Batch Reinforcement Learning for Optimizing Longitudinal Driving Assistance Strategies

Olivier Pietquin
SUPELEC
UMI 2958 (GeorgiaTech - CNRS)
2 rue Édouard Belin
57070 Metz (France)
olivier.pietquin@supelec.fr

Fabio Tango
Centro Ricerche Fiat
Strada Torino, 50
10043 Orbassano (Italy)
fabio.tango@crf.it

Raghav Aras
SUPELEC
IMS Research Group
2 rue Édouard Belin
57070 Metz (France)
raghav.aras@supelec.fr

***Abstract*—Partially Autonomous Driver's Assistance Systems (PADAS) are systems aiming at providing a safer driving experience to people. Especially, one application of such systems is to assist the drivers in reacting optimally so as to prevent collisions with a leading vehicle. Several means can be used by a PADAS to reach this goal. For instance, warning signals can be sent to the driver or the PADAS can actually modify the speed of the car by braking automatically. An optimal combination of different warning signals together with assistive braking is expected to reduce the probability of collision. How to associate the right combination of PADAS actions to a given situation so as to achieve this aim remains an open problem.**

In this paper, the use of a statistical machine learning method, namely the reinforcement learning paradigm, is proposed to automatically derive an optimal PADAS action selection strategy from a database of driving experiments. Experimental results conducted on actual car simulators with human drivers show that this method achieves a significant reduction of the risk of collision.

I. Introduction

A. Context

The work presented in this paper has been realized inside the co-funded European project ISi-PADAS[1]. This project aims at developing and implementing a Partially Autonomous Driver Assistance System (PADAS in short) in a static driving simulator, including the interfaces between the driver and the system (comprising haptic, visual and audio interfaces). The development of such a system was based on the in-depth analysis of the car-accidents, conducted in the project, about causes of driver's errors responsible for rear-end crashes [1]. For this aim, the dataset comprising 4256 accidents from Braunschweig 2002 has been used [2]. Additionally, a sample from the German National Accident Database (Statistisches Bundesamt) from 2002 was used including 185004 accidents [3]. All rear-end crashes take part for 73.3% of analysed accidents and in particular 22.81% of the severe accidents (with major damage of more than 6000 Euro, injuries or fatalities). Following a vehicle too closely represents the most important cause of accident in 85,61% of all the cases analyzed. Moreover, 75,98% of the rear-end crashes occur in undisturbed/flow traffic (where the expectation of an event is probably low). In this context, many studies have shown the benefits of Forward Collision Warning (FCW) and of Adaptive Cruise Control (ACC) in reducing the number and severity of rear-end collisions [4], especially in conditions where drivers have to cope with car following tasks in limited traffic flows or heavy - but not congested - traffic[2]. In this context, the PADAS developed in the project was focused in assisting drivers during their longitudinal driving task and it is called Longitudinal Support System (or LoSS, in short). In particular, it can prevent a collision with a leading vehicle by providing warnings to the driver, up to bringing the vehicle to a halt independently of driver's action, through the support for an assisted braking action. The system has two mechanisms at its disposal in order to realize this collision-avoidance capability:

- it can provide warning signals to the driver
- it can decelerate the vehicle.

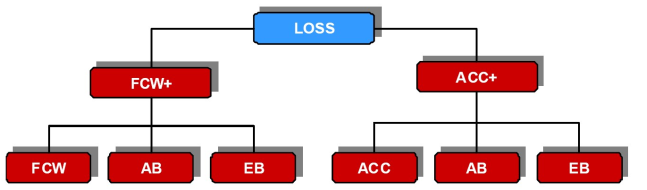

Fig. 1. architecture of the LOSS application, showing the two modalities and the constituting functions

Fig. 1 illustrates the functional scheme of PADAS application: Two types (or modes) of LOSS have been considered: the Advanced Forward Collision Warning (FCW+, in short) and the Advanced Adaptive Cruise Control (ACC+, in short) which are both constituted by 3 functions:

- Forward Collision Warning (FCW) or Adaptive Cruise Control (ACC), respectively for FCW+ and for ACC+

[1]www.isi-padas.eu

[2]see http://www.prevent-ip.org for several field test performed by Dutch Ministry of Transport or see [5]

