Taylor & Francis
Taylor & Francis Group

Applied Artificial Intelligence
An International Journal

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/uaai20

Autonomous Behavior Selection For Self-driving Cars Using Probabilistic Logic Factored Markov Decision Processes

Héctor Avilés, Marco Negrete, Alberto Reyes, Rubén Machucho, Karelly Rivera, Gloria de-la-Garza & Alberto Petrilli

To cite this article: Héctor Avilés, Marco Negrete, Alberto Reyes, Rubén Machucho, Karelly Rivera, Gloria de-la-Garza & Alberto Petrilli (2024) Autonomous Behavior Selection For Self-driving Cars Using Probabilistic Logic Factored Markov Decision Processes, Applied Artificial Intelligence, 38:1, 2304942, DOI: 10.1080/08839514.2024.2304942

To link to this article: https://doi.org/10.1080/08839514.2024.2304942

APPLIED ARTIFICIAL INTELLIGENCE
2024, VOL. 38, NO. 1, e2304942 (25 pages)
https://doi.org/10.1080/08839514.2024.2304942

Autonomous Behavior Selection For Self-driving Cars Using Probabilistic Logic Factored Markov Decision Processes

Héctor Avilés[a], Marco Negrete[b], Alberto Reyes[c], Rubén Machucho[a], Karelly Rivera[a], Gloria de-la-Garza[a], and Alberto Petrilli[d]

[a]Information Technologies Program, Polytechnic University of Victoria, Victoria, Tamaulipas, Mexico; [b]Faculty of Engineering, National Autonomous University of Mexico, Mexico city, Mexico; [c]Control, Electronics and Communications, National Institute of Electricity and Clean Energies, Morelos, Mexico; [d]Department of Robotics, Tohoku University, Sendai, Miyagi, Japan

ABSTRACT

We propose probabilistic logic factored Markov decision processes (*PL-fMDPs*) as a behavior selection scheme for self-driving cars. Probabilistic logic combines logic programming with probability theory to achieve clear, rule-based knowledge descriptions of multivariate probability distributions, and a flexible mixture of deductive and probabilistic inferences. Factored Markov decision processes (*fMDPs*) are widely used to generate reward-optimal action policies for stochastic sequential decision problems. For evaluation, we developed a simulated self-driving car with reliable modules for behavior selection, perception, and control. The behavior selection module is composed of a two-level structure of four action policies obtained from PL-fMDPs. Three main tests were conducted focused on the selection of the appropriate actions in specific driving scenarios, and the overtaking of static obstacle vehicles and dynamic obstacle vehicles. We performed 520 repetitions of these tests. The self-driving car completed its task without collisions in 99.2% of the repetitions. Results show the suitability of the overall self-driving strategy and PL-fMDPs to construct safe action policies for self-driving cars.

ARTICLE HISTORY
Received 10 October 2023
Revised 11 December 2023
Accepted 2 January 2024

Introduction

The appropriate selection of driving behaviors to respond to the different scenarios on the road is essential to achieving full automation in self-driving cars. By driving behaviors, we understand reactive or short-term actions such as "obstacle avoidance," "acceleration," or "stopping short" that a self-driving car must choose and execute appropriately to reach its destination safely (Da Lio et al. 2023).

A variety of strategies for the selection of driving behaviors are available (Liu et al. 2021). However, artificial neural networks (ANNs), *first-order logic*

CONTACT Marco Negrete marco.negrete@ingenieria.unam.edu Faculty of Engineering, National Autonomous University of Mexico, 04510, Mexico
Supplemental data for this article can be accessed online at https://doi.org/10.1080/08839514.2024.2304942.

rule systems, and probabilistic models are the most common approaches to that purpose (Aksjonov and Kyrki 2022; Paden et al. 2016; Sun et al. 2020). ANNs are outstanding at learning and recognition of noisy, cluttered, or partial data (Cai et al. 2016). They can also absorb controlling tasks of other specialized components (Zhu et al. 2022).

Despite their advantages, inference in ANNs requires the parallel propagation of input signals through a massive quantity of weighted links among their processing units. This process obscures the influence of the components of the ANNs on their output, and external *ad-hoc* mechanisms are required to achieve some *explainability* and *interpretability* (Padalkar, Wang, and Gupta 2023; Zhang et al. 2021). These features are crucial in self-driving cars, for example, when inspecting the source of a car malfunction or determining legal responsibilities in a collision scenario. In contrast to ANNs, rule-based systems inherently provide rich characterizations of a problem domain by expressing properties and relationships between objects through interpretable logic rules and facts. They also offer traceable inference mechanisms that contribute to explaining the conclusions (Mishra 2022). Moreover, probability theory is the most prevalent approach for handling uncertainty. It extends the binary truth values $\{1, 0\}$ of first-order logic to a *degree of belief* in the interval $[0, 1] \subset \mathbb{R}$. Consequently, it is not surprising that there is a growing interest in probabilistic logic frameworks that blend logical clarity with flexible combinations of deductive and probabilistic inferences (Riguzzi 2022).

Therefore, in this paper, we propose probabilistic logic factored Markov decision processes (*PL-fMDPs*) to generate action policies that serve as a behavior selection scheme for self-driving cars. PL-fMDPs are factored Markov decision processes (*fMDPs*) (Boutilier, Dearden, and Goldszmidt 2000) coded in a probabilistic logic programming language. We use MDP-ProbLog (Bueno et al. 2016) which is a specialized programming language developed on top of ProbLog (Fierens et al. 2015) to represent and solve discrete, infinite-horizon fMDPs. MDP-ProbLog considers Boolean *state fluents* to describe the state space of the system, the probable effect of the actions over the state fluents, and utility values for state fluents, actions, and auxiliary atoms. The solution of a PL-fMDP is a deterministic, reward-optimal action policy (called from now on simply as "policy"). Following this approach, simple probabilistic rules and facts allow us to express complex spatial relationships between the self-driving car and other vehicles on the road, in a comprehensible and readable manner. For evaluation, we developed a self-driving car using a realistic simulator. In addition to behavior selection, the architecture of the self-driving car includes perception and control components. We utilize simple control laws and perception techniques, as we concentrate our efforts on behavior selection and our approach does not require highly accurate model-based controllers or precise estimation of obstacle position and speed. The behavior selection module is composed of four

policies arranged in a two-level hierarchy and obtained from four PL-fMDPs that handle eight Boolean state fluents and four driving actions. The test bench consists of a straight road with 2 one-way lanes, a self-driving car and from 0 to 10 obstacle vehicles, either stationary or in motion, at different speeds. We conducted three primary tests, each increasing in complexity, to evaluate (a) the selection of the most convenient action in specific driving scenarios, (b) overtaking static obstacle vehicles, and (c) overtaking obstacle vehicles in motion. The system was tested across a total of 520 scenarios, arising from variations in the self-driving car's speed, the number of obstacle vehicles, and their starting positions and speeds. Each scenario was tested once. On average, the self-driving car finished its task correctly in 99.2% of the trials. Results show the appropriateness of both, the overall self-driving strategy and PL-fMDPs to construct safe behavior selection policies.

This paper extends our prior work presented in (Avilés et al. 2022), in which we assessed the identification of the driving scenarios and the selection of actions using a single policy obtained from a single PL-fMDP. Notably, we have reorganized the overall software architecture of the self-driving car, fine-tuned decision thresholds for some perceptual and control capabilities, and addressed a few minor software glitches. We also introduce three new PL-fMDPs to construct a behavior selection module that comprises four complementary policies. Finally, we performed two new tests and repeated the tests originally performed in the early paper to assess the selection of actions, achieving improved outcomes.

