Probabilistic logic description of Markov decision processes for modeling driving behaviors in self-driving cars

Hector Avilés, Marco Negrete, Rubén Machucho, Karelly Rivera, David Trejo y Héctor Vargas

***Abstract*— The selection of driving behaviors is a central component of autonomous cars. Although rules-based strategies and probability models are among the most successful techniques for selecting such behaviors, there is still the need to combine the flexibility and conceptual clarity of rule-based representations with the uncertainty handling provided by probabilistic models. Therefore, we propose to generate action policies from the probabilistic logical description of a Markov decision process (MDP) for an self-driving car to avoid collisions with other vehicles in its environment. We consider three behaviours, that is *keep distance*, *change lane*, and *cruise*. As state variables, we consider the relative spatial position of the other vehicles. We tested our proposal in simulation in different scenarios where the vehicle has to decide whether to change lane or keep distance. Results show the correct execution of the overall system and the appropriateness of the knowledge representation and decision strategy to choose the pertinent action that prevents potential collisions for the self-driving car.**

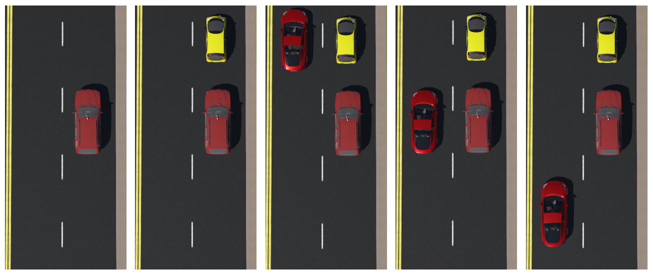

Fig. 1: Example of the modeled environment. Left: Vehicle is moving at a cruise speed. Second left: It detects another car and pass it. Three rightmost: Vehicle detects a car in front but should not pass and just keep distance.

I. Introduction

Self-driving cars have received significant attention in the last decades due to the potential positive effects for the future of mobility. The components of an self-driving can roughly be classified as perceptual and decision making subsystems. One of the most important elements of decision making is the behavior selection module that decides what reactive or short-term action (e.g., braking, accelerating, or stopping) is more convenient as a response to the current environment [1].

Neural networks, rule-based models and probabilistic models are among the most common representations for behavior selection. On the one hand, neural networks has a outstanding performance in both, learning from numerical examples and for the classification of noisy or partial observations. Unfortunately, the internal codification of knowledge in neural networks is not human-readable. Unfortunately, this condition hinders the inspection of the decision-making process, that it is particularly relevant for self-driving cars. On the other hand, rule-based models provides understandable descriptions of the domain, usually in the form of predicate logic rules and facts and facilitates the explanation of decisions. Moreover, probability theory is the most widely used scheme for handling uncertainty and despite the propositional nature of the random variables, it relaxes the binary truth assumptions proper of first-order logic.

Therefore, in this document, we propose an action policy obtained from a probabilistic logic representation of a Markov decision process (MDP) as a behavior selection strategy for a self-driving car to avoid accidents with other moving vehicles. MDPs are encoded in MDP-ProbLog [2] that is based on ProbLog tools [4]. This approach allows us: i) to express complex relationships between the self-driving car and other cars on the road using fairly simple probabilistic logic rules and facts, ii) to model uncertainty involved in the result of an action of the self-driving car, and iii) to assign negative utilities for different potential crash situations and positive rewards for desirable actions and states. Our setting considers the self-driving car and four vehicles traveling on a one-way street with two lanes, and three simple actions: *change lane*, *keep distance* and *cruise*. Every time the self-driving starts its movement in the right lane. The perceptual subsystem indicates to the car if there is a vehicle in front of it in the right lane, a car ahead in the left lane, or a car aside to the left, or a car behind it in the left lane. Then, the self-driving car has to decide which action is convenient given the presence or absence of the other vehicles. Figure 1 depicts five of $2^4 = 16$ possible arrangements and the action taken on each one. We tested our proposal in the Webots simulator [12] extended with a Simulation of Urban Mobility (SUMO) interface [7]. We carried out 20 repetitions of the action selection task on each configuration. Our experimental results shows a classification accuracy of 90% when selecting actions, and also the proper execution of the overall system. The main contribution of this paper is twofold. First, in the best of our knowledge, this is the first time probabilistic logic techniques are used to model behavior selection in an self-driving vehicle. Second, it is shown the appropriateness of probabilistic logic representation to describe driving behaviors and to generate an action policy that prevents potential collisions.