- Assisted Braking (AB), for both
- Emergency Braking (EB), for both

FCW represent the “traditional”forward collision warning system: if the driver travels too close (too short headway) or too fast with respect to the vehicle ahead on the same trajectory, s/he receive a warning of different severities, depending on the dynamic conditions between the two vehicles. This is the first function for the FCW+ system. Alternatively, the first function can be the “traditional” ACC, in which not only the speed is kept at the specific value (like in the Cruise Control), but also the distance is kept within pre-defined threshold of headway. ACC is used as first function in the ACC+ system. Then, the other two functions are common to both the systems. In particular, AB function provides the proper assistance to the driver: if the driver acts on the brake pedal, thus indicating the will to brake, but not performing in the most appropriated way, the system is able to modulate the braking action automatically. Finally, if the driver ignored warning and AB did not intervene (i.e. driver did not apply on the brake) EB function acts, in order to avoid accidents or at least to minimise the effects, in case it is not avoidable anymore.

B. Contribution

This paper addresses the problem of finding an optimal warning and intervention strategy for a partially autonomous driver’s assistance system (PADAS). Here, an optimal strategy is regarded as the one minimizing the risk (probability) of collision with an obstacle ahead, while keeping the number of warnings and interventions as low as possible, in order to support the driver and avoid distraction or annoyance. A novel approach to this problem is proposed, based on the solution of a sequential decision making problem solved via the Reinforcement Learning (RL) [6] paradigm and learning on a fixed dataset.

In the RL paradigm, a learning agent (the controller or decision maker) learns an optimal control strategy by either interacting with the system or observing interactions between another controller and the system. To do so, the system’s dynamic as to be cast into the Markov Decision Processes (MDP) formalism [7]. The system is then considered as made up of *states* (situations to which a control as to be associated) and the control is considered as being a sequence of *actions* applied to the system. In every situation, the controller has to choose an action to apply to the system. Once the action has been applied, the system steps to another state according to its own dynamics and a *reward* qualifying the quality of the transition is generated. The controller has thus to learn a mapping between situations and actions (named as *strategy* or *policy*) that maximises the long run reward.

The collision avoidance problem is thus seen as a sequential decision making problem to optimize given a set of {vehicle+driver}-PADAS interaction sequences in which each sequence $\{s_1, a_1, s_2, a_2, \ldots, s_T, a_T\}$ consists of the vehicle+driver’s states $\{s_i\}$ and the PADAS’ reactions $\{a_i\}$ to it. Each such sequence engenders a probability of collision, and an optimal strategy is one which gives rise to a subset of this set of sequences for which the average or expected probability of collision is the smallest. In our problem, we do not know the probability of collision associated with any given sequence. The only data that is available to us consists only of small subset of the set of sequences of {vehicle+driver}-system interactions. On the other hand, we must find a strategy that takes into account all possible vehicle-system interaction sequences.

Few attempts to use RL in the case of autonomous driving can be found in the literature and most of them are related to steering the vehicle [8], [9], [10]. In this paper, we focus on RL for a continuous longitudinal support to the driver, which starts from the pre-collision and collision warnings for the driver and ends to the automatic emergency braking, through an assisted braking action. Therefore, we consider specifically the longitudinal driving task - car-following situation - and in that case, the driver is completely in the control-loop of the vehicle for the first two stages of the supporting function, while the system takes the full control of the vehicle only in the last phase of the emergency braking. In this context, the decision maker has to take into account driver’s behavior in the dynamic of the controlled system which is not case in the steering task. The fact that the driver stays in the loop makes the system highly stochastic (and maybe non-stationary) because of inter- and intra-driver variability, that is to say that to similar situations from the point of view of the PADAS, different reactions can occur given that the driver is different or the perception of the situation is different even for a same driver.

The rest of this paper is organized as follows. Section II presents the theoretical background of Reinforcement Learning and Markov Decision Processes. Section III presents how the problem of PADAS strategy optimisation can be cast into the MDP paradigm. In Section IV, experimental results are provided. Finally, Section IV concludes and provides possible future investigations.

II. Markov Decision Processes

The problem of optimally controlling a stochastic dynamic system is often addressed by the machine learning community in the framework of the Markov Decision Processes (MDP) [7]. An MDP is a stochastic finite state machine which is formally described in the next section.