Related Work

In (Smith, Petrick, and Belle 2021), a framework for integrated logical and probabilistic intent reasoning using basic switch sensor data from smart homes and assisted living facilities is introduced. Utilizing ProbLog, the framework infers the most likely intention based on agent actions and crucial sensor readings from the environment. In (Al-Nuaimi et al. 2021), a novel approach combining the model checker for multi-agent systems (MCMAS) and the probabilistic model checker (PRISM) for autonomous vehicles (AV) is introduced. The system presented includes a perception system feeding data into a rational agent (RA) based on a belief-desire-intention (BDI) architecture, which uses a model of the environment and is connected to the RA to verify its decision-making process. MCMAS is used to verify the agent's logic during design, and PRISM provides probabilities during run-time. Testing on a new AV software platform and a parking lot scenario confirms the feasibility, and practical implementation on an experimental test is carried out (Mu et al. 2022). focuses on optimizing the selection of behaviors for a robot through a repetitive prediction-selection process. In this approach, the framework predicts how humans will respond to a series of consecutive robot-guiding

behaviors over the long term. The goal is to determine the best sequence of actions using a technique called Monte Carlo Tree Search (MCTS). By doing so, the robot aims to enhance human comprehension of its active instructions and facilitate their decision-making regarding when and where to transition between different behaviors. This approach seeks to improve the effectiveness of human-robot interaction during guidance tasks. In (Spanogiannopoulos, Zweiri, and Seneviratne 2022), the authors propose a novel approach for generating collision-free paths in non-holonomic self-driving cars using an incremental sampling-based planner like Rapidly-exploring Random Trees (RRT). The approach generates incremental collision-free path segments from the start to end configuration, relying only on local sensor data (point cloud data) in quasi-static unknown environments. The planner generates real-time paths that ensure safety and has been extensively tested and validated in popular benchmark simulation environments. In (Dortmans and Punter 2022), provides an overview of lessons learned from applying Behavior Trees to autonomous robots. For the design of the "action selection policy" (a mapping from states to actions), they propose a method that simplifies the representation of the world's state space using Boolean conditions as predicates. Additionally, the robot's complex algorithms for actuation are summed up in a set of actions. These actions are responsible for transitioning the world from one pre-condition state to a post-condition state with a high probability of success. This approach expedites the representation of the world and robot behaviors, making it more suitable for decision-making and control processes. In (Yuan, Shan, and Mi 2022), the authors use deep reinforcement learning and game theory for the decision-making of an ego-vehicle at an unsignalized intersection, where there are other vehicles. The behaviors of these vehicles are conservative, aggressive, and adaptive. There is no communication between the vehicles. The tests are carried out in the Gazebo-ROS simulator. In (Duan et al. 2020), the authors use interconnected neural networks to build a master driving policy, and three maneuvering policies for driving an autonomous vehicle. The maneuvers are driving in the lane, and changing between the left and right lanes. Each maneuver has sub-policies that allow low-level movements of the vehicle to be carried out in the lateral and longitudinal directions. With this hierarchy, they manage to reduce the amount of data necessary in supervised learning, which requires covering all possible management scenarios. Autonomous driving has also been addressed using an optimal control approach. This approach consists of minimizing a cost function subject to several restrictions: system dynamics and obstacle shapes, positions, or velocities. The solution of such minimization results in the control input signals to drive the vehicle toward a goal point or trajectory while avoiding obstacles. Examples of this approach are (Lindqvist et al. 2020; Pepe et al. 2019; Zhang, Liniger, and Borrelli 2021) and (Wang et al. 2023). The drawback of the optimal control approach for movement planning in self-driving cars is the

assumption that obstacles have positions and velocities that can be described analytically, sometimes even constant positions are assumed. We take a probabilistic approach to develop our behavior selection system due to the inherent handling of uncertainty. Application of optimal control in problems different to path planning for self-driving cars can be found in Precup et al. (2008), Unguritu and Nichitelea (2023), and Rigatos et al. (2016).

ProbLog Overview

ProbLog is a probabilistic logic programming language to describe discrete, multivariate probability distributions and answers probabilistic queries. It extends the syntax of Prolog with probability values.

A ProbLog program is a tuple $(\mathcal{L}, \mathcal{F})$ where $\mathcal{L}$ is a logic program (a finite set of logic rules and facts), and $\mathcal{F}$ is a finite set of probabilistic facts with the syntax $\alpha::f$, such that f is a *(positive) ground atomic formula*,[1] and $\alpha \in [0, 1]$. Probabilistic facts can be understood as mutually independent Bernoulli random variables, each one with a success probability. ProbLog also accepts *normal* rules with probability values, e.g., "$\alpha::a:-\ b_1, \ldots, b_n,\ +\ b_{n+1}, \ldots,\ +\ b_{n+\ell}.$", where $n, \ell \in \mathbb{N}_{\not\vdash}$, $a, b_1, \ldots, b_n$ are atoms $+b_{n+1}, \ldots, +b_{n+\ell}$ are *negative atoms*, and $\alpha \in [0, 1]$ is the probability value attached to the atom a. The sub-expression to the left of the symbol ":-" (which means "if") is the *head* of the clause, and the sub-expression to the right is its *body*. In the body, commas function as logical conjunctions. Semicolons can be used as logical (inclusive) disjunctions. The symbol\+ means *negation as failure*. Therefore, the normal rule can be read as follows: "atom a will succeed with probability α *if* the atoms from b_1 to b_n are true and the atoms from b_{n+1} to $b_{n+\ell}$ are not true." In probability theory notation, this rule is expressed as $p(a|b_1, \ldots, b_n, \neg b_{n+1}, \ldots, \neg b_{n+\ell}) = \alpha$ (ProbLog does not allow the definition of the probability of failure of an atom; instead, this value is calculated internally). Notice that probabilistic rules are actually traditional logic rules extended with an auxiliary probabilistic fact. For example, the rule "$0.4::a_1:-\ b_1, +b_2.$"is treated by ProbLog as a combination of the probabilistic fact "$0.4::aux.$"with the pure logic rule "$a_1:-\ b_1, +b_2, aux.$".

ProbLog allows *annotated disjunctions* (Vennekens, Verbaeten, and Bruynooghe 2004). A fact with annotated disjunctions "$\alpha_1::a_1; \alpha_2::a_2; \cdots; \alpha_n::a_n.$" emulates a finite, discrete random variable that takes $n \geq 2$ mutually exclusive, categorical values a_i with a probability of occurrence α_i, for all $i = 1, \ldots, n$, and $\sum_{i=1}^{n} \alpha_i = 1$. A rule with annotated disjunctions has the form "$\alpha_1::a_1; \alpha_2::a_2; \ldots \alpha_m::a_m:-\ body.$", where $m \geq 2$, and $\sum_{i=1}^{m} \alpha_i = 1$. The body is a *cause* that produces mutually exclusive *effects* a_i, each one with a probability of success α_i, for all $i = 1, \ldots, m$.

For probabilistic inference, ProbLog follows the *distribution semantics* (Sato 1995). Let PL be a ProbLog program. Assume that the probabilistic facts $\mathcal{F} = \{f_1, \ldots, f_n\} \subset PL$ have a fixed order. In addition, let $\mathcal{P}(\mathcal{F}) = \{\mathrm{F}_i\}_{i=1}^{2^n}$ be the *power set* of $\mathcal{F}$. The union of the subset $\mathrm{F}_i \in \mathcal{P}(\mathcal{F})$ of $\mathcal{F}$ with the logic part $\mathcal{L}$ of PL induces a new logic program $L_i = \mathcal{L} \cup \mathrm{F}_i$, for $i = 1, \ldots, 2^n$. Also, let L be the set of all the possible logic programs obtained from PL with the previous procedure. A probability distribution over the logic programs $L_i \in \mathrm{L}$, for all $i = 1, \ldots, 2^n$ is calculated as follows:

$$p(L_i|PL) = \prod_{j=1}^{n} \alpha_j \left[\left[f_j \in L_i\right]\right] + (1 - \alpha_j)\left[\left[f_j \notin L_i\right]\right] \tag{1}$$

where α_j is the probability value associated to the probabilistic fact f_j, for $j = 1, \ldots, n$, $P(L_i|PL) \in [0, 1]$, for $i = 1, \ldots, 2^n$, and $\sum_{i=1}^{2^n} P(L_i|PL) = 1$.[2] ProbLog supports querying of the *most probable explanation*, *marginal probabilities*, and *conditional probabilities* of ground and non-ground atoms. Here, we will focus on conditional probabilities of ground queries, as they are the ones required by MDP-ProbLog. Let $\mathcal{H}_b$ be the *Herbrand base* of PL (The Herbrand base is the set of all grounded atoms that can be formed with atoms and terms in the theory PL). We assume here a finite Herbrand base. Moreover, let $q \in \mathcal{H}_b$ be a ground atom query. Additionally, assume that $E \in \mathcal{H}_b$ is a subset of ground atoms for which the Boolean truth values $\mathbf{e}$ have been observed. This truth assignment is referred to as the *evidence* and is denoted as $E = \mathbf{e}$. The answer to a query request of the ground atom q given the evidence $E = \mathbf{e}$ is the success probability value $p(q|PL; E = \mathbf{e})$ of q, such that $PL \models q$. This probability is calculated as follows:

$$p(q|PL; E = \mathbf{e}) = \frac{\sum_{i=1}^{2^n} p(L_i|PL)\left[\left[L_i \models q \wedge E = \mathbf{e}\right]\right]}{\sum_{j=1}^{2^n} p(L_j|PL)\left[\left[L_j \models E = \mathbf{e}\right]\right]} \tag{2}$$

Logical consequences in ProbLog are determined by a two-valued *well-founded semantics* (Van Gelder, Ross, and Schlipf 1991). ProbLog efficiently solves probabilistic inference by constructing a *ground program* L_g containing relevant facts and rules found through *SLD-resolution* from the grounding of the program PL. It also eliminates loops in rules and generates a Clark's complete (Clark 1978) Boolean sentence in *conjunctive normal form* (*CNF*). The CNF is then compiled into a weighted Boolean formula and an *arithmetic circuit* for *weighted model counting* (Chavira and Darwiche 2008).

