Probabilistic Logic Markov Decision Processes for Modeling Driving Behaviors in Self-driving Cars

Héctor Avilés[1], Marco Negrete[2], Rubén Machucho[1(✉)], Karelly Rivera[1], David Trejo[2], and Héctor Vargas[3]

[1] Polytechnic University of Victoria, 87138 Cd. Victoria, Tamaulipas, Mexico
{havilesa,rmachuchoc,1930435}@upv.edu.mx
[2] Faculty of Engineering, National Autonomous University of Mexico, 04510 Mexico City, Mexico
{mnegretev,davidtrejofs}@comunidad.unam.mx
[3] Popular Autonomous University of Puebla, 72410 Puebla, Mexico
hectorsimon.vargas@upaep.mx

Abstract. Rule-based strategies and probability models are among the most successful techniques for selecting driving behaviors of self-driving cars. However, there is still the need to explore the combination of the flexibility and conceptual clarity of deterministic rules with probabilistic models to describe the uncertainty in the spacial relationships among the entities on the road. Therefore, in this paper we propose an action policy obtained from a probabilistic logical description of a Markov decision process (MDP) as a behavior selection scheme for a self-driving car to avoid collisions with other vehicles. We consider three behaviours: *keep distance*, *overtaking*, and *steady motion*. The state variables of the MDP signal the presence or absence of other vehicles in the surroundings of the ego car. Simple probabilistic logic rules characterize the probability distribution of the immediate future state of the autonomous car given the current state and action. The utility model of the MDP rewards the autonomous car when no car is ahead and it penalizes two types of crashes accordingly to their severity. We simulated our proposal in 16 possible scenarios. The results show the appropriateness of both, the overall system and the decision-making strategy to choose actions that prevents potential accidents of the self-driving car.

Keywords: Autonomous vehicles · Probabilistic logic · Markov decision processes · Computer vision

1 Introduction

The development of self-driving cars have received significant attention in the last decades due to the potential positive effects for the future of mobility. Some

This work was supported by UNAM-DGAPA under grant TA101222 and Consorcio de IA CIMAT-CONACYT.

A. C. Bicharra Garcia et al. (Eds.): IBERAMIA 2022, LNAI 13788, pp. 366–377, 2022.
https://doi.org/10.1007/978-3-031-22419-5_31

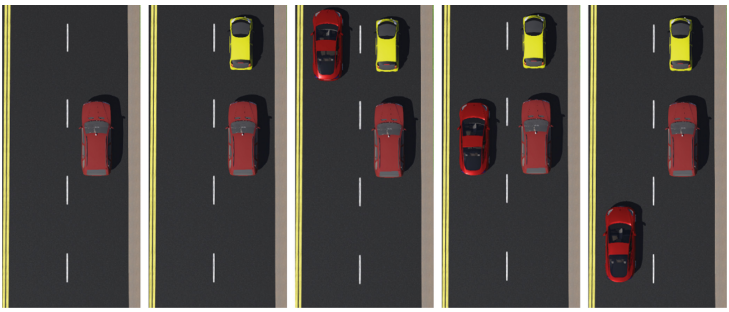

Fig. 1. Example of the modeled environments. Left: Because there is no other car around, the self-driving vehicle moves at a cruise speed. Second left: The self-driving detects another car and should pass it. Three rightmost: Vehicle detects a car in front but should not overtake and just keep distance.

of these benefits include the increase of safety or health care, and the reduction of economic costs [1–3]. An autonomous car has several components that strongly interact among themselves and 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 given the current environment.