A. Definition

An MDP is defined as a tuple $\{S, A, P, R, \gamma\}$ where S is the set of states (the different configurations of the system), A is the set of actions (which cause a change of the system’s state), $P : s, a \in S \times A \to p(.|s, a) \in \mathcal{P}(S)$ is a set of Markovian transition probabilities (the probability to transit from one state s to another s' given that action a was taken in state s), a reward function

$R : s, a, s' \in S \times A \times S \to r = R(s, a, s') \in \mathbb{R}$ associating a scalar to each transition and a discounting factor γ which decreases long-term rewards' influence.

The Markov property means that the probability to step from one state s to another s' given an action a depends only on s and a and not on the previously visited states and taken actions :

$$P(s_{i+1}|s_i, a_i, s_{i-1}, a_{i-1}, \ldots, s_0, a_0) = P(s_{i+1}|s_i, a_i) \quad (1)$$

The way the controller selects actions is modeled by a so-called *policy*, $\pi : s \in S \to \pi(.|s) \in \mathcal{P}(A)$, which associates to each state a probability distribution over actions. The quality of such a policy is quantified by a so-called *value function*,

$$V^\pi(s) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, \pi],$$

which associates to each state the expected cumulative discounted reward from starting in the considered state and then following the given policy. An optimal policy π^* is one of those which maximize the associated value function for each state:

$$\pi^* \in \operatorname*{argmax}_{\pi} V^\pi.$$

One can also define the *state-action value* (or Q-) *function* which provides an additional degree of freedom on the first action to be chosen:

$$Q^\pi(s, a) = E[\sum_{i=0}^{\infty} \gamma^i r_i | s_0 = s, a_0 = a, \pi]. \quad (2)$$

An optimal deterministic policy can thus be obtained by maximizing the optimal Q-function over actions:

$$\pi^* = \operatorname*{argmax}_{a \in A} Q^*(s, a). \quad (3)$$

The Q-function generalizes the value function in the sense that $V^\pi(s) = E_{a|s,\pi}[Q^\pi(s, a)]$. This is why only the Q-function will be considered in the rest of this paper. Notice that knowing the $Q^*(s, a)$ function for a given MDP is enough to derive an optimal policy. So most of algorithms aim at estimating this Q^* function.

B. Dynamic Programming

Because of the Markovian property of the transition probabilities (see eq. (1)), the expression of the Q-function (eq. (2)) can be rewritten as:

$$\begin{aligned} Q^\pi(s, a) &= E_{s'|s,a}[R(s, a, s') + \gamma Q^\pi(s', \pi(s'))] \\ &= T^\pi Q^\pi(s, a) \end{aligned} \quad (4)$$

This is the so-called Bellman evaluation equation and T^π is the Bellman evaluation operator.

The optimal Q-function $Q^*(s, a)$ can also be written as a Bellman equation (the Bellman optimality equation, where T^* is the Bellman optimality operator):

$$\begin{aligned} Q^*(s, a) &= E_{s'|s,a}[R(s, a, s') + \gamma \max_{b \in A} Q^*(s', b)] \\ &= T^* Q^*(s, a) \end{aligned} \quad (5)$$

Dynamic programming (DP) [11] aims at computing the optimal policy π^* using Bellman equations if the transition probabilities and the reward function are known. Two classes of algorithms can be distinguished.

First, the *policy iteration* algorithm computes the optimal policy in an iterative way. The initial policy is arbitrary set to π_0. At iteration k, the policy π_{k-1} is evaluated, that is the associated Q-function $Q^{\pi_{k-1}}(s, a)$ is computed. To do so, eq. (4) is used. Since T^π is linear eq. (4) defines a linear system that can be solved by standard methods or by an iterative method using the fact that Q^π is the unique fixed-point of the Bellman evaluation operator (T^π being a contraction that is $\|T^\pi V^1 - T^\pi V^2\| \leq \alpha \|V^1 - V^2\|$ with $\alpha \in]0, 1]$):

$$\hat{Q}^\pi_i = T^\pi \hat{Q}^\pi_{i-1}, \quad \forall \hat{Q}^\pi_0 \lim_{i \to \infty} \hat{Q}^\pi_i = Q^\pi.$$