Factored Markov Decision Processes

2.1. Elements of fMDPs Factored Markov decision processes model stochastic sequential decision-making problems that involve an *agent* who decides actions and the *environment* in which the agent operates. The states of the system are defined through the values of a set of *state variables*. Transitions among the states are uncertain when an action is executed, and numerical feedback called a reward is provided recurrently to the agent for each state-action pair. Examples showcasing the wide application of fMDPs in decision-making can be found in (Avilés-Arriaga et al. 2009; Reyes et al. 2020), where fMDPs are employed in power plants and service robots, respectively.

An fMDP is a 5-tuple $(\mathrm{X}, \mathbf{X}, \mathcal{A}, p, R)$, in which:

i) $\mathrm{X} = \{X_i\}_{i=1}^{n}$ is a finite, totally ordered set of n discrete state random variables.[3]
ii) $\mathbf{X}$ is a set of all possible n-tuples $\mathbf{x} = (x_i)_{i=1}^{n}$, such that x_i is the value of the random variable, for all $i = 1, \ldots, n$ (there exists a one-to-one correspondence between $\mathbf{X}$ and the state-space of the system).
iii) $\mathcal{A}$ is a finite set of actions the agent can choose.
iv) $p_{X'_1,\ldots,X'_n|X_1,\ldots,X_n}(\mathbf{x}'|\mathbf{x}, a) \in [0, 1]$ is a discrete, joint probability transition function, $(\mathbf{x}, a) \in \mathbf{X} \times \mathcal{A}$, and $\mathbf{x}' \in \mathbf{X}$ ($\mathbf{x}'$ is a *post-action* state, and $\mathbf{x}$ is a *pre-action* state).
v) $R(\mathbf{x}, a)$ is a reward function, where $(\mathbf{x}, a) \in \mathbf{X} \times \mathcal{A}$.

In fMDPs, it is frequently taken for granted that all post-action state variables are mutually independent among them given the current state $\mathbf{x}$ and an action a (Equation (3)). Moreover, further *conditional independence* assumptions are applied to eliminate some of the pre-action state variables that do not affect the probability value of X'_i (Equation (4)):

$$p_{X'_1,\ldots,X'_n|X_1,\ldots,X_n}(\mathbf{x}'|\mathbf{x}, a) = \prod_{i=1}^{n} p_{X'_i}(x'_i|\mathbf{x}, a) \tag{3}$$

$$= \prod_{i=1}^{n} p_{X'_i}(x'_i|\mathrm{obs}(\mathrm{Pa}(X'_i)), a) \tag{4}$$

where $\mathrm{obs}(\mathrm{Pa}(X_{i'}))$ stands for the observed values of the *parent* pre-action state variables $\mathrm{Pa}(X_{i'}) \subseteq \mathrm{X}$ of $X_{i'}$ that along with the action a influence the probability of $X_{i'}$. From now on, random variables as subscripts in the probability theory notation $p_{X'_i}(x_{i'}|\mathrm{obs}(\mathrm{Pa}(X'_i)), a)$ will be omitted whenever it is clear that x'_i is the value of the post-action state variable X'_i.

Solution of fMDPs

In this work, the solution of an fMDP is a *reward-optimal (deterministic) policy* $\pi_*(\mathbf{x}) = a \in \mathcal{A}$, for all $\mathbf{x} \in \mathbf{X}$. The policy π_* maximizes the expected cumulative discounted return $v_*(\mathbf{x})$ for every state $\mathbf{x} \in \mathbf{X}$ with $\gamma \in [0, 1]$ as the discount factor. Often, v_* is calculated prior to π_* through an iterative approximation based on the *Bellman update equation*:

$$v_{k+1}(\mathbf{x}) = \max_a [R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}' \in \mathbf{X}} v_k(\mathbf{x}') \prod_{i=1}^{n} p(x'_i | \text{obs}(\text{Pa}(X'_i)), a)] \tag{5}$$

for all $\mathbf{x} \in \mathbf{X}$. Equation (5) is repeated until the largest absolute difference $|v_{k+1}(\mathbf{x}) - v_k(\mathbf{x})|$ for any state $\mathbf{x} \in \mathbf{X}$ in two consecutive iterations is smaller than some error threshold, namely, $\frac{\epsilon(1-\gamma)}{2\gamma}$, where $\epsilon > 0$ establishes a maximum error bound, and as a consequence $|v_{k+1}(\mathbf{x}) - v_*(\mathbf{x})| < \frac{\epsilon}{2}$ (Puterman 1994). Upon convergence, the ϵ-optimal policy π_ϵ is recovered by the equation $\pi_\epsilon(\mathbf{x}) = \arg\max_a \left[R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}' \in \mathbf{X}} v_{k+1}(\mathbf{x}') p(\mathbf{x}' | \mathbf{x}, a)\right]$, for each $\mathbf{x} \in \mathbf{X}$. This method is known as *value iteration algorithm*.

Probabilistic Logic Factored Markov Decision Processes

Definition

A probabilistic logic factored Markov decision process is a tuple $(\mathrm{X}_F, \mathcal{A}, \mathcal{U}, \mathcal{T}_r, \mathcal{R}_r, \mathcal{C})$, such that:

(1) $\mathrm{X}_F = \{\mathrm{x}_i\}_{i=1}^n$ is a finite, totally ordered set of n state fluents.
(2) $\mathcal{A} = \{a_i\}_{i=1}^m$ is a finite, non-empty set of m actions.
(3) $\mathcal{U}$ is a finite set of utility assignments.
(4) $\mathcal{T}_r$ is a finite non-empty set of factored transition rules.
(5) $\mathcal{R}_r$ is a finite set of rules called the *reward model*.
(6) $\mathcal{C}$ is a finite set of complementary or auxiliary atoms (atoms that are neither state fluents nor actions).

State fluents in X_F are probabilistic facts that correspond to (Boolean) state random variables in fMDPs.[4] Let $\mathbf{X}$ be the set of m Boolean n-tuples $\mathbf{x} = (x_i)_{i=1}^n$, such that $x_i \in \{1, 0\}$ is the truth value of the i-th state fluent x_i, for all $i = 1, \ldots, n$. Each joint truth value assignment $\mathbf{x}$ for the state fluents defines one and only one state of the system and vice versa. Utility assignments are ordered pairs $(a, u) \in \mathcal{U}$, where a is either a state fluent, an action, or a complementary atom. The value $u \in \mathbb{R}$ is the utility assigned to a. Rules in $\mathcal{T}_r$ represent conditional probability functions of the type $p(\mathrm{x}' | \text{obs}(\text{Pa}(\mathrm{x}')), a)$, where $\mathrm{x}' \in \mathrm{X}_F$ is a positive post-action state fluent (i.e., $\mathrm{x}' = 1$), and

$\text{obs}(\text{Pa}(\text{x}')) \subseteq \text{X}_F$ are the truth values of the subset of pre-action state fluents $\text{Pa}(\text{x}')$ that together with $a \in \mathcal{A}$ influence the truth value of x'. The reward model $\mathcal{R}_r$ is the set of probabilistic rules that restrict the utility assignments.

Policy Calculation

MDP-ProbLog uses value iteration to compute the $\in$-optimal policy $\pi_\in$ (Sec. 2.2). The main difference with respect to traditional fMDPs is that MDP-probLog specifies the reward function $R(\mathbf{x}, a)$ through a weighted linear combination of utilities over state fluents, actions, and complementary atoms:

$$R(\mathbf{x}, a) = \sum_{\substack{(\text{x},u)\in\mathcal{U} \\ \text{x}\in\text{X}_F}} up(\text{x}|\mathbf{x}, a) + \sum_{\substack{(\text{a},u)\in\mathcal{U} \\ \text{a}\in\mathcal{A}}} up(\text{a}|\mathbf{x}, a) + \sum_{\substack{(\text{c},u)\in\mathcal{U} \\ \text{c}\in\mathcal{C}}} up(\text{c}|\mathbf{x}, a) \tag{6}$$

for all pairs of evidence $(\mathbf{x}, a) \in \mathbf{X} \times \mathcal{A}$. Conditional probabilities $p(\cdot|\mathbf{x}, a)$ are obtained from ProbLog. Additive utility of (independent) state fluents means that a utility $u \in \mathbb{R}$ associated to a (positive) state fluent x will be weighted and added up to $R(\mathbf{x}, a)$ as long as x is consistent with the evidence $(\mathbf{x}, a)$. Similarly, an action $\text{a} \in \mathcal{A}$ with an utility u will contribute to $R(\mathbf{x}, a)$ independently on the state $\mathbf{x}$ in which a it is performed, whenever $\text{a} = a$. Utilities of complementary atoms serve the purpose of rewarding or penalizing specific combinations of state fluents and actions, thereby enhancing flexibility in knowledge description within this framework. An example illustrating this concept for calculating policies is provided in Section 4.4. ProbLog provides the probability value of each post-action state fluent. The probability transition function $p(\mathbf{x}'|\mathbf{x}, a)$ and the Bellman update equation for v_{k+1} are computed by MDP-ProbLog as defined in Equation (4) and (5), respectively.