*This work was supported by UNAM-DGAPA under grant IA106520

II. Related work

In related work in the area of autonomous vehicles, researchers extract information from the vehicle's environment, using distance sensors, cameras, radars, gps, etc. They also store in various frameworks, the knowledge of various risk situations of vehicle collision, or the recognition of the surrounding objects of the vehicle. Among the most common frameworks are fuzzy sets, neural networks, finite state machines (FSM), etc. In the decision making stage, they use FSM, prolog rules, fuzzy inference, etc. Three of the most important works in this field are briefly described below.

In [3] Chen and et al. present a hierarchical controller for autonomous vehicle management, this controller is divided into two levels: high and low. The high level uses readings from vehicle sensors such as camera, gps, digital map, radar, etc. and has a finite state machine (FSM), with which it identifies the relative position of surrounding cars, it also predicts a possible collision. This information is passed to another (FSM), which determines the behavior of the driver to avoid possible collisions with other vehicles. At the low level, they develop a pair of controllers: the first one for the lateral movement that the vehicle will develop and the second one for the longitudinal movement; guaranteeing with this the stability of the vehicle, during its advance and lane change.

In [6] Lu Huang et al. construct a 9-region grid, where the autonomous vehicle is placed in the center, surrounded by 8 possible neighboring vehicles or empty places. They also consider 3 classes, the obstacles, the road network and the driving scenario. In the first class, there are the static obstacles, such as trees, buildings, cones on the road, and also the dynamic obstacles, such as cyclists, pedestrians and animals. In the second class, highways, urban areas and parts of them, such as traffic lights, lane markings, etc., are considered. In the last class, scenarios include passing through a tunnel, a bridge or a road intersection. With this information they develop a prolog program for decision making, and perform the autonomous driving of the vehicle.

In [9] A. Rizaldi and M. Althoff investigate the case of a possible collision between two autonomous vehicles, they consider that if an autonomous vehicle complies with the traffic rules established by international conventions, then it would be exempted from crash liability. To do this, they construct a set of high-order logic rules and use Isabelle/HOL software to test their results.

III. Theoretical background

This section gives a brief overview of ProbLog, MDPs and MDP-ProbLog. This is not intended to be an in-depth discussion of each one of these models, but rather to present some general ideas behind the representation and solving of MDPs using probabilistic logic.

A. ProbLog overview

ProbLog is both, a declarative language and a set of tools for integrating independent ground probabilistic facts into logic programs. It extends the syntax of Prolog. ProbLog support inductive definitions, cyclic or recursive rules, transitive relations, multiple non-mutually exclusive bodies with the same head, and arbitrary atoms as evidence. ProbLog follows the distributed semantics proposed by T. Sato [10] for calculating the probability of a query given the probabilistic logic program. It also adopts the well-founded semantics [11] for determining whether a query is logically entailed by a ground logic program. In addition to the construction elements of Prolog, ProbLog incorporates probabilistic facts (atomic ground formulas that can be understood as binary true/false random variables), probabilistic rules and annotated disjunctions. The latter emulate probabilistic discrete multi-valued random variables in which at most one value occurs at a time. These additional components are measured with probability of occurrence assigned to them.

B. Markov decision processes

Markov decision processes [8] is a standard model for sequential decision-making. An MDP is composed by: a) a finite set of states that are completely observable (i.e., the current state is known), b) a finite set of actions, c) a transition function that models the uncertainty of reaching a future state given an action executed in the current state, and d) a reward function for each pair state-action. This function is used to compute the numerical value of a state, that corresponds to the immediate reward for the state-action pair, plus the average of the cumulative future rewards from each new possible state following a fixed policy.