Roughly, this means any policy π can be evaluated by starting from an arbitrary initial Q-value Q^π_0 and than iteratively applying eq. (4) to it. Once evaluated, the policy is improved, that is π_k is greedy respectively to $Q^{\pi_{k-1}}$:

$$\pi_k(s) = \operatorname*{argmax}_{a \in A} Q^{\pi_{k-1}}(s, a).$$

Evaluation and improvement steps are iterated until convergence of π_k to π^* (which can be demonstrated to happen in a finite number of iterations when $\pi_k = \pi_{k-1}$). In practice, the learning is stopped when two consecutive policies are identical which happens when $\pi_k = \pi^*$ since the policy which is greedy according to its own Q-function is the optimal policy by definition (eq. (3)).

The second algorithm is the *value iteration* algorithm. It aims at estimating directly the optimal state-action value function Q^* which is the solution of eq. (5). The T^* operator is not linear (because of the max operator), therefore computing Q^* via standard system-solving methods is not possible. However, it can be shown that T^* is also a contraction [12]. Therefore, according to Banach fixed-point theorem, Q^* can be estimated using the following iterative way:

$$\hat{Q}^*_i = T^* \hat{Q}^*_{i-1}, \quad \forall \hat{Q}^*_0 \lim_{i \to \infty} \hat{Q}^*_i = Q^* \quad (6)$$

However, the convergence takes an infinite number of iterations. Practically speaking, iterations are stopped when some criterion is met, classically a small difference between two iterations: $\|\hat{Q}^*_i - \hat{Q}^*_{i-1}\| < \xi$. The estimated optimal policy (which is what we are ultimately interested in) is greedy respectively to the estimated optimal Q-function: $\hat{\pi}^*(s) = \operatorname{argmax}_{a \in A} \hat{Q}^*(s, a)$.

C. Approximate DP and Reinforcement learning

Dynamic programming makes two very strong assumptions. First it assumes that the dynamic of the system is known and therefore that transition probabilities and the reward function are available. Second, an exact representation of the Q-function is supposed to be computable. Both assumptions are rarely met. Instead of having access to the dynamics of the system two options are often possible: either a database of interaction examples is available, either the system itself can be tested through a trial-and-error process. In the former case, the class of algorithms aiming at estimating an optimal policy from a fixed data set is named *Approximate Dynamic Programming* (ADP). It forms a set of *batch* algorithms (off-line) to solve the optimal control problem at sight. In the later case, the learning agent has to improve its strategy *online* while interacting with the system. This is the general *Reinforcement Learning* problem. In any case, if the state-action space is too large to be enumerated so as to have an exact (tabular) representation of the Q-function $Q(s, a)$, then it is common to learn a parametric approximation $Q_\theta(s, a)$ where θ is a set of parameters to learn. The parameterization can be of any form. For example, if the Q-function is approximated by a neural network like in [13], the set of parameters θ contains the synaptic weights of the network. However, a common choice is the linear parameterization like in [14]:

$$Q_\theta(s, a) = \sum_{i=1}^{p} \phi_i \theta_i = \phi^T \theta \qquad (7)$$

Most renown ADP algorithms are Least Square Temporal Differences (LSTD) [14], Least Square Policy Iteration (LSPI) [15] and Fitted Value Iteration (FVI) [16]. LSTD is an efficient batch algorithm for policy evaluation. LSPI uses LSTD in a policy iteration framework to learn an optimal policy from a database of interactions. FVI is more related to the value iteration algorithm and aims at directly discovering the optimal Q-function from a dataset.

There are a lot of online reinforcement learning algorithms in the the literature. They are mostly based on the computation of a temporal difference which can be seen as the difference between the right and left members of one of the Bellman equations for one transition. For instance, in the case of the Bellman evaluation equation, the temporal difference δ_i^E for a given transition $\{s_i, a_i, r_i, s_{i+1}, a_{i+1}\}$ is given by:

$$\delta_i^E = r_i + \gamma \hat{Q}_{\theta_{i-1}}(s_{i+1}, a_{i+1}) - \hat{Q}_{\theta_{i-1}}(s_i, a_i)$$

where θ_i is the evaluation of the Q-function parameter set at time i. The temporal difference δ_i^O for the Bellman optimality equation is:

$$\delta_i^O = r_i + \gamma \max_{b \in A} \hat{Q}_{\theta_{i-1}}(s_{i+1}, b) - \hat{Q}_{\theta_{i-1}}(s_i, a_i)$$