The procedure to calculate the $\in$-optimal policy $\pi_\in$ in MDP-ProbLog involves (i) the translation of each state fluent as a Boolean fact, and the composition of actions into a fact with annotated disjunctions to ensure only one action is considered at a time, (ii) the introduction of the computed discounted value $\lambda v_k(\mathbf{x})$ into the MDP-ProbLog program as the utility value of each state $\mathbf{x} \in \mathbf{X}$, (iii) the identification of the ground atoms and rules relevant for answering probabilistic queries with respect to the atoms with utility assignments, (iv) the compilation of those atoms and rules into ProbLog for probability querying, and (v) the calculation of the optimal policy with the value iteration algorithm, implemented within MDP-ProbLog.

Language Syntax

MDP-ProbLog reserves special logic atoms to declare state fluents, actions, and utilities. A state variable is introduced through the unary atom

state_fluent/1 whose parameter is the identifier of the state variable.[5] In a similar form, each action is declared through the unary atom action/1. Utilities are assigned with the binary atom utility/2 that requires both, the identifier of the atom that will be rewarded or penalized, and its utility value. Factored transition rules and rules in the reward model are written as any other probabilistic rule in ProbLog.

Methodology

Software Architecture

The architecture of the self-driving car is depicted in Figure 1. It is composed of three main modules: the perception module, the control module, and the behavior selection module. The perception module performs visual lane detection, detection of other vehicles around the self-driving car, crash detection, distance estimation to the vehicle in the front of the self-driving car, and the identification of the lane of the road the self-driving car is driving on. The data provided by the perceptual module are continuously transmitted to the control and behavior selection modules. The control module performs lane tracking and all driving behaviors. The behavior selection module chooses the actions the control module should carry out. Note that the self-driving car does not have prior knowledge of either the initial positions or velocities of the obstacle vehicles. Instead, its decision-making module relies on basic perception capabilities (specifically, it only requires Boolean values indicating the presence or absence of the obstacles). The three modules are detailed in

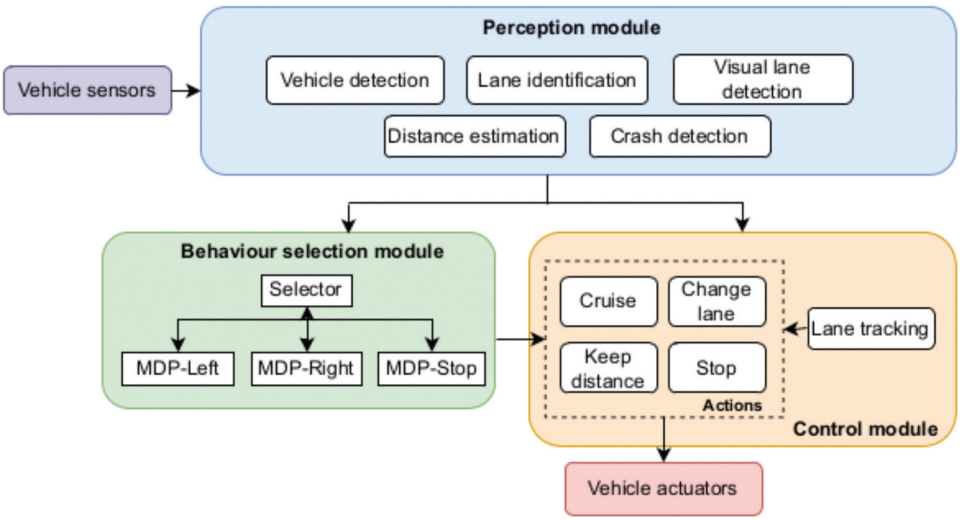


Figure 1. Software architecture of the self-driving car.

the next sections. The architecture was implemented on ROS Noetic, OpenCV 4.2.0, Webots R2022a, Ubuntu 20.04, and programmed in Python 3.

Perception Module

The self-driving car has an onboard RGB camera for lane detection, a 3D-Lidar to detect the presence of other vehicles, and an accelerometer for crash detection. RGB camera and 3D-Lidar include additive noise with a standard deviation $\sigma = 0.02$.

Visual analysis for lane detection is made with the following steps: (a) we crop the image to keep only the part below the horizon, and then we apply a triangular RoI to discard regions where lanes are not likely to appear, (b) a Gaussian filter is applied and borders are detected by Canny edge detection, (c) Hough transform is used to find lanes in normal form (ρ, θ) considering the bottom-center of the image (car hood) as the origin, (d) lines that are almost vertical or horizontal are discarded and the rest is divided into left and right borders, and finally, (e) lines are averaged by weighting their corresponding lengths, and the results are taken as the borders of the current lane. Figure 2 shows the intermediate and final results of the previous steps on lane detection.

Vehicle detection is performed in predetermined locations around the self-driving car. Figure 3 shows the locations and their labels in each lane using cardinal and ordinal directions with respect to the center of the road (hereinafter, we will abbreviate cardinal and ordinal labels by their initials N, W, E,

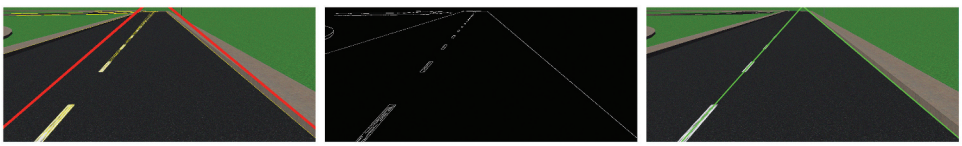
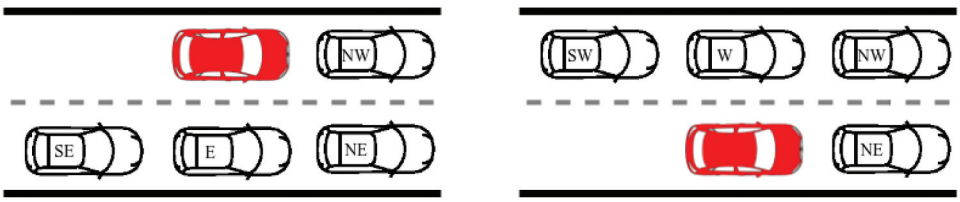


Figure 2. Lane detection process. Left: region of interest. Center: canny edge detection. Right: detected lanes by Hough Transform.



(a) Left lane (b) Right lane

Figure 3. Preestablished locations and their labels around the self-driving car (in red) where it is expected to see other vehicles (outlined) on each lane.

and NW, SW, NE, SE, respectively). A 3D LiDAR sensor mounted on the top of the self-driving car retrieves a point cloud of *xyz* readings in a three-dimensional space centered on the roof of the self-driving car. Readings hitting the ground or above some height threshold, or falling inside the bounding box of the self-driving car, are considered spurious data and are removed. The presence or absence of the surrounding vehicles is decided by counting the number of points that lie inside a rectangular cuboid valid for each location. Distance estimation is implemented to keep an appropriate distance to the vehicle in front of the self-driving car. A crash alarm is triggered if the next rule holds: $d_a = \sqrt{(a_{x,t} - a_{x,t-1})^2 + (a_{y,t} - a_{y,t-1})^2 + (a_{z,t} - a_{z,t-1})^2} \geq \Delta_a$, where d_a represent the Euclidean distance between two consecutive accelerometer readings $(a_{x,t-1}, a_{y,t-1}, a_{z,t-1})$ and $(a_{x,t}, a_{y,t}, a_{z,t})$, each component in m/s^2 units. The threshold Δ_a, which is set at 400 m/s^2, was obtained by experimentation. The identification of the lane in which the car is located is determined by comparing the current (x, y, z) coordinates of the self-driving car in the global coordinate system of the simulated environment, given that the coordinates of the center of the road are known.

Control Module

The control module is responsible for the execution of four driving actions: cruise, keep_distance, change_lane, and stop. Lane tracking is at the core of cruise and keep_distance, so it is going to be presented first. Lane tracking keeps the car on track and it is performed by comparing an observed lane (ρ_o, θ_o) and a desired lane (ρ_d, θ_d). An error is calculated for left and right lanes and speed-steering signals are calculated depending on the executed behavior as explained later. Figure 4 shows these quantities. Green lines depict the desired lines that should be observed if the car is well centered in the lane. Cyan lines represent the lines actually observed. Linear speed and steering are calculated depending on the executed action as follows:

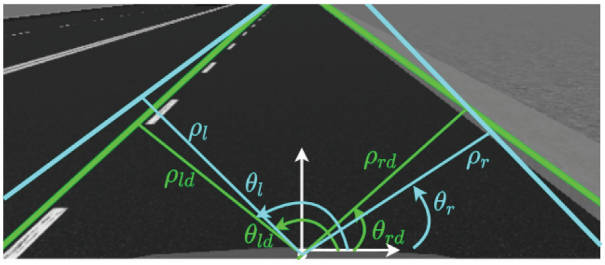


Figure 4. Parameters of the desired (green) and observed (cyan) lines described in normal form (ρ, θ).