The solution of the MDP is an action policy (i.e., an state-action pair for each state of the system) that satisfies the Bellman's equation. The idea is to maximize the future cumulative reward for each state of the system. Here, we consider infinite-horizon MDPs and for them, a discount factor $[0, 1)$ must be used. The optimal policy can be obtained by algorithms such as value iteration (iterative maximization of the value of each state over all possible actions until convergence) and policy iteration (the iterative improvement of policies from an arbitrary initial policy). A typical real problem could involve several binary state variables whose values defines each state of the MDP. As the number of state variables increases, it is more convenient to use a factored representation of the transition function [5]. This has the advantage of simplify transition model by focusing on the effect of an action over a subset of the state variables.

C. MDP-ProbLog

MDP-ProbLog is a specialization of ProbLog for modeling infinite horizon factored MDPs that uses the algorithm value iteration for solving the MDP. An MDP-ProbLog program is a valid ProbLog program that has the following primitives: 1) a finite set of atomic formulas divided into a finite set of state variables, a finite set action predicates, and a finite set of utility predicates, 2) a finite set of auxiliary probabilistic facts, 3) a finite set of transition rules, 4) and a finite set of reward rules. Special atoms are used to declare these elements, e.g., state variables are identified with the atom 'state_fluent', actions are denoted with the atom 'action' and utilities with the atom 'utility'. state_fluent and action are unary predicates that receives as parameter the identifier of a state variable and action, respectively. utility is a binary predicate. For them it is assigned the action or state for which a reward is given and its corresponding positive or negative reward.

IV. Methodology

A. Design of the MDP

As stated above, we use four binary state variables that we call *free_North*, *free_NorthWest*, *free_West* and *free_SouthWest*. These true/false state variables indicate whether there is a vehicle in the respective relative position. The declaration for them is simple, for example, state_fluent(free_North) is the definition of the variable free_North. The definition of the actions is straightforward too, e.g., action(keep_distance).

The encoding of the transition model require to define which state variables modifies its truth value when an action is performed. For example, for the action change_lane we have:

0.9::free_North(1) :- free_NorthWest(0), free_West(0), change_lane, not(higher_vel_SouthWest(0)).
0.05::free_North(1) :- not(free_NorthWest(0)), change_lane.
0.05::free_North(1) :- not(free_West(0)), change_lane.

Accordingly to first rule, free_North will be true in the next epoch with a probability of 0.9 if free_NorthWest and free_West are true (there is enough space to change lane) and the action change_lane is performed. Similarly, the latter two rules indicate that there is a low probability of 0.5 that free_North is true if there is not free space to the northwest and west. Initially, we constructed the set of rules for the probability distribution of the factorized transition function for each action and state variable. This model resulted in several ProbLog rules that later were considerable reduced via the unification algorithm, by observing that for many rules the same probability was assigned and that they were logically entailed by other rules.

Finally, reward model is defined as:

utility(free_North(1), 5).
utility(rearEnd_crash(1), -30).
utility(sideSwipe_crash(1), -10).
utility(keep_distance, -10).
utility(change_lane, -1).

0.99::rearEnd_crash(1) :- not(free_North(1)), cruise, not(keep_distance).
0.95::rearEnd_crash(1) :- not(free_NorthWest(1)), change_lane.
0.95::sideSwipe_crash(1) :- not(free_West(1)), change_lane.

In this case, we assign a positive reward whenever the self-driving car has free space in front of it, and different negative rewards for rearEnd_crash and sideSwipe_crash predicates. These negatives rewards reflects the difference on the severity of the consequences of these types of crashes. The reward is also negative for change_lane, as we consider that involves a potential risk of an accident.

B. The environment

The situation we tested consist on a vehicle moving at a cruise speed that detects an obstacle-car in front of it. The self-driving car has to decide whether to pass the obstacle car or to brake and keep distance. Decision is made based on the existence of more obstacle-cars in the side lane. This situations were modeled using the Webots simulator [12]. Figure 1 shows examples of the modeled situations.