Temporal differences algorithms use this quantity δ_i to update the parameters estimation after each interaction using a Widrow-Hoff-like equation:

$$\theta_i = \theta_{i-1} + K_i \delta_i$$

where K_i is a gain. Algorithms differ by the temporal difference considered and the gain computed. For instance, the SARSA algorithm [6] uses the δ_i^E temporal difference and the gain K_i is a simple learning rate. It is an online version of policy iteration. The Q-learning algorithm [17] uses the δ_i^O temporal difference also with a simple learning rate and is an online version of the value iteration algorithm. Other methods exist like the Kalman Temporal Differences framework (KTD) [18] where the gain is computed with a Kalman filter and either temporal difference can be used, Gaussian Processes Temporal Differences (GPTD) [19] where the gain is computed with Gaussian Processes and where only the evaluation temporal difference can be used *etc.*

D. On-policy vs Off-policy

According to what has been said in subsections II-B and II-C, one can distinguish two different categories of algorithms: those based on the Bellman evaluation algorithm and those based on the Bellman optimality equation. The former algorithms are called *on-policy* algorithms because they evaluate the currently used policy and then improve it to finally obtain an optimal policy. The later are called *off-policy* algorithms since they aim at learning the Q-function of the optimal policy from observations of an other policy. These algorithms are generally used when the system cannot be controlled during learning. It is the case when learning from a fixed set of data (the agent cannot generate new transitions than those contained in the dataset) or when there are constrains on the strategy that are admissible by the system to be controlled. It is the case in our application since there is a human driver in the loop and the learning decision maker cannot try any random sequence of action for example.

E. Model-based vs Model-free

The algorithms mentioned in subsections II-B and II-C are called *model-free* algorithms because they don't need a model of the dynamics of the system. That means that the transition probabilities and the reward function (the model of the MDP) are not known neither computed by the algorithm. It is also possible to evaluate these quantities either from a database or from online interactions. This gives rise to *model-based* algorithms. If the state-action space is small enough to be enumerated, than the these quantities are discrete values that can be evaluated by a simple counting (maximum likelihood estimation). The DYNA-Q algorithm [6] proceeds this way. If not, these quantities have to be estimated as parametric functions like in the case of the Q-function in the previous section [20]. For example, the transition probabilities can be approximated as having a Gaussian distribution and the parameters (μ, σ) of each transition probability can be estimated through interactions. A comparison between both types of method can be found in [21]

III. PADAS as an MDP

To cast the PADAS optimization problem described in Section I-B into the Markov Decision Process (MDP) framework [7], one has to define state and action spaces and a reward function. The optimal policy will be the one that maximizes the expected discounted reward over the long run. Let's remind the problem we want to solve. The collision avoidance problem described in the introduction consists of devising a method of using the system's resources (capability to send warning signals to the driver and to decelerate the vehicle) such that the probability of a collision is minimized while introducing the minimum of disturbance for the driver.

We will consider that at any time t, a collision of the host vehicle with the leading vehicle occurs if the *time to collision* at time t drops below a threshold ϵ. Time to collision θ at time t is defined as

$$\theta_t = \frac{d_t}{v_t - v'_t}$$

where d_t is the distance to the next obstacle, v_t is the velocity of the car and v'_t is the velocity of the obstacle. These variables can be measured with equipment available on cars with PADAS installed.

The state is thus made of a single continuous real value which is the estimated time to colision θ_t. The elementary actions the PADAS can take are of two kind : (i) apply braking pressure equal to b times the maximum braking pressure of the vehicle ($b \in [0, 1]$), (ii) send a warning signal to the driver (warnings can be of audio,visual or haptic type). An actual PADAS action will be any combination of warnings and braking pressure application. The reward function is defined according to three different costs : a cost for the time to collision (the higher the time to collision the lower the cost), a cost for braking (again, the larger the braking, the larger is the cost) and finally a cost for sending signal (with each signal is associated a level of urgency; the more urgent the signal, the larger is the cost). Of course, a very high cost is associated to a time to collision of 0 (which means collision). Intempestive breaking and warnings are also judged to have a negative impact on the driving experience and negative rewards are associated.