Cruise

This action consists of keeping the car in the current lane assuming there is no obstacle. We use the following control law to track the lane: $\delta = K_\rho e_\rho + K_\theta e_\theta$ where δ is the steering angle of the self-driving car, K_ρ, K_θ are tuning constants and e_ρ and e_θ are the errors between the parameters of the observed and desired parameters of the lines. These errors are calculated as $e_\rho = \left((\rho_{ld} - \rho_l + \rho_{rd} - \rho_r)/2\right)$ and $e_\theta = ((\theta_{ld} - \theta_l + \theta_{rd} - \theta_r)/2)$. In this action, linear speed is constant $v = K_v$.

Keep Distance

The goal of this action is to keep the car at a desired distance from the car in front (North position). Lane tracking is still performed to keep the car in the current lane. In this action, steering is calculated similar to the cruise action. Linear speed is calculated using a proportional control:

$$v = \begin{cases} v_c + K_d(d - d_d) & \text{if} \quad d < d_d \\ v_c \quad \text{otherwise} \end{cases} \tag{7}$$

where d is the distance between the self-driving car and the car in front of it and d_d is the desired distance. K_d is a tuning constant.

Change Lane

This action is intended to be used for passing cars. Change can be made to left of right, depending on the current lane (a two-lane way is assumed). We calculate steering as: $\delta = \arcsin(\frac{\dot{\theta}L}{v})$ where v is the current car linear speed, $\omega = \dot{\theta}$ is a desired angular speed and L is the distance between rear and front tires. Quotient $\dot{\theta}L/v$ must be in [−1,1] for δ to exist, but this is easily achieved if ω is set according to the car's physical capabilities. For example, if the car is stopped ($v = 0$), it is not possible to turn left or right with any steering value. In this work, we set $\omega = 1.2$ [rad/s] and $L = 2.9$ m. Linear speed v is the current speed and can vary depending on the previous executed action.

Lane changing is achieved with two consecutive turns. For change to the left, the car turns first to left until reaching a given position threshold y_1, and then it turns right until being aligned to the left lane. Figure 5 shows the two movements to perform a change-lane to the left. Turning to the right uses the same logic.

Stop

This action sets both speed and steering to zero.

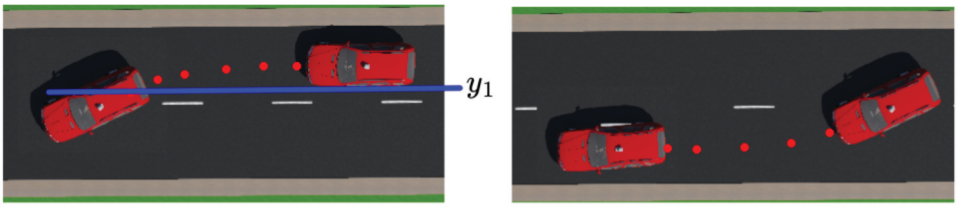


Figure 5. Sequence of movements for lane changing.

Behavior Selection Module

Design of PL-fMdps

Before discussing the PL-fMDPs designed for this work, the state fluents that model the state-space of the system are presented. These state fluents are as follows: success, right_lane, free_NE, free_E, free_SE, free_NW, free_W, free_SW. State fluent success is true if the last action did not lead to a collision, and it is false otherwise. State fluent right_lane, is true if the self-driving car is in the right lane, and it is false if the car is in the left lane. Each one of the other state fluents is true if the location referred by its name is free (no vehicle present), and it is false, otherwise.

The first PL-fMDP, called "MDP-Selector", is aimed to derive an action policy to decide which one of the other three specialized policies must be triggered at each moment. Actions select_left_policy, select_right_policy, and select_stop_policy are used for this purpose. The second PL-fMDP, called "MDP-Left", was designed to generate a driving policy that uses the left lane for passing maneuvers mostly. The third PL-fMDP is called "MDP-Right," and its purpose is to produce a policy that privileges navigation through the right lane. Lastly, the fourth PL-fMDP called "MDP-Stop" was designed to explore a policy for bringing the car to a complete stop in the event of a car crash, or to take no action otherwise. Table 1 lists the state fluents and actions used on each one of those PL-fMDPs.

Due to space constraints, we present only some of the blocks of code that we consider relevant to give a flavor of the construction of a PL-fMDP program.

Table 1. PL-fMdps and their state fluents and actions.

PL-fMDP	State fluents	Actions
MDP-Selector	success, right_lane	select_left_policy select_right_policy select_stop_policy
MDP-Left	free_NW, free_NE, free_E, free_SE	cruise, keep_distance change_lane (to right)
MDP-Right	free_NE, free_NW free_W, free_SW	cruise, keep_distance change_lane (to left)
MDP-Stop	success	stop do_nothing

For example, two factored transition rules for the action change_lane in MDP-Left are:

```
% Action: change_lane
0.99::free_NE(1)    :- free_NE(0), free_E(0),
                       (free_SE(0);\+free_SE(0)), change_lane.
0.01::free_NE(1)    :- (\+free_NE(0);\+free_E(0)), change_lane.
0.99::free_NW(1)    :- free_NW(0), change_lane.
0.01::free_NW(1)    :-\+free_NW(0), change_lane.
0.99::free_E(1)     :- free_E(0), change_lane.
0.01::free_E(1)     :-\+free_E(0), change_lane.
0.99::free_SE(1)    :- free_SE(0), change_lane.
0.01::free_SE(1)    :-\+free_SE(0), change_lane.
```

The first rule indicates that when traveling in the left lane, free_NE will most likely be true in the immediate future with a high probability value of 0.99 if both free_NE (which refers to the space ahead in the right lane) and free_E (referring to the space to the right in the right lane) are currently true, and the action change_lane is performed. The rule also explicitly specifies that the presence or absence of an obstacle vehicle in space SE is not relevant for this maneuver. This rule is based on the implicit assumption that the self-driving car consistently navigates at higher speeds than the other vehicles, thus ensuring there is sufficient room for a safe turn. The second rule indicates that free_NW will be true with a low probability value of 0.01 after changing to the right lane, whenever either free_NE or free_E, or both are currently false (there is no enough free space for the maneuver, and trying to execute it will almost surely produce a collision, in accordance with the reward model below). The rest of the rules are included to reflect the assumption that the truth value of free_NW, free_E and free_SE are not directly influenced by change_lane (if each one of the state fluents are true, they will almost certainly remain true, and if they are false, it is unlikely that they will become true). These rules emulate the description of Spudd (Hoey et al. 1999), an early influential software package for coding fMDPs using *algebraic decision diagrams*, and where it is customary to explicitly specify which state variables are not affected by the execution of an action.

The utility assignments and the rules of the reward model in the program MDP-Left are:

```
% Utilities
utility(free_NE(0), 1.0).
utility(free_NW(0), 1.0).
utility(rearEnd_crash, −4.0).
utility(sideSwipe_crash, −2.0).
```

```
utility(keep_distance, −1.0).
% Reward model
0.99::rearEnd_crash    :-\+ free_NW(0), cruise.
0.99::rearEnd_crash    :-\+free_NE(0), change_lane.
0.95::sideSwipe_crash :-\+free_E(0), change_lane.
```

In these rules, a positive utility is assigned to the state fluents free_NE and free_NW whenever the self-driving car has its NE or NW free. Negative utilities are attached to two complementary atoms called rear_crash and sideSwipe_crash, which represent the occurrence of rear-end and sideswipe collisions, respectively. The difference in their penalization reflects an assumed variation in severity among these types of accidents within our setting. The last three rules stipulate how likely each one of these collisions will happen if the corresponding conditions are met. Finally, utility is also negative for keep_distance, because this action may lead to an undesirable delay in reaching the destination.

Action Policies and the Behavior Selection Module

The action policy obtained from the program MDP-Left is called simply as "Left" and it is presented in Table 2. Since MDP-Left has four binary state fluents, the policy Left comprises $2^4 = 16$ deterministic state-action associations. For example, states 1–6 say that keep_distance is the best action when there is a vehicle at NW and either, at E or at NE, or both (actions change_lane and cruise are not possible), independently on the presence of a car at SE location. The cruise action is chosen (states 7–12) whenever NW is free, but there are vehicles at E and NE that impede a safe turn. Similarly, states 13–16 select change_lane if E and NE are free. The main difference between the policies Left and "Right" obtained from the program MDP-Right is that the latter is more

Table 2. Action policy left.