Neural networks, rule-based models and probabilistic models are some of the most common representations for behavior selection. On the one hand, neural networks has a outstanding performance in learning from numerical examples and in classifying noisy or partial observations. Unfortunately, the internal knowledge representation of neural networks is not human-readable. This condition hinders the inspection of the decision-making process, that it is essential 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 which simplify the explanation of the 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 of first-order logic. Therefore, in this document, we propose an action policy obtained from a probabilistic logic representation of an MDP as a behavior selection strategy for a self-driving car to avoid accidents with other moving vehicles. The MDP is encoded in MDP-ProbLog [4] that is developed on top of ProbLog [5]. Our approach allows us: i) to express spatial relationships between the self-driving car and the other cars on the road using fairly simple probabilistic logic rules and facts, and ii) to assign negative utilities for different potential crash situations and positive rewards for desirable states and actions. In our setting the self-driving car and four vehicles travel on a one-way street with two lanes. The ego car considers three simple actions: *overtaking*, *keep distance* and *steady motion*. The self-driving starts its movement in the right lane. The perceptual

system reports if there is another car in front of it in the right lane, a car ahead in the left lane, a car aside to the left, or a car behind it in the left lane (we call these positions North, North-West, West and South-West, respectively). Then, the self-driving car has to decide which action is more convenient given the presence or absence of the other vehicles. Figure 1 depicts 5 of the 16 possible settings and the action taken on each one. We performed 160 repetitions of the previous experiment. Our initial results shows a correct classification rate of 92.5% on a simulated environment when testing at low speeds and 88.1% at faster speeds.

The main contribution of this paper is twofold. First, in the best of our knowledge this is the first time a probabilistic logic approach is used to select behaviors in autonomous vehicles. Second, it is shown the suitability of the probabilistic logic representation to describe driving knowledge and to generate action policies that prevent potential collisions.

2 Related Work

In [6] 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 [7] 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 [8] 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.

3 Theoretical Background

3.1 ProbLog Overview

ProbLog is a declarative language and a suite of algorithms that integrate independent ground probabilistic facts into the syntax of Prolog programs. ProbLog supports inductive definitions, cyclic or recursive rules, transitive relations, multiple non-mutually exclusive bodies with the same head, arbitrary atoms as evidence, marginal and joint probabilities given the evidence, and *annotated disjunctions* [9]. ProbLog follows *distribution semantics* proposed by T. Sato [10] for calculating the probability of a query or goal given a probabilistic logic program. A ProbLog program P_L is an ordered pair (F_p, R), where:

1. $F_p = \{f_1, f_2, ..., f_n\}$ is a finite set of independent probabilistic facts
2. $R = \{r_1, r_2, ..., r_m\}$ is a finite set of probabilistic normal rules

A probabilistic fact is a ground Prolog fact augmented with a probability value. They can be also understood as Boolean random variables. In a similar form, probabilistic rules are Prolog rules extended with probabilities. An important restriction is that no head of a rule in R is a member of F_p. The probability of a query q given a probabilistic logic program P_L is obtained by the next equation:

$$P(q|P_L) = \sum_{\forall L, L \models q} P(L|P_L) \tag{1}$$

where L is a ground logic program and the probability of q is equal to the sum of the probability of each ground logic program L that logically entails q. The probability $P(L|P_L)$ is calculated as follows:

$$P(L|P_L) = \prod_{f_i \in L} P(f_i) \prod_{f_i \notin L} (1 - P(f_i)), \tag{2}$$

that is, the product of the probability of every fact f_i assumed to be true or false under the interpretation (total choice) associated to L. For n Boolean facts, 2^n logic programs can be derived. Well-founded semantics [11] is adopted for determining whether a query is logically entailed by a ground logic program via the well-founded model of the logic program L.

3.2 Markov Decision Processes

Markov decision processes [12] are standard models for sequential decision-making. A (discrete time) MDP is a 4-tuple (S, A, P, R) in which:

a) $S = \{s_1, s_2, ..., s_n\}$ is a finite set of n states that are completely observable (i.e., the current state is known)
b) $A = \{a_1, a_2, ..., a_m\}$ is a finite set of m actions

c) $P(s'|a, s)$ is a transition function that models the uncertainty of reaching a future state $s' \in S$ given an action $a \in A$ executed in the current state $s \in S$
d) $R(s, a)$ is a reward function for each pair (s, a)

The reward function is used to compute the cumulative expected reward $V^{\pi}(s)$ of a state s, that corresponds to the immediate reward for the state-action pair, plus the average of the cumulative future rewards from the next states following a fixed policy π. The solution of the MDP is an optimal action policy π^* that maximizes the value $V^{\pi}(s)$ of each state s accordingly to the Bellman's equation:

$$V^{\pi}(s) = R(s, \pi(s) = a) + \gamma \sum_{s' \in S} V^{\pi}(s') P(s'|\pi(s) = a, s) \tag{3}$$

where $\gamma \in [0, 1)$ is a discount factor for infinite-horizon MDP. 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). As the number of states increases, a factorization of the transition function [13] using *state variables* is more convenient. The assignment of values to state variables defines the states of the MDP. This factored representation of states has the advantage of simplifying the transition model by focusing on the effect of the actions over subsets of the state variables.

