a MDP. However, their approach has been developed to solve a MDP and not to solve directly a multiagent reinforcement learning problem where the transition function is unknown.

9. CONCLUSION

In this article, we presented a preliminary CACC approach which combines a low-level controller to carry out low-level actions such

as following vehicles and a high-level controller which coordinates vehicles and chooses the right low-level controller according to the state of other vehicles. These controllers have been designed using reinforcement learning techniques and game theory for multiagent coordination. We showed that reinforcement learning can provide very interesting results for the efficiency of the low-level ACC controller as well as for coordination control. This article showed promising results for complete CACC design using reinforcement learning.

However, much work has to be done to implement every CACC functionalities with reinforcement learning techniques. Even though we described vehicle-following control and lane-changing coordination, many other control policies could be added. We plan to improve efficiency of our approaches and integrate them into our general architecture to test it in a realistic vehicle simulator.

10. REFERENCES

[1] T. Basar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Classics In Applied Mathematics, 2nd edition, 1999.

[2] W. R. Bishop. *Intelligent Vehicle Technology and Trends*. Artech House, 2005. ISBN = 1-58053-911-4.

[3] D. Dolgov and E. H. Durfee. Graphical models in local, asymmetric multi-agent Markov decision processes. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-04)*, 2004.

[4] R. Emery-Montermerlo. Game-theoretic control for robot teams. Technical Report CMU-RI-TR-05-36, Robotics Institute, Carnegie Mellon University, August 2005.

[5] J. R. Forbes, T. Huang, K. Kanazawa, and S. J. Russell. The batmobile: Towards a bayesian automated taxi. In *Fourteenth International Joint Conference on Artificial Intelligence*, pages 418–423, 1995.

[6] N. Fulda and D. Ventura. Dynamic Joint Action Perception for Q-Learning Agents. In *2003 International Conference on Machine Learning and Applications*, 2003.

[7] S. Hallé and B. Chaib-draa. A Collaborative Driving System based on Multiagent Modelling and Simulations. *Journal of Transportation Research Part C (TRC-C): Emergent Technologies*, 13(4):320–345, 2005.

[8] R. Hallouzi, V. V., H. H., P. Morsink, and P. J. Communication based longitudinal vehicle control using an extended kalman filter. In *IFAC Symposium on Advances in Automotive Control*, Salerno, 2004.

[9] S. Hallé. Automated highway systems: Platoons of vehicles viewed as a multiagent system. Master's thesis, Université Laval, Québec, 2005.

[10] J. R. Kok and N. Vlassis. Sparse Cooperative Q-learning. In R. Greiner and D. Schuurmans, editors, *Proc. of the 21st Int. Conf. on Machine Learning*, pages 481–488, Banff, Canada, july 2004. ACM.

[11] M. Littman. Friend-or-Foe Q-learning in General-Sum Games. In M. Kaufmann, editor, *Eighteenth International Conference on Machine Learning*, pages 322–328, 2001.

[12] D. E. Moriarty and P. Langley. Distributed learning of lane-selection strategies for traffic management. Technical report, Palo Alto, CA., 1998. 98-2.

[13] R. Munos. *L'apprentissage par renforcement, étude du cas continu*. PhD thesis, Ecole des Hautes Etudes en Sciences Sociales, Antony, 1997.

[14] J. Naranjo, C. Gonzalez, J. Reviejo, R. Garcia, and T. de Pedro. Adaptive fuzzy control for inter-vehicle gap keeping. *Intelligent Transportation Systems, IEEE Transactions on*, 4(3):132 – 142, Sept. 2003.

[15] M. D. Pendrith. Distributed reinforcement learning for a traffic engineering application. In *the fourth international conference on Autonomous Agents*, pages 404 – 411, 2000.

[16] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of AI research*, 16:389–423, 2002.

[17] D. Swaroop. *String Stability of Interconnected systems : an application to platooning in automated highway systems*. PhD thesis, Departement of Mechanical Engineering, University of California, Berkeley, 1994.

[18] S. Tsugawa. Issues and recent trends in vehicle safety communication systems. *LATSS Research*, 29(1):7–15, 2005.

[19] P. Varaiya. Smart cars on smart roads : Problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.

[20] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: model and experiments. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *the Fifth International Conference on Autonomous Agents*, pages 616–623, Montreal, Canada, 2001. ACM Press.

[21] C. Ünsal, P. Kachroo, and J. S. Bay. Simulation study of multiple intelligent vehicle control using stochastic learning automata. *IEEE Transactions on Systems, Man and Cybernetics - Part A : Systems and Humans*, 29(1):120–128, 1999.