	State fluents				
State #	free_E	free_NE	free_NW	free_SE	Action
1	0	0	0	0	keep_distance
2	1	0	0	0	keep_distance
3	0	1	0	0	keep_distance
4	0	0	0	1	keep_distance
5	1	0	0	1	keep_distance
6	0	1	0	1	keep_distance
7	0	0	1	0	cruise
8	1	0	1	0	cruise
9	0	1	1	0	cruise
10	0	0	1	1	cruise
11	1	0	1	1	cruise
12	0	1	1	1	cruise
13	1	1	0	0	change_lane
14	1	1	1	0	change_lane
15	1	1	0	1	change_lane
16	1	1	1	1	change_lane

conservative when it comes to changing to the left lane (as stated above, policy Left is for overtaking only, and then it favors to return to the right lane as soon as possible). Policy Right only allows changing to the left lane in one state, when NE is occupied but W, NW, and SW are free. When in the right lane, vehicles approaching from SW must be considered, as left-lane vehicles usually move faster, and switching lanes could be risky. Policy "Selector" has four state-action associations. If success is true, Selector chooses the policy appropriate for the current lane; otherwise, the policy "Stop" is chosen. This latter policy is the simplest, involving only two state-action associations: one for stopping the car if success is false, and the second action is to do nothing otherwise.

To implement the behavior selection module, the policies were arranged in a two-level hierarchical structure, as depicted in Figure 1. Only three of the policies are active at every moment: Selector, Stop, and either Left or Right, in accordance with the position of the car. Recall from Table 1 the number of state fluents on each PL-fMDP. This task decomposition results in $2^2 + 2^1 + 2^4 + 2^4 = 38$ states, in comparison to the $2^8 = 256$ states that are required for a single PL-fMDP[6] with the same number of fluent variables. In our experience, this reduction in the number of states sped up the design, testing, and debugging of the PL-fMDPs.

Convergence Criteria of the PL-fMdps

Figure 6 shows two convergence plots derived from the solution of the PL-fMDPs. The plot on the left showcases $\max_{\mathbf{x}\in\mathbf{X}} |v_{k+1}(\mathbf{x}) - v_k(\mathbf{x})|$, representing the maximum error observed for any state $\mathbf{x} \in \mathbf{X}$ between the kth and $(k+1)$st iterations of the value iteration algorithm when solving each PL-fMDP. Similarly, the plot on the right illustrates the evolution of $\max_{\mathbf{x}\in\mathbf{X}} v(\mathbf{x})$ through iterations, indicating the maximum value $v(\mathbf{x})$ obtained for any state $\mathbf{x} \in \mathbf{X}$ in each iteration. The maximum number of iterations observed was 50, although in all cases the policies do not change after a few iterations. Policies were obtained by setting $\epsilon = 0.1$ and $\gamma = 0.9$, the default values of MDP-ProbLog for these parameters. Therefore, the error threshold is approximately $0.005\overline{5}$.

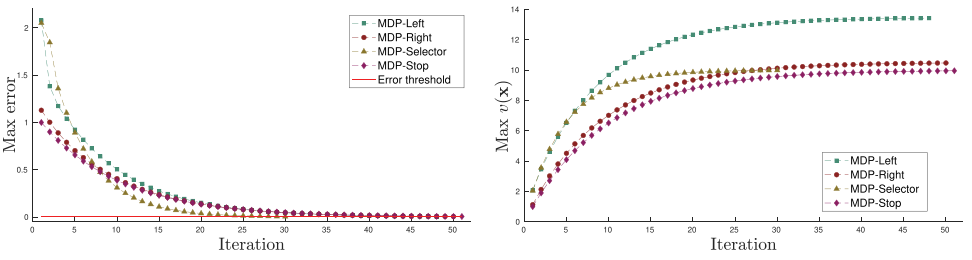


Figure 6. Convergence graphs for the solution of the PL-fMdps using value iteration: the maximum error for each on each iteration (left), and the maximum state value (right).

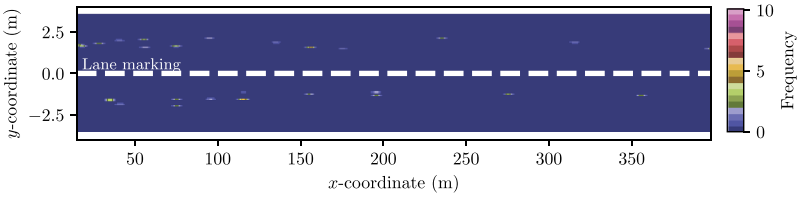


Figure 7. Heat map of the *xy*-coordinates (in meters) of the obstacle vehicles at their initial positions in all the tests.

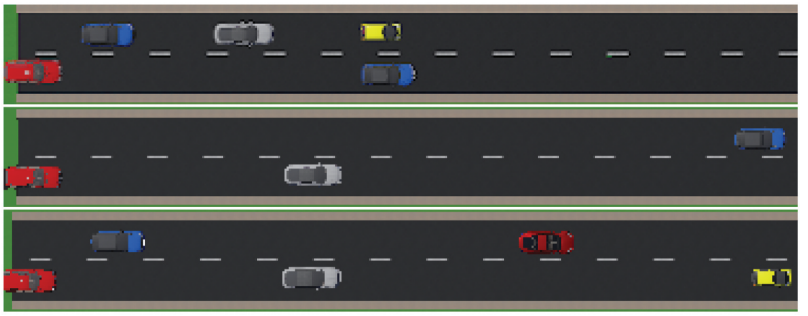


Figure 8. Overhead view of initial positions of the obstacle vehicles in tests 1, 2, and 3 (shown from top to bottom, respectively). The self-driving car painted in red, is always at the bottom left.

Evaluation and Results

Three primary tests were conducted to assess the effectiveness of the overall system and the behavior selection module. In the first test, the appropriateness of the actions that the self-driving car must choose is evaluated in 16 driving scenarios. In the second test, the self-driving car must overtake stationary obstacle vehicles. In the third test, the self-driving car must overtake moving obstacle vehicles that travel at different speeds. In the latter two tests, the goal is to measure the number of times the self-driving car overtakes the obstacle vehicles without colliding with them. A factorial design is used to classify the repetitions of those tests by varying the speed of the self-driving car and the number of obstacle vehicles. In all the tests, before engines start, each obstacle vehicle is randomly translated forward or backward (x-axis), by up to ± 2 meters from its original location. Figure 7 illustrates the xy-coordinates of the initial positions assigned to the obstacle vehicles in all tests after randomizing the x-coordinate. Obstacle vehicles were intentionally not perfectly centered in each lane. Figure 8 depicts top views of the initial positions of the cars in these tests, which are detailed below.

Test 1: Selection of Behaviors in Specific Scenarios

The goal of this experiment is to evaluate the ability of the self-driving car to select the most appropriate action for the current driving scenario. The 16

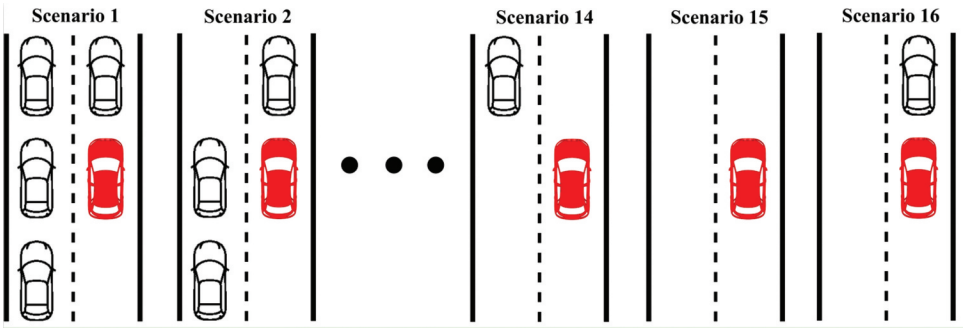


Figure 9. Examples of some of the 16 scenarios derived from policy right and designed for test 1.

scenarios that were tested are derived from the states of the policy Right. In all the scenarios, the self-driving car initiates at the beginning of the right lane with 0 to 4 moving obstacle vehicles distributed ahead in both, the right and left, lanes. Figure 9 shows some of the 16 scenarios in this test, considering the presence or absence of four obstacle vehicles and the self-driving car. The minimum distance between the self-driving car and the obstacle vehicles was 6 m. The maximum linear velocity of the self-driving car was set to 28 km per hour. Obstacle vehicles travel at a speed of 18.8 km per hour. The test ended when either the correct decision was made according to the policy or when there was a collision with another vehicle.