3.3 MDP-ProbLog

MDP-ProbLog models infinite horizon factored MDPs. The tuple (A_t, F_p, R) is a n MDP-ProbLog program if:

i) A_t is a finite set of atomic formulas that is divided into:
   (1) S_F, a finite set of state (or fluent) variables
   (2) A, a finite set of action predicates
   (3) U, a set of utility predicates
ii) F_p is a finite set of auxiliary probabilistic facts
iii) R is a set of rules categorized as:
   (a) T_r, a finite set of transition rules
   (b) R_r, a finite set of reward rules

Special predicates are used to declare these elements. For example, state variables are identified with the unary predicate 'state_fluent', and actions are denoted with the predicate 'action' that receive as parameter the identifier of a state variable and action, respectively. The predicate 'utility' assigns a positive or negative reward to actions and states of the MDP. MDP-ProbLog uses the algorithm value iteration for solving the MDP.

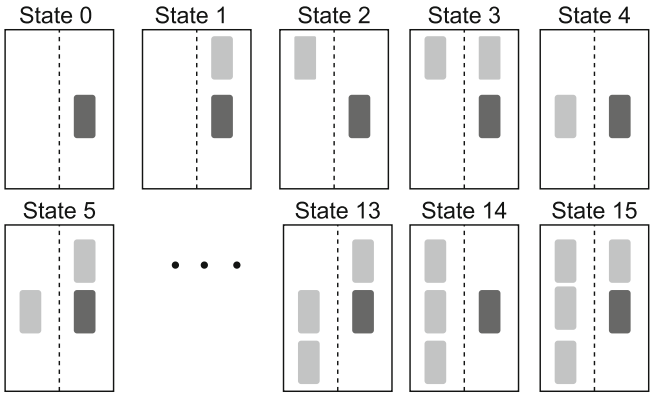

Fig. 2. MDP States to model the environment. The dark grey rectangle represents the self-driving car and the light grey rectangles are the nearby (obstacle) vehicles.

4 Methodology

4.1 Design of the Probabilistic Logic MDP

As stated above, our setting consists on a autonomous vehicle moving at a steady speed and every time it has to decide whether to continue at a constant speed, to overtake or to keep its distance accordingly to the existence of other cars nearby. Figure 2 shows some of the states considered to model the environment.

Herein after North, North-West, West and South-West will be referred to as N, NW, W and SW, respectively. States are factored by four Boolean state variables that we call *free_N*, *free_NW*, *free_W* and *free_SW*. These state variables represent regions around the car which can be free or occupied. In MDP-ProbLog, the syntax for the declaration of the state variables is simple. For example, state_fluent(free_N) is the definition of the variable free_N. The definition of the actions is straightforward too, e.g., action(keep_distance).

The transition model of the MDP-ProbLog requires to know the truth value of each state variable. These values are obtained by our visual perception system as discussed in Sect. 4.2. Understandable probabilistic logic rules were designed to model the transition to potential future states given the current values of the state variables and the actions. For example, for the overtaking action we have:

```
0.9::free_N(1) :- free_NW(0), free_W(0), overtaking.
0.05::free_N(1) :- not(free_NW(0)), overtaking.
0.05::free_N(1) :- not(free_W(0)), overtaking.
```

