Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Decision making for autonomous vehicles: Combining safety and optimality

J.J. Verbakel [*,1] **M. Fusco** [**,1] **D.M.C. Willemsen** [***,1]
J.M. van de Mortel-Fronczak [*] **W.P.M.H. Heemels** [*]

[*] *Eindhoven University of Technology, Eindhoven, The Netherlands.*
[**] *The MathWorks B.V., 5652 XR Eindhoven, The Netherlands.*
[***] *Integrated Vehicle Safety Department, TNO, 5708 JZ Helmond, The Netherlands.*

Abstract: In this paper, we propose a novel decision maker design for an autonomous vehicle driving on a highway, considering safety and optimality, and which is scalable, i.e., remains computationally tractable for more complex situations. This is realized in two stages. First, all safe actions are found, and second, from these actions the optimal action is selected, according to (weighted) criteria that capture safety, comfort and efficiency. The design combines rule-based safety checks with the solution of a Markov decision process, found through a tree search algorithm, to fulfill the safe, smart and scalable requirements of the decision maker. The design is validated in simulation using eight different scenarios. The performance of the new design is compared to the performance of a rule-based controller. This comparison is done using three performance criteria that aim to capture safety, efficiency and comfort.

Keywords: Autonomous Vehicles; Mission planning and decision making; Multi-vehicle systems

1. INTRODUCTION

As cars become more intelligent, automated vehicles (AVs) come closer to being commercially available. When going to higher levels of automation, the AV controller needs to make increasingly more decisions on how and where to drive. In a recent review (Schwarting et al. [2018]), a distinction is made between three different control architectures for AVs, being sequential planning, behavior-aware planning and end-to-end planning (Fig. 1). Sequential planning separates perception, decision making and path planning into three sequential steps. A perception module combines sensor inputs to increase the reliability of information regarding the environment. The decision maker (DM) uses the perception outputs to select a discrete action from a set of possible actions, such as 'lane keeping' or 'lane change to the left'. The path planner uses this action to generate a reference path or trajectory, which is executed by low-level controllers. In behavior-aware planning, the path planning is also (partially) done by the DM. If input data processing is incorporated in the decision making process as well, the controller is called an end-to-end controller. Only sequential planning methods (top row in Fig. 1) are discussed in this paper, a review of other methods can be found in Schwarting et al. [2018]. In that review, also several future issues of interest are mentioned. One of those issues is a lack of a method that provides safe performance in a complex environment while modeling the interaction with other road-users in highway scenarios. As shown in Zhou et al. [2017], highways provide a structured environment and consistent behavior of the road users on a short prediction horizon, unlike urban driving. Therefore, a more high-level DM with discrete actions is likely to be sufficient for highway scenarios, motivating a sequential planning architecture.

[1] The authors would like to thank all national funding authorities and the ECSEL Joint Undertaking, which funded the PRYSTINE project under the grant agreement number 783190.

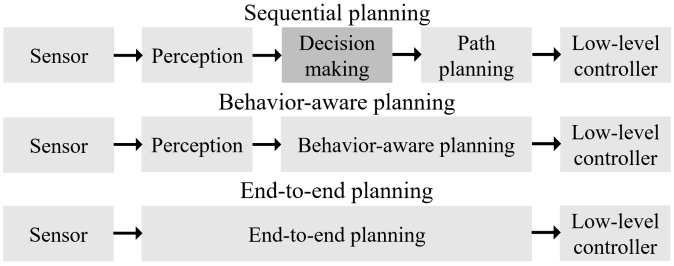

Fig. 1. Three control architectures. The highlighted block shows the component discussed in this paper.

Designing the DM using only if-then-else statements, implemented manually from the system specifications, has several drawbacks. This approach is error-prone and correct behavior is only guaranteed through exhaustive testing. To address the latter point, a rule-based DM can also be automatically generated if the requirements are expressed formally, enforcing properties as completeness with respect to the requirements. For instance, in Korssen et al. [2017], supervisor synthesis is used to create a cruise control system, including human interaction. A supervisor provides guarantees on safety, however it does not select one action if multiple safe actions exist. Here, we call selecting an optimal action *smart*. In Kim and Langari [2014], a two-player game-theory model is used for deciding to switch lanes or not. This model makes a smart decision, but safety is not considered for the decision. In Wongpiromsarn et al. [2010], linear temporal logic (LTL)

is used to navigate an AV while avoiding obstacles, this method is safe and smart, but does not scale to more vehicles. Another approach is using Markov decision processes (MDPs) to model decision making. An MDP can be solved offline, as in Abbeel and Ng [2004], Guan et al. [2018] or online as in Zhou et al. [2017]. Solving an MDP means here finding the action that maximizes the long-term (discounted) reward. To handle uncertainties in the state, a partially observable MDP is used in Liu et al. [2015]. MDP-based approaches typically are smart and scalable, but unsafe actions can still be possible, because the decision is the result of an optimization driven by a cost function, which penalizes, but does not *prohibit* unsafe actions.