IV. Experiments and Results

The experiments were divided into three phases: data collection, optimal strategy learning and testing. The learning is done in a batch manner, that is from collected data, and not online (during the driving). This choice is made so as not to disturb the drivers by changing the PADAS intervention strategy while they are driving the car. Indeed, these changes in the warning and braking strategy, meaning that into a similar situation different PADAS actions could occur, could induce unusual behaviors and stress of the drivers which will not be encountered in real scenarios.

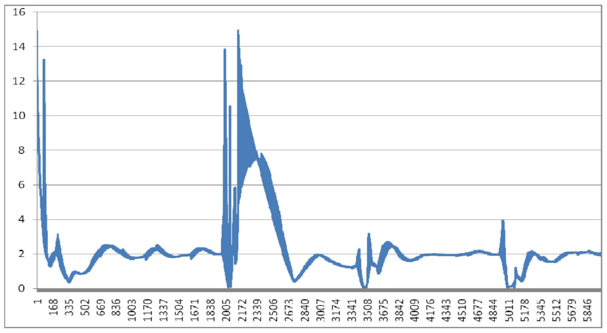

Fig. 2. Episode example : time to collision vs time

A. Data Collection

A driving experiment has been conducted on a ScanerII[3] car simulator, a fixed based system that comprises a mock-up of a car with real driving controls (*i.e.* seat, steering wheel, pedals, gear, handbrake), a digital simulated dashboard displaying a traditional instrumental panel and a frontal projection screen where the simulated environment is displayed to the driver. This type of simulator can output data at a frequency of 20 Hz (= 0.05 s). Five subjects participated to the experiments, driving on an extra-urban and motorways scenarios for more than 1 hour each, using the ACC+ mode of the PADAS. The PADAS used an initial strategy π_0 which was based on handcrafted rules (the time to collision was discretized and to each bin was associated an action) and stochastic noise (that is random actions could sometimes be selected) was added to artificially explore the state-action space.

Trials of about 600 minutes involving different human drivers driving the simulator were conducted. The simulation trials data is organized in the form of a set of *episodes*. Each episode is sequence of state-action pairs of the MDP model of the collision avoidance problem described in Section III. Values of θ could be computed every 300ms which provided a large amount of transition to learn. An example of particular episode is provided on Fig. 2

B. Optimal Strategy Learning

As explained in Section II, several approaches can be envisioned to solve the optimal control problem at sight. Particularly, model-free or model-based algorithms could be investigated. In this paper is reported the result of the learning with a model-based algorithm (see Section II-E), that is a model that learns the transition probabilities of the {vehicle+driver} system. The {vehicle+driver} system is indeed a stochastic system since there are inter- and intra-driver variabilities inducing different reactions to similar situations given that the driver is different or even given that the perception of the situation is different for a

[3]www.scaner2.com

same driver. Moreover, since we learn from a fixed set of data, an *off policy* algorithm (see Section II-D) is chosen.

A state in this MDP is a time to collision. The minimum value of the time to collision is 0 seconds and the maximum value can be considered to be 50 seconds (as far as collision avoidance is concerned, any value above 50 seconds can be considered equivalent to 50 seconds). So, a state in the MDP is a number in the interval [0, 50]. This is an infinite set. A model-based approach would require to learn transition probabilities between states. Either we learn parameters of continuous distributions, either we discretized the state space. In order to render it finite, we exhaust the interval into disjoint partitions of unequal sizes. As an example: [0, 0.5[, [0.5, 1[, [1, 1.5[, [1.5, 2[, [2, 3[, [3, 5[, [5, 7[, [7, 10[, [10, 15[, [15, 50]. These partitions (or bins) represent the states of a (finite) MDP. The portioning just described gives 10 states and was driven by expert knowledge. So, if the time to collision is say 0.73 seconds, the MDP is in the 2nd state, if it is 2.15 seconds, the MDP is in 5th state and so on.

The probabilities $P(.|s,a)$ of stepping from one state (as defined hereabove) to another given that an action has been taken are computed for every tuple (s, a, s') present in the database. The associated costs are then computed (a cost is regarded as a negative reward) and the value iteration algorithm (see Section II-B) is applied to the learnt MDP.