Accordingly to first rule, free_N will be true in the next epoch with a probability of 0.9 if free_NW and free_W are true and the action overtaking is performed (i.e., there is enough space to overtake). The latter two rules indicate that there is a low probability of 0.05 of observing free_N as true if there is not free space to the northwest or to the west. Initially, we manually designed probabilistic logic

rules for the factored transition function by considering all the state variables. This approach resulted in several probabilistic logic rules. However, we reduced the rules for each action by applying the resolution-like algorithm by hand on rules that share the same probability value. In this manner, we derived fewer and simpler rules that subsumes the original ones. Finally, the reward model is defined as:

```
utility(free_N(1), 5).
utility(rear_crash(1), -30).
utility(side_crash(1), -10).
utility(keep_distance, -10).
utility(overtaking, -1).

0.99::rear_crash(1) :- not(free_N(1)), steady_motion,
not(keep_distance).
0.95::rear_crash(1) :- not(free_NW(1)), overtaking.
0.95::side_crash(1) :- not(free_W(1)),  overtaking.
```

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 rear crashes and side swipe crashes (Fig. 3). These negatives rewards reflects a difference on the severity of the consequences of these types of accidents. The reward is also negative for overtaking, as we consider that this action involves some risk of accident, and for keeping the distance, because selecting that action delays the arrival to the destination.

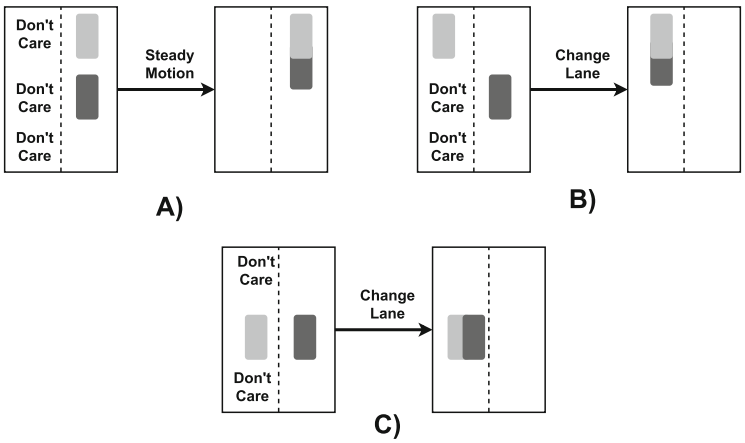

Fig. 3. Examples of states and actions that lead to the different types of crashes in our setting: A) and B) show rear crashes. C) shows a side swipe

4.2 Perceptual System

To simulate the environment we used the Webots simulator [14]. 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. Noise was added to both sensors using the tools provided by Webots. The simulated car allows to set linear speed v and steering δ as control inputs.

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

Figure 4 show the general steps to detect lanes. Lane tracking is performed using a proportional control that combines distance and angle to the lane border. Consider the Fig. 5. 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. Consider the kinematic model of the vehicle:

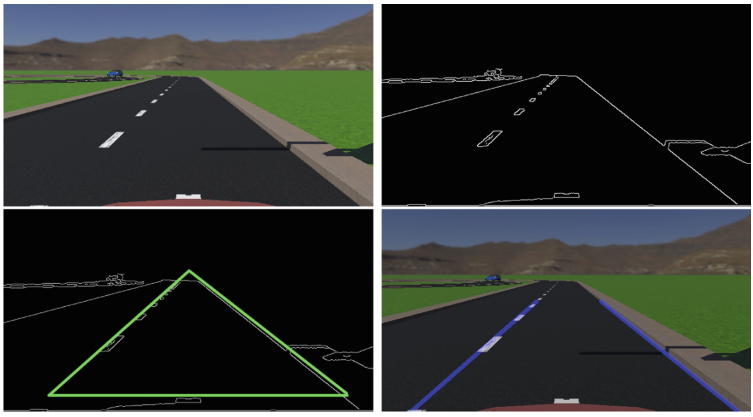

Fig. 4. Segmentation Process. From top-left to bottom-right: Original image, border detection with Canny, triangular region of interest and resulting segmented lines.