Furthermore, Model Predictive Control (MPC) can be used, see, e.g., applications in low-level control [Falcone et al., 2007] or cooperative driving [Mohseni et al., 2017, Kim and Kumar, 2014]. In MPC, the optimal control inputs are computed for a (short) horizon, and the first control input is applied. The control inputs are typically obtained by solving at each (discrete) time instant a finite-horizon optimization problem based on a specific cost function, satisfying constraints and system dynamics and using the current state measurement. A disadvantage of MPC can be the computational time, see Mohseni et al. [2017], certainly when mixed-integer optimization problems have to be solved as, for instance, in Fabiani and Grammatico [2019], where an MPC motion planning with safety constraints is used in a mixed integer potential game. Because of the above reasons, an alternative method is investigated, which has links to MPC (in particular, receding horizon implementations), but strives explicitly for safety guarantees and fast computations, next to optimality.

Rule-based methods are safe but not smart, while MDP-based and similar approaches are smart but not safe. There is no general agreement about which method is best suited to design a DM. The contribution of this paper is a DM design that takes safe and smart decisions in a complex environment, including interaction with other road-users. To this end, a new design that operates in two stages is proposed. First, all safe actions are found, and second, from these actions the optimal action is selected, according to criteria that try to capture safety, comfort and efficiency. The novel contribution of this paper is a new design, which combines rule-based safety checks together with an MDP solved through a tree search algorithm. The methods are chosen to be scalable, i.e., they will still be computationally tractable for large systems.

The remainder of this paper is structured as follows. In Section 2, the problem is formalized. Section 3 introduces the two-stage DM design. Sections 4 and 5 explain each of the two stages in more detail. In Section 6 the performance is compared to the performance of a manually programmed rule-based DM. Finally, conclusions are given in Section 7.

2. PROBLEM FORMULATION

In this section, the problem is formalized. First, the input to the DM, i.e., the system state, is given, as should be available from a perception module; how these values are obtained is outside the scope of this project. Then, the output of the DM, i.e., the chosen action, is given,

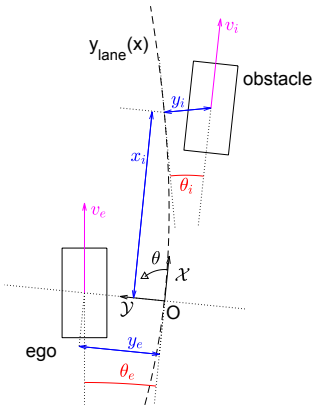

Fig. 2. The vehicle variables that are used as input to the DM. Coordinates are given in blue, angles in red and velocities in magenta. The 3rd order lane polynomial is given by $y_{lane}(x)$.

Table 1. Nine actions as a combination of lateral maneuver and speed update

	LCL	LK	LCR
accelerate	LCL_a	LK_a	LCR_a
constant speed	LCL_c	LK_c	LCR_c
decelerate	LCL_d	LK_d	LCR_d

as presented to a path planning module, see Sequential planning in Fig. 1. Last, the requirements for the DM are given, leading to the problem formulation.

2.1 System state

The vehicle variables that are used by the DM are shown in Fig. 2. The origin of the coordinate system is placed on the ego lane center, its $\mathcal{X}$ direction is tangential to the lane, $\mathcal{Y}$ is perpendicular to $\mathcal{X}$ in counterclockwise direction, such that it intersects the ego vehicle center-of-gravity. It is assumed that the road curvature is small, which holds for highway scenarios, such that the coordinate system can be assumed rectangular.

The state is divided in two parts, being road information and the states of all (ego and surrounding) vehicles. The road is modeled as a 3rd order polynomial that describes the center of the ego lane. The multi-vehicle system state $\mathbf{X}$ is a matrix containing vectors $\mathbf{x}_i$ with $i = e$ or $i \in \mathbb{N}_{[1,n_v]}$, with vehicle information of detected vehicles, i.e.,

$$\mathbf{x}_i = [x_i, y_i, v_i, \theta_i, l_i, h_i, w_i]^\top \in \mathbb{R}^7 \quad (1)$$

$$\mathbf{X} = [\mathbf{x}_e, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_v}] \in \mathbb{X} = \mathbb{R}^{7\times(n_v+1)} \quad (2)$$

In each $\mathbf{x}_i$, x_i and y_i are the vehicle position, with $x_e = 0$, v_i is the vehicle speed, $\theta_i \in (-\pi, \pi]$ is the heading of the vehicle w.r.t. the ego lane, $l_i \in \mathbb{N}$ is the lane number, and h_i and $w_i \in \mathbb{R}_+$ are the length and width of the vehicle. It is assumed all information can be accurately detected without communication. As opposed to the intended actions of obstacle vehicles, which need to be communicated, as used in Fabiani and Grammatico [2019].