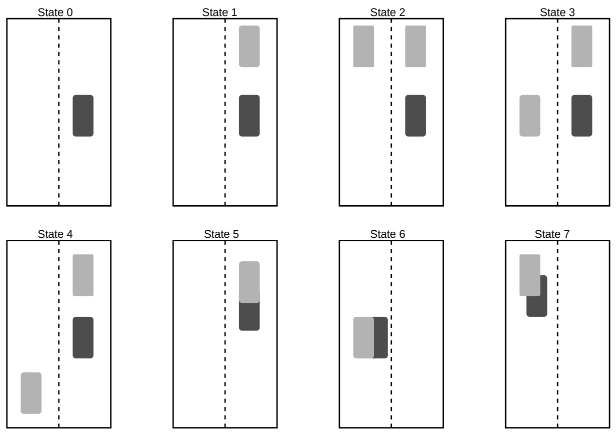

Fig. 2: MDP States to model the environment.

C. Decision system

We modeled the environment using a Markov Decision Process programmed with ProbLog. Figure 2 shows the states used to model the environment.

D. The test bed

To simulate the environment we used the Webots simulator and the SUMO library. The simulated car is equipped with an RGB camera, used for lane segmentation and tracking, and a 3D-Lidar, used to detect and estimate position and speed of other vehicles. The simulated car allows to set linear speed v and steering δ as control inputs.

1) Lane segmentation and tracking: The segmentation process is based on Canny edge detection and Hough Transform for finding lines. Since almost half of the image corresponds to the landscape above the horizon, and considering that due to perspective, lanes form a triangle in front of the car, we define a region of interest where lanes are most likely to be found, reducing the computation time. General process for lane segmentation can be summarized in the following steps:

1) Set a triangular Region of Interest where lanes are most likely to be found
2) Get the set of edges using a Canny edge detector
3) Use the Hough-Transform to find lines in the edges

Lane tracking is performed using a proportional control that combines distance to the lane border and angle with respect to the lane border. Consider figure 3. Green lines represent the desired lines we should observe if the car is well centered in the lane. Cyan lines represent the lines actually observed. Expressing lines in the normal form, (ρ_{ld}, θ_{ld}) and (ρ_{rd}, θ_{rd}) represent the desired line parameters for left and right lane borders respectively, and (ρ_l, θ_l) and (ρ_r, θ_r) represent the actual observed borders. We use the following control laws to track lane:

$$v = C_b \tag{1}$$
$$\delta = K_\rho e_\rho + K_\theta e_\theta \tag{2}$$

with e_ρ and e_θ calculated as:

$$e_\rho = ((\rho_{ld} - \rho_l + \rho_{rd} - \rho_r)/2) \tag{3}$$
$$e_\theta = ((\theta_{ld} - \theta_l + \theta_{rd} - \theta_r)/2) \tag{4}$$
$$\tag{5}$$

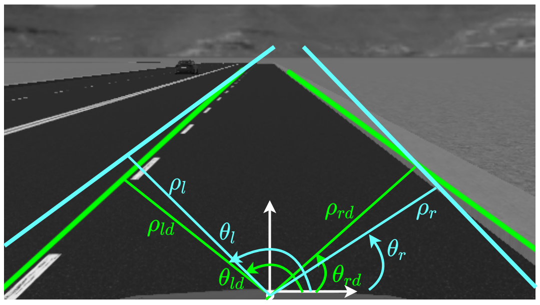

Fig. 3: Variables for lane tracking

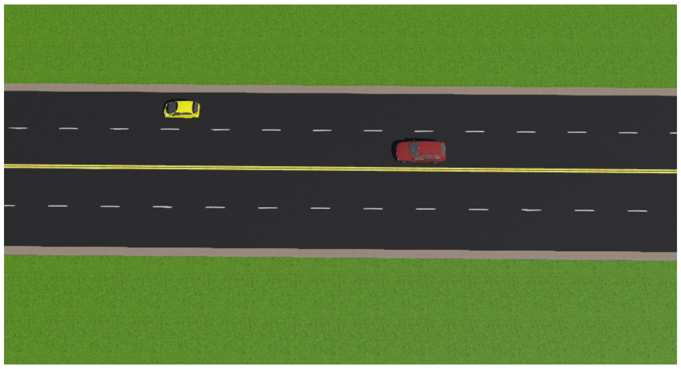

Fig. 4: Caption

Steering is calculated based on the error between desired and observed lines. Linear speed is set as a constant selected based on the current behavior: steady motion, keep distance and passing.