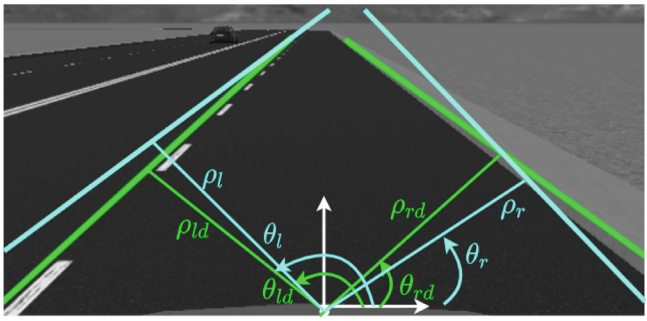

Fig. 5. Variables for lane tracking

$$\dot{x} = v\cos(\theta + \delta) \tag{4}$$
$$\dot{y} = v\sin(\theta + \delta) \tag{5}$$
$$\dot{\theta} = \frac{v}{l}\sin(\delta) \tag{6}$$

with $[x, y, \theta]$, the current configuration of the vehicle, l, the distance from rear to front tires and (v, δ) the linear speed (set by throttle and brake) and steering angle, respectively, taken as control inputs. We propose the following control laws to track lane:

$$v = C_b \tag{7}$$
$$\delta = K_\rho e_\rho + K_\theta e_\theta \tag{8}$$

with

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

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: cruise motion, keep distance and overtaking.

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.

5 Experiments and Results

Table 1 represents the policy obtained with the MDP previously described. The symbol **F** means there is a car to the N, NW, SW or W and **T** means that the corresponding position is free. Depending of what is detected using the lidar sensor, we choose one of the three behaviors.

Table 1. Policy obtained from the probabilistic logic MDP.

N	NW	SW	W	Action	N	NW	SW	W	Action
F	**F**	**F**	**F**	keep distance	**F**	**F**	**F**	**T**	keep distance
T	**F**	**F**	**F**	Steady motion	**T**	**F**	**F**	**T**	Steady motion
F	**T**	**F**	**F**	keep distance	**F**	**T**	**F**	**T**	Overtaking
T	**T**	**F**	**F**	Steady motion	**T**	**T**	**F**	**T**	Steady motion
F	**F**	**T**	**F**	keep distance	**F**	**F**	**T**	**T**	keep distance
T	**F**	**T**	**F**	Steady motion	**T**	**F**	**T**	**T**	Steady motion
F	**T**	**T**	**F**	keep distance	**F**	**T**	**T**	**T**	Overtaking
T	**T**	**T**	**F**	Steady motion	**T**	**T**	**T**	**T**	Steady motion

This policy was implemented using a finite state machine with the vehicles positions as input signals and the chosen behavior as output signals. In order to add uncertainty to the environment, we added Gaussian noise with zero-mean both to camera and lidar. Camera noise has a Std Dev of 5% of the maximum value. Lidar noise has a Std Dev of 0.1 m. In each test, speeds of surrounding cars were set randomly with uniform distribution in the interval (3.0, 4.0).

To evaluate our approach, we performed an experiment with 20 repetitions of each possible configuration listed in Table 1. 10 repetitions with slow random speeds (in the range [3.5,5] m/s) and another 10 repetitions with faster random speeds (in the range [5,10] m/s). The confusion matrix of the 160 repetitions for slower speeds, that includes the number of repetitions in which the system took correct and wrong actions is shown in Table 2. The confusion matrix for faster speeds is shown in Table 3. Figure 6 shows an example situation where the car chooses to overtake. As it is shown, in most cases, the car performed the expected action and only in the 7.5% of cases, for lower speeds, and 11.87% for faster speeds, the car selected a non-expected behavior. However, only in 4 cases the car crashed with one of the surrounding vehicles, when moving at lower speed, and only in 7 cases, when moving at faster speeds, most likely because an error in the position estimations.