C. Results

Two optimal strategies were constructed based on two different partitions of the set of possible time to collisions. We refer to the two policies as π_1^* and π_2^*. For the sake of demonstration, one of both learnt strategies (π_1^*) is explained in Table I:

TABLE I
Learnt policy π_1^*

Time to Collision Interval	**Action to Take**
[0, 0.5[	send emergency signal, apply maximum brake
[0.5, 0.5[	send danger signal, apply 80% brake
[1, 2[	send danger signal, apply 40% brake
[2, 3[	send danger signal, apply 20% brake
[3, 10[	send collision warning signal, don't apply any brake
[10, 50]	Send normal signal, don't apply any brake

In the testing phase, π_1^*, π_2^* and the hand-coded strategy π_0 that served to generate the database (but without the stochastic action selection) were tested by using each of them in turn as the PADAS strategy in simulation trials involving different human drivers. Note that the hand-coded policy was based just on common sense. To assess the quality of the learnt strategies, two variables were monitored: the time to collision and the distance between the vehicles. A collision was said to have occurred if the time to collision dropped to less than 1.5 seconds or if the distance dropped to less than 1 meter. Table II gives the results of the learnt policies $\pi_{1,2}^*$ and the handcrafted policy π_0 (this time without random actions). It is important to notice that the policies $\pi_{1,2}^*$ have been learnt from samples generated with a policy π_0 which was acceptable for drivers (off-policy learning with an acceptable behavioral policy). Indeed, a completely random policy could theoretically provide better results since the whole state-action space could be sampled uniformly, yet it would have impacted the drivers' behavior. Table II lists the percentage of samples in which a collision occurred according to the two definitions given above. These results show that the strategies derived using the MDP approach render the driving experience safer by reducing the number of collisions and near collisions.

TABLE II
Learnt policy performance ($\pi_{1,2}^*$) vs handcrafted policy (π_0)

	Time to Collision	Distance
π_1^*	2.1%	1.5%
π_2^*	1.5%	2.4%
π_0	3.8%	4.3%

V. Conclusions and perspectives

In this paper, an approach for conceiving an optimal warning and intervention strategy for a PADAS is presented. The problem is modelled as an unknown Markov Decision Process, that is the transition probabilities are unknown because it depends on the drivers reactions to driving situation and PADAS interventions. Data collected from simulation trials with real human drivers were used to learn the unknown parts of this MDP. The optimal control problem is solved using the value iteration algorithm to obtain an optimal strategy. Since the approach consists in learning the model of the MDP, it can be regarded as a sort of model-based reinforcement learning.

Obtained results indicate that there exists a real benefit in adopting this approach: the percentage of collisions or near collisions drops down by a non-negligible amount while the optimized cost function ensure not to disturb the drivers with unnecessary warning signals. The other advantage of our approach is that the optimal strategy is constructed directly from the data. No hypothesis is thus made about driver's behavior. Moreover, the learning phase doesn't introduce bias effects on the driver's behavior neither. Finally, it shows that learning can be done from recorded information which can actually be obtained on real cars, which means that the learning could be improved and even personalized for each driver.

This work opens up interesting possibilities for conceiving intelligent systems for vehicles but also to model drivers. Indeed, the MDP approach provides the right framework to allow, as a by-product, to construct a driver behaviour model embodied in the state transition probabilities of the MDP. It could therefore be possible to simulate drivers or to infer some information about the user's perception of the situation.

In addition, the entire range of algorithms for solving MDPs, including those from the domain of reinforcement learning such as LSPI [15] or FVI [16] (briefly described in Section II-C) can be used to constructing sequential decision making strategies for intelligent systems that are based entirely on observed data, and not on complex hypothesis. This would particularly be interesting since they are able to deal with continuous state-action spaces which is the case in the problem described here. Indeed, the partitioning of the state and action space described in Section IV-B introduces an aliasing problem that may lead to suboptimal strategies. The use of algorithms dealing with continuous state-action spaces could actually implicitly learn the optimal partitioning and thus produce more accurate policies.