Test 2: Overtaking Stationary Vehicles

In this test, all obstacle vehicles were lined up ahead of the self-driving car and interleaved in the two lanes, maintaining a distance of approximately 40 m between each vehicle. This distance was visually estimated to ensure that the self-driving car had enough space for passing maneuvers. The self-driving car consistently started behind the other vehicles, occupying the last position in the right lane. Two independent variables were defined: (i) the maximum linear velocity of the self-driving car, set at 20, 24, or 28 km per hour, and (ii) the number of obstacle vehicles on the road, either 5 or 10. Environments with five obstacle vehicles comprised two vehicles in the left lane and three vehicles in the right lane. Environments with 10 obstacle vehicles followed a similar distribution pattern, with half of the vehicles in the left lane and half in the right lane.

Test 3: Overtaking Moving Vehicles

Test 3 is similar to Test 2, with the distinction that obstacle vehicles were now in motion. Vehicles in the left lane navigated at a speed of 18.8 km per hour, while those in the right lane traveled at a speed of 14.4 km per hour. To account for the

speed difference between the left and right lanes, obstacle vehicles in the left lane were significantly moved backward, as illustrated in the bottom image of Figure 8. Due to this reallocation, the initial distances between obstacle vehicles were reduced to approximately 17 m compared to Test 2.

Results and Discussion

The three tests were executed on standard laptop computers with Intel Core i7 processors 8GB of RAM and standard integrated graphics cards. We completed 10 repetitions for each driving scenario of Test 1 (160 repetitions in total). Results are presented in Table 3. In all cases, the selection of actions was accurate with respect to the policy Right. For Test 2 and Test 3, we conducted 30 repetitions for each of the 12 possible combinations of self-driving car speeds and the number of obstacle vehicles, resulting in a total of 360 repetitions. Table 4 summarizes the results. The self-driving car successfully overtook the static obstacle vehicles in all repetitions. When obstacle vehicles move, the self-driving car overtook them without collisions in 97.7% of the trials. In 2.3% of the remaining repetitions, the self-driving car collided with other vehicles, particularly when changing lanes.

These crashes could be attributed to factors such as the permissibility of the policy Left for lane changing, failures in detecting neighboring vehicles, or the limitations in the capacity of the laptop computers to perform the required physics calculations, as continuously warned by the simulator. Although we believe these are plausible explanations, it is also necessary to add more information to the system, such as an estimate of the speed and distance of the surrounding vehicles. The overall weighted average with respect to the number of repetitions in each test is 99.2%. We are carefully analyzing log files to improve the

Table 3. Results of test 1.

	Chosen action			
Scenario #	Keep_distance	Cruise	Change_lane	Expected action
1	100%			keep_distance
2	100%			keep_distance
3	100%			keep_distance
4	100%			keep_distance
5	100%			keep_distance
6	100%			keep_distance
7	100%			keep_distance
8		100%		cruise
9		100%		cruise
10		100%		cruise
11		100%		cruise
12		100%		cruise
13		100%		cruise
14		100%		cruise
15		100%		cruise
16			100%	change_lane

Table 4. Percentage results as a function of the number of obstacle vehicles and speed of the self-driving car for both tests 2 and 3, based on 360 repetitions of both tests.

	# of Obstacle Vehicles	Self-driving car speed		
		20 km/h	24 km/h	28 km/h
Static obstacle vehicles	5	100%	100%	100%
	10	100%	100%	100%
Moving obstacle vehicles	5	100%	100%	90%
	10	96.6%	100%	100%

change_lane maneuver and decrease the possibility of this type of collision in more complex driving scenarios. In total, the self-driving vehicle covered more than 340 km and took 618372 driving decisions. The trajectories drawn by the self-driving car in the second and third tests are depicted in Figure 10. In these plots, it is shown that the traveling paths were consistent through the repetitions. Moreover, while lane changing can be clearly observed when avoiding fixed obstacles, it should be noted that overtaking the moving cars in the left lane (which travel at a faster speed than those in the right lane) requires covering a greater distance.

Although the selection of the actions strongly depends on the effectiveness of the perceptual and control modules in the tests, we did not evaluate their output, as it is not the main focus of this work. Instead, we considered the self-driving car as a whole, integrated system.

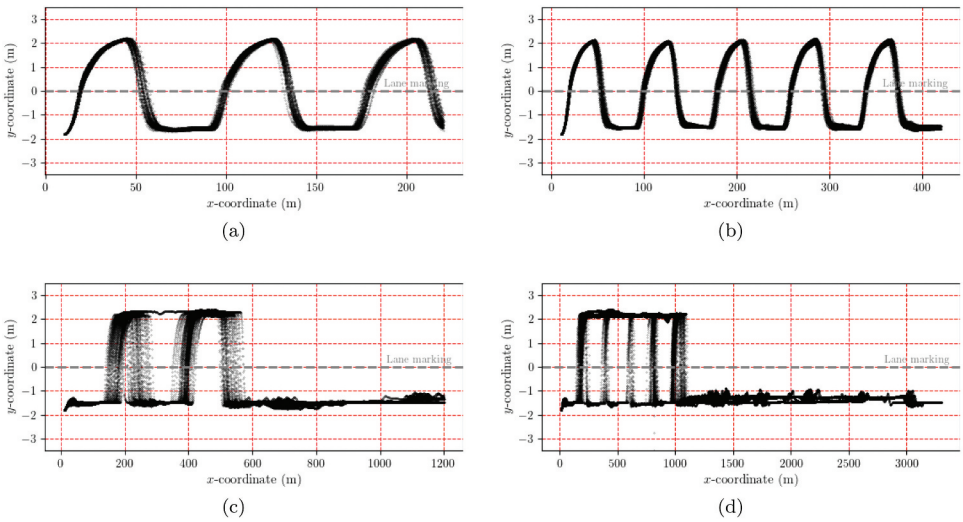


Figure 10. Trajectories of self-driving car in test 2 and test 3: (a) and (b) depict the self-driving car overtaking 5 and 10 stationary vehicles, while (c) and (d) illustrate the overtaking of 5 and 10 dynamic vehicles, respectively.

Conclusions and Future Work

We presented a strategy based on probabilistic logic representation of factored Markov decision process for behavior selection in self-driving cars. The behavior selection module is composed of a two-level hierarchy of four action policies obtained from the same number of PL-fMDPs. A PL-fMDP provides a rule-based description of a factored Markov decision process that is easy to comprehend and communicate, simplifying its design and coding. It is well known in the literature that decomposing the decision-making task can considerably reduce the exponential growth of the state space in a single fMDP with respect to the number of state variables. This reduction implies less human effort and computational resources to debug and inspect the resulting policies. An additional result of this work is a self-driving car software architecture that is freely available on the Web. Evaluation results show that this architecture provides reliable navigation in the proposed scenarios. We consider it can be used to develop and test new perceptual and control capabilities, as well as other driving behaviors or behavior selection methods.

As future work, we plan to use the data recorded in the log files generated from our testing to refine the state transition function of the PL-fMDPs. In addition, we plan to extend MDP-ProbLog with annotated disjunctions, thereby introducing state fluents with multiple values. We believe this extension will simplify the incorporation of more elaborated non-binary information into the PL-fMDPs, such as speed levels of the surrounding vehicles, the presence of road signs, or the state of the traffic lights. Also, we will test our approach in more complex scenarios, such as the Simulation of Urban Mobility (SUMO) platform and the Autominy car V4 developed by the Free University of Berlin. Furthermore, we will explore the development of causal models with a focus on explaining driving decisions and identifying circumstances that may lead our self-driving car to a collision.

Notes

1. An atomic formula (or *atom*, for short) has the form "$a(t_1, \ldots, t_n)$," for $n \geq 0$, where a is the identifier of the atom, and each argument t_i for $i = 1$ to n is a *term* (that is, a variable, a constant, or a compound term). An atom is *grounded* when none of its arguments are variables or when they do not contain variables.
2. Iverson bracket function $[\![\cdot]\!]$ evaluates to 1 if the propositional condition enclosed in the brackets holds, and it evaluates to 0 otherwise.
3. In this document, prime notation is employed to differentiate between *post-action* state variables $X'_1, .., X'_n$ and *pre-action* state variables $X_1, .., X_n$.

4. Lowercase letters are used to denote state fluents, rather than uppercase letters for state variables as in the previous section, in order to adhere to the standard definition of atoms in Prolog.
5. Wherever used after its introduction in an MDP-ProbLog program, a pre-action (resp. post-action) state fluent is identified by adding a value 0 (resp. 1) as its first parameter.
6. Notice that free_NE, free_NW, and success are used twice in the hierarchy, so they are counted only once for the single PL-fMDP.

Acknowledgements

The authors would like to thank Sergio Yahir Hernandez-Mendoza for his generous support in conducting part of the tests in this work, and the reviewers for their insightful and interesting comments and feedback.