2) Vehicles speed estimation: As mentioned before, the situation we tested consist on a decision system to choose whether to pass another car or not. Thus, we need to estimate other cars' position and velocity. To achieve this, we used the 3D Lidar sensor mounted on the top of the simulated vehicle. Vehicle position and speed estimation can be summarized in the following steps:

- Point cloud filtering by distance and height
- Clustering by K-means
- Velocity estimation by Kalman Filters

As a first step, we eliminate the points in the point-cloud that are part of the ground. Since the 3D Lidar detects its own car, we also filter points with coordinates inside a bounding box representing the car. We cluster the filtered cloud using K-means and each centroid is taken as a position measurement to estimate speed with a Kalman Filter.

V. Experiments and Results

Table I represents the policy obtained with the Markov Decision Process previously described. The symbols N, NW, W and SW (North, North-West, West and South-West) represent the regions around the car which can be free or occupied. In Table I, 0 means there is a car in that regions and 1, it is free. Depending of what is detected using the lidar sensor, we choose one of three behaviors: Steady-Motion, Keep-Distance (with the car in front) or Take-Over.
Agregar tabla con la política. Describir implementación en una

N	NW	W	SW	Action	N	NW	W	SW	Action
0	0	0	0	Keep dist	0	0	0	1	Keep dist
1	0	0	0	Keep dist	1	0	0	1	Keep dist
0	1	0	0	Keep dist	0	1	0	1	Keep dist
1	1	0	0	Steady	1	1	0	1	Keep dist
0	0	1	0	Keep dist	0	0	1	1	Keep dist
1	0	1	0	Keep dist	1	0	1	1	Keep dist
0	1	1	0	Keep dist	0	1	1	1	Keep dist
1	1	1	0	Keep dist	1	1	1	1	Keep dist

TABLE I: Policy obtained with the MDP.

máquina de estados. Mostrar imágenes describiendo la situación y la decisión que tomó el vehículo.

VI. Conclusions and Future Work

In this paper we have presented a probabilistic logic method to generate the action policies of an autonomous vehicle in three behaviors: lane change, keep moving and keep the distance to other vehicles. We tested the method in a simulated environment using Webots and MDP-ProbLog. The test environment consisted of two lanes, with five possible vehicles surrounding the autonomous vehicle. The results obtained demonstrate that the autonomous vehicle can be driven safely and collision-free by making use of the generated policies.
As future work, it will be tested in other environments, with more lanes and other possible collision risk situations.

References

[1] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. M. Paixao, F. Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, 165:113816, 2021.

[2] T. P. Bueno, D. D. Mauá, L. N. De Barros, and F. G. Cozman. Markov decision processes specified by probabilistic logic programming: representation and solution. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 337–342. IEEE, 2016.

[3] K. Chen, B. Yang, X. Pei, and X. Guo. Hierarchical control strategy towards safenbsp;driving of autonomous vehicles. *Journal of Intelligent amp; Fuzzy Systems*, 34(4):2197–2212, 2018.

[4] D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

[5] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. *arXiv preprint arXiv:1301.6704*, 2013.

[6] L. Huang, H. Liang, B. Yu, B. Li, and H. Zhu. Ontology-based driving scene modeling, situation assessment and decision making for autonomous vehicles. In *2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, pages 57–62. IEEE, 2019.

[7] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner. Microscopic traffic simulation using sumo. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, 2018.

[8] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[9] A. Rizaldi and M. Althoff. Formalising traffic rules for accountability of autonomous vehicles. In *2015 IEEE 18th international conference on intelligent transportation systems*, pages 1658–1665. IEEE, 2015.

[10] T. Sato and Y. Kameya. PRISM: a language for symbolic-statistical modeling. In *IJCAI*, volume 97, pages 1330–1339. Citeseer, 1997.

[11] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):619–649, 1991.

[12] Webots. http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software.