Table 2. Performance of the decision system with speeds in [3.5, 5] m/s

	Chosen action		
Ideal action	Steady motion	Keep distance	Overtaking
Steady motion	80	0	0
Keep distance	0	57	3
Overtaking	0	9	11

Table 3. Performance of the decision system with speeds in [5, 10] m/s

	Chosen action		
Ideal action	Steady motion	Keep distance	Overtaking
Steady motion	80	0	0
Keep distance	0	44	16
Overtaking	0	3	17

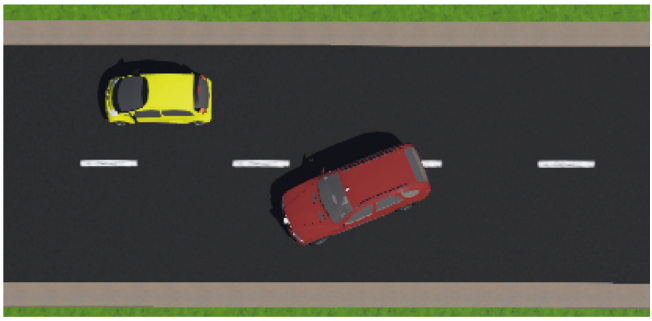

Fig. 6. Example of the overtaking behavior.

6 Conclusions and Future Work

We have presented a probabilistic logic method to generate the action policies of an autonomous vehicle in three behaviors: steady motion, keep distance and overtaking. We tested the method in a simulated environment using Webots and MDP-ProbLog. The test environment consisted of two lanes, with four possible vehicles surrounding the autonomous vehicle. The results obtained demonstrate that the vehicle can be driven safely and collision-free by making use of the generated policies. As future work, the system will be tested considering more lanes and other possible collision risk situations. Also, the decision system will be improved to take into account the speed of surrounding vehicles, and not only their relative positions. Finally, we propose to implement our proposal in AutoMiny Version 4, an open source 1:10 scale autonomous vehicle developed at the Free University of Berlin, and that is equipped with a small PC, a Lidar sensor, a RGB-D camera, an IMU, and actuators for driving and steering. Testing will be performed in a small car test track of 3×6 m.

Acknowledgments. The authors would like to thank Vincent Derkinderen (KU Leuven) and Thiago P. Bueno (University of São Paulo) for their comments and guidance and the two anonymous reviewers for their insightful suggestions.

References

1. Silva, O., Cordera, R., González-González, E., Nogués, S.: Environmental impacts of autonomous vehicles: a review of the scientific literature. Sci. Total Environ. **830**, 154615 (2022)
2. Howard, D., Dai, D.: Public perceptions of self-driving cars: the case of Berkeley, California. In: Transportation Research Board 93rd Annual Meeting, vol. 14, pp. 1–16 (2014)
3. Xu, W., Wei, J., Dolan, J.M., Zhao, H., Zha, H.: A real-time motion planner with trajectory optimization for autonomous vehicles. In: 2012 IEEE International Conference on Robotics and Automation, pp. 2061–2067. IEEE (2012)
4. Bueno, T.P., Mauá, D.D., De Barros, L.N., Cozman, F.G.: Markov decision processes specified by probabilistic logic programming: representation and solution. In: 2016 5th Brazilian Conference on Intelligent Systems (BRACIS), pp. 337–342. IEEE (2016)
5. Fierens, D., et al.: Inference and learning in probabilistic logic programs using weighted boolean formulas. Theory Pract. Logic Program. **15**(3), 358–401 (2015)
6. Chen, K., Yang, B., Pei, X., Guo, X.: Hierarchical control strategy towards safe driving of autonomous vehicles. J. Intell. Fuzzy Syst. **34**(4), 2197–2212 (2018)
7. Huang, L., Liang, H., Yu, B., Li, B., Zhu, H.: Ontology-based driving scene modeling, situation assessment and decision making for autonomous vehicles. In: 2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS), pp. 57–62. IEEE (2019)
8. Rizaldi, A., Althoff, M.: Formalising traffic rules for accountability of autonomous vehicles. In: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, pp. 1658–1665. IEEE (2015)
9. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) ICLP 2004. LNCS, vol. 3132, pp. 431–445. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27775-0_30
10. Sato, T., Kameya, Y.: PRISM: a language for symbolic-statistical modeling. In: IJCAI, vol. 97, pp. 1330–1339. Citeseer (1997)
11. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM (JACM) **38**(3), 619–649 (1991)
12. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons (2014)
13. Hoey, J., St-Aubin, R., Hu, A., Boutilier, C.: Spudd: stochastic planning using decision diagrams. In: Proceedings of International Conference on Uncertainty in Artificial Intelligence (UAI 1999) (1999)
14. Webots. http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software