Data availability statement

The source code of this work (that includes the simulated environment, the self-driving system, and the four PL-fMDPs), and a video recording showing a run of the system are freely available at: https://github.com/mnegretev/AutonomousBehaviorSelection2023.

Disclosure statement

No potential conflict of interest was reported by the author(S).

Funding

This work was partially supported by UNAM-DGAPA under grant TA101222 and AI Consortium - CIMAT-CONAHCYT.

References

Aksjonov, A., and V. Kyrki. 2022. A safety-critical decision-making and control framework combining machine-learning-based and rule-based algorithms. *SAE International Journal of Vehicle Dynamics, Stability, and NVH* 7 (3). *arXiv preprint arXiv:2201.12819*. doi:10.4271/10-07-03-0018.

Al-Nuaimi, M., S. Wibowo, H. Qu, J. Aitken, and S. Veres. 2021, SEP. Hybrid verification technique for decision-making of self-driving vehicles. *Journal of Sensor & Actuator Networks* 10(3):42. doi:10.3390/jsan10030042.

Avilés-Arriaga, H. H., L. E. Sucar, E. F. Morales, B. A. Vargas, J. Sánchez, and E. Corona. 2009. Markovito: a flexible and general service robot. *Design and Control of Intelligent Robotic Systems* 177:401–23.

Avilés, H., M. Negrete, R. Machucho, K. Rivera, D. Trejo, and H. Vargas. 2022. Probabilistic logic Markov decision processes for modeling driving behaviors in self-driving cars. *Ibero-American Conference on Artificial Intelligence (IBERAMIA)*, 366–77, Cartagena de Indias, Colombia: Springer. Springer.

Boutilier, C., R. Dearden, and M. Goldszmidt. 2000 August. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1–2):49–107. doi:10.1016/S0004-3702(00)00033-3.

Bueno, T. P., D. D. Mauá, L. N. De Barros, and F. G. Cozman. 2016. Markov decision processes specified by probabilistic logic programming: representation and solution. *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, 337–42, Recife, Brazil: IEEE.

Cai, Z., Q. Fan, R. S. Feris, and N. Vasconcelos. 2016. A unified multi-scale deep convolutional neural network for fast object detection. *Computer Vision–ECCV 2016: 14th European Conference*, Amsterdam, The Netherlands, October 11–14, 2016, *Proceedings, Part IV 14*, 354–70, Springer.

Chavira, M., and A. Darwiche. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172 (6–7):772–99. doi:10.1016/j.artint.2007.11.002.

Clark, K. L. 1978. Negation as failure. In *Logic and data bases*, ed., H. Gallaire and J. Minker. Boston, MA: Springer. doi: 10.1007/978-1-4684-3384-5_11.

Da Lio, M., A. Cherubini, G. P. R. Papini, and A. Plebe. 2023. Complex self-driving behaviors emerging from affordance competition in layered control architectures. *Cognitive Systems Research* 79:4–14. doi:10.1016/j.cogsys.2022.12.007.

Dortmans, E., and T. Punter. 2022. Behavior trees for smart robots practical guidelines for robot software development. *Journal of Robotics* 2022 (7):7419–30. doi:10.1155/2022/3314084.

Duan, J., S. Eben Li, Y. Guan, Q. Sun, and B. Cheng. 2020. Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data. *IET Intelligent Transport Systems* 14 (5):297–305. doi:10.1049/iet-its.2019.0317.

Fierens, D., G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt. 2015. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming* 15 (3):358–401. doi:10.1017/S1471068414000076.

Hoey, J., R. St-Aubin, A. Hu, and C. Boutilier. 1999. SPUDD: Stochastic planning using decision diagrams. *Proceedings of International Conference on Uncertainty in Artificial Intelligence (UAI '99)*, Stockholm, Sweden.

Lindqvist, B., S. S. Mansouri, A.-A. Agha-Mohammadi, and G. Nikolakopoulos. 2020. Nonlinear mpc for collision avoidance and control of UAVs with dynamic obstacles. *IEEE Robotics and Automation Letters* 5 (4):6001–08. doi:10.1109/LRA.2020.3010730.

Liu, Q., X. Li, S. Yuan, and Z. Li. 2021. Decision-making technology for autonomous vehicles: Learning-based methods, applications and future outlook. *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 30–37, Indianapolis, IN, USA. IEEE.

Mishra, P. 2022. Model explainability for rule-based expert systems. In *Practical explainable AI using python*, ed. P. Mishra, 315–26. Berkeley, CA: Apress. doi:10.1007/978-1-4842-7158-2_13.

Mu, Z., W. Fang, S. Zhu, T. Jin, W. Song, X. Xi, Q. Huang, J. Gu, and S. Yuan. 2022. A multi-modal behavior planning framework for guide robot. *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Jinghong, China.

Padalkar, P., H. Wang, and G. Gupta. 2023. Nesyfold: A system for generating logic-based explanations from convolutional neural networks. *arXiv preprint arXiv:2301.12667*.

Paden, B., M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles* 1 (1):33–55. doi:10.1109/TIV.2016.2578706.

Pepe, G., M. Laurenza, D. Antonelli, and A. Carcaterra. 2019. A new optimal control of obstacle avoidance for safer autonomous driving. *2019 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, 1–6, Turin, Italy.

Precup, R.-E., S. Preitl, J. Tar, M. Tomescu, M. Takacs, P. Korondi, and P. Baranyi, 2008 9. Fuzzy control system performance enhancement by iterative learning control. *IEEE Transactions on Industrial Electronics* 55(9):3461–75. Accessed November 27, 2023. doi:10.1109/TIE.2008.925322.

Puterman, M. L. 1994. *Markov decision processes: Discrete stochastic dynamic programming*. Hoboken, NJ, US: John Wiley & Sons.

Reyes, A., L. E. Sucar, P. H. Ibargüengoytia, and E. F. Morales. 2020, May. Planning under uncertainty applications in power plants using factored Markov decision processes. *Energies* 13(9):2302. doi:10.3390/en13092302.

Rigatos, G., P. Siano, D. Selisteanu, and R. E. Precup. 2016, Nonlinear optimal control of oxygen and carbon dioxide levels in blood. *Intelligent Industrial Systems* 3 (2):61–75. Accessed November 27, 2023. doi:10.1007/s40903-016-0060-y.

Riguzzi, F. 2022. *Foundations of probabilistic logic programming: Languages, semantics, inference and learning*. Denmark: River Publishers.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. *Proceedings of the 12th international conference on logic programming (ICLP'95)*, 715–29, Citeseer.

Smith, G., R. Petrick, and V. Belle. 2021. Intent recognition in smart homes with problog. *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, Kassel, Germany.

Spanogiannopoulos, S., Y. Zweiri, and L. Seneviratne. 2022. Sampling-based non-holonomic path generation for self-driving cars. *Journal of Intelligent & Robotic Systems* 104(1). doi:10.1007/s10846-021-01440-z.

Sun, Z., J. Zou, D. He, Z. Man, and J. Zheng. 2020. Collision-avoidance steering control for autonomous vehicles using neural network-based adaptive integral terminal sliding mode. *Journal of Intelligent & Fuzzy Systems* 39 (3):4689–702. doi:10.3233/JIFS-200625.

Unguritu, M.-G., and T.-C. Nichitelea. 2023 Design and assessment of an anti-lock braking system controller. *Romanian Journal of Information Science and Technology* 2023 (1):21–32. Online; Accessed November 27, 2023. doi:10.59277/ROMJIST.2023.1.02.

Van Gelder, A., K. A. Ross, and J. S. Schlipf. 1991. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)* 38 (3):619–49. doi:10.1145/116825.116838.

Vennekens, J., S. Verbaeten, and M. Bruynooghe. 2004. Logic programs with annotated disjunctions. In *International Conference on Logic Programming*, Saint-Malo, France, 431–45. Springer.

Wang, K., C. Mu, Z. Ni, and D. Liu. 2023. Safe reinforcement learning and adaptive optimal control with applications to obstacle avoidance problem. *IEEE Transactions on Automation Science and Engineering*, doi:10.1109/TASE.2023.3299275.

Yuan, M., J. Shan, and K. Mi. 2022. Deep reinforcement learning based game-theoretic decision-making for autonomous vehicles. *IEEE Robotics and Automation Letters* 7 (2):818–25. doi:10.1109/LRA.2021.3134249.

Zhang, X., A. Liniger, and F. Borrelli. 2021. Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology* 29 (3):972–83. doi:10.1109/TCST.2019.2949540.

Zhang, Y., P. Tiňo, A. Leonardis, and K. Tang. 2021. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence* 5 (5):726–42. doi:10.1109/TETCI.2021.3100641.

Zhu, W., Y. Lu, Y. Zhang, X. Wei, and Z. Wei. 2022. End-to-end driving model based on deep learning and attention mechanism. *Journal of Intelligent & Fuzzy Systems* 42 (4):3337–48. doi:10.3233/JIFS-211206.