Next steps involve the integration of this MDP model with the driver's distraction classifier, in order to improve the efficacy of the PADAS, also depending on the driver's states. In addition, we would like to integrate MDP model of PADAS with a model for driver's manoeuvres classification and prediction, based on Hidden Markov Models. In fact, the possibility to "understand" driver's intention can make the PADAS able to intervene and to support drivers in a more acceptable and appropriated way, anticipating wrong behaviors even before they can be a serious danger for drivers themselves and for the other road actors. In addition, we think to extend MDP approaches to other road traffic actors, so including not only the system "driver + vehicle", but also some specific categories of vulnerable road users, such as pedestrians, cyclists, *etc.*

Acknowledgment

The research leading to these results has received funding from the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement no FP7 - 218552, Project ISi-PADAS (Integrated Human Modelling and Simulation to support Human Error Risk Analysis of Partially Autonomous Driver Assistance Systems). The authors would like to specially thank the ISi-PADAS consortium that has supported the development of this research

References

[1] E. Muhrer and M. Vollrath, "Results from accident analysis," ISI-PADAS project deliverable, Tech. Rep. Task 1.22, 2009.

[2] M. Vollrath et al., "Ableitung von anforderungen an ein fahrerassistenzsystem aus sicht der verkehrssicherheit," Bundesanstalt für Straßenwesen, Fahrzeugtechnik, Tech. Rep. F 60, 2006.

[3] S. Briest and M. Vollrath, "In welchen situationen machen fahrer welche fehler? ableitung von anforderungen an fahrerassistenzsysteme durch," in *In-Depth- Unfallanalysen. In VDI (Ed.), Integrierte Sicherheit und Fahrerassistenzsysteme*, 2006, pp. 449 – 463.

[4] R. J. Kiefer, J. Salinger, and J. J. Ference, "Status of nhtsa's rear-end crash prevention research program," National Highway Traffic and Safety Administration, Tech. Rep. 05-0282, 2005.

[5] J. D. Lee, D. V. McGehee, T. L. Brown, and M. L. Reyes, "Collision Warning Timing, Driver Distraction, and Driver Response to Imminent Rear-End Collisions in a High-Fidelity Driving Simulator," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 44, no. 2, pp. 314–334, Summer 2002.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, 3rd ed. The MIT Press, March 1998.

[7] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics, vol. 6, pp. 679–684*, 1957.

[8] K.-D. Kuhnert and M. Krödel, "Autonomous vehicle steering based on evaluative feedback by reinforcement learning," in *Machine Learning and Data Mining in Pattern Recognition*, ser. Lecture Notes in Computer Science, P. Perner and A. Imiya, Eds. Springer Berlin / Heidelberg, 2005, vol. 3587, pp. 405–414.

[9] T. Martinez-Marin, "A reinforcement learning algorithm for optimal motion of car-like vehicles," in *Proceedings of the 7th International IEEE Conference on Intelligent Transportation Systems*, oct. 2004, pp. 47 – 51.

[10] S.-Y. Oh, J.-H. Lee, and D.-H. Choi, "A new reinforcement learning vehicle control architecture for vision-based road following," *IEEE Transactions on Vehicular Technology*, vol. 49, no. 3, pp. 997 –1005, may. 2000.

[11] R. Bellman, *Dynamic Programming*, 6th ed. Dover Publications, 1957.

[12] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, April 1994.

[13] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, March 1995.

[14] S. J. Bradtke and A. G. Barto, "Linear Least-Squares algorithms for temporal difference learning," *Machine Learning*, vol. 22, no. 1-3, pp. 33–57, 1996.

[15] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[16] G. Gordon, "Stable Function Approximation in Dynamic Programming," in *Proceedings of the International Conference on Machine Learning (IMCL 95)*, 1995.

[17] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambidge, 1989.

[18] M. Geist and O. Pietquin, "Kalman Temporal Differences," *Journal of Artificial Intelligence Research (JAIR)*, 2010.

[19] Y. Engel, S. Mannor, and R. Meir, "Reinforcement Learning with Gaussian Processes," in *Proceedings of the International Conference on Machine Learning (ICML 05)*, 2005.

[20] A. L. Strehl and M. L. Littman, "An Analysis of Model-Based Interval Estimation for Markov Decision Processes," *Journal of Computer and System Sciences*, 2006.

[21] C. Atkeson and J. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proceedings of the IEEE International Conference onRobotics and Automation*, vol. 4, Apr. 1997, pp. 3557 –3564 vol.4.