2.2 Actions

Based on the state information provided to the DM by the perception module, an action is chosen. An action is

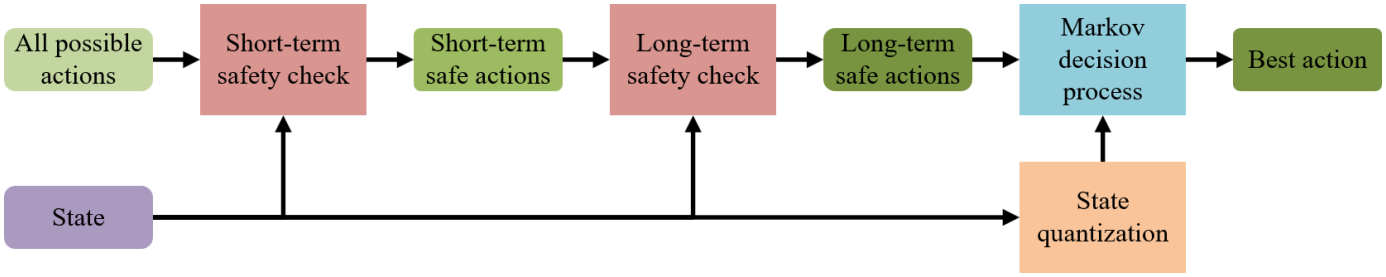

Fig. 3. Decision Maker (DM) architecture. Rectangles show the different components. Arrows show the flow of information between components. The information, which is transferred, is given in rounded rectangles.

defined by the combination of a lateral maneuver and a speed update. There are three lateral maneuvers (lane change to the left (LCL), lane keeping (LK) and lane change to the right (LCR)) and three speed maneuvers (increase speed (a), maintain speed (c) and reduce speed (d)). The increase and reduce speed are offset from the current speed by one quantization step, as is detailed in Section 5.2. This results in a set A consisting of 9 possible actions, see Table 1.

2.3 Objectives and problem formulation

The DM has various objectives. Clearly, it should produce safe behavior. Next to safety, the ego vehicle should also comply with traffic rules and move towards a destination. Lastly, the controller should do all this without too much activity, as to create comfortable driving. From these objectives, three quantitative performance criteria are selected: ***safety***, ***liveness*** and ***activity***. *Safety* is measured by the Time To Collision (TTC) as introduced by Hayward [1971]. TTC is a common safety measure for two vehicles and it gives the time until a collision will occur given constant heading and velocity for both vehicles. A higher TTC suggests a safer situation. TTC is computed using the method given in Laureshyn et al. [2010]. For multiple vehicles, the lowest TTC for all obstacle vehicles is used. The TTC is limited between 0 and 15 seconds, since negative TTC and very distant collisions are not considered relevant. TTC gives the safety of a single moment, the root mean square (RMS) of the TTC for each sample time is used to get the safety of a longer period of time. By using RMS instead of mean, lower TTCs are penalized more.

$$safety = 15 - \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(15 - TTC(i))^2}, \qquad (3)$$

where n is the number of samples and $TTC(i)$ is the lowest TTC for all vehicles at sample i. This nonnegative measure is chosen such that a higher number means a safer controller. Obviously, stopping at the side of the road is considered a perfectly safe solution. However, this is not a desirable solution. To prevent such solutions, *liveness* is introduced. We formalize *liveness* by the distance d_{trav} traveled by the ego vehicle. A larger d_{trav} means better performance. *Activity* is measured by the number of initiated lane changes. Note that for this criterion, smaller is better. These measures quantify safe and smart behavior. Scalability means that algorithms allow for larger systems (more vehicles), while remaining computationally tractable. The DM has to output an action every $1s$. Therefore, the available time for all computations combined is limited to $1s$. The problem that we want to address can now roughly be stated as: given $\mathbf{X}$ at a (given) discrete time, find $a \in A$ that guarantees a lower bound on single moment safety, maximizes d_{trav}, maximizes *safety*, and minimizes the number of lane changes. Note that safety is still included in the optimization, because otherwise it might be beneficial to keep safety as low as the constraints admit, which is not desired.

3. DECISION MAKER ARCHITECTURE

The DM selects an action in two stages (see Fig. 3). First, safe actions are selected (the red components), second, the best safe action is selected to be smart (the blue component). Safety is checked for all actions in a short-term and a long-term safety check. Only actions that pass both safety checks are considered safe. An action is selected in the second stage by searching for the optimal action in an MDP. The orange component creates a quantized state for use in the MDP. The arrows show the flow of information between the components. The information, which is transferred between components, is given in the rounded rectangles in Fig. 3.

4. SAFETY STAGE

The first stage of the DM is the safety stage. This stage is split into two parts, which are introduced below.

4.1 Short-term safety

Short-term safety is measured here by taking the time to collision (TTC) of a single time step. A higher TTC suggests a safer situation. TTC is computed based on $\mathbf{X}$ of (2). Note that it depends on position and velocities of both ego vehicle and surrounding vehicles. Due to computational delay, we make a one-step ahead prediction of $\mathbf{X}$ first, which is used in the safety check. An action is safe if the TTC for that action exceeds a certain threshold ($1.5s$) for all vehicles. The threshold of $1.5s$ was chosen to ensure safety during the execution of the action and a short while after.

4.2 Long-term safety

The long-term safety criterion is only checked for short-term safe actions. The criterion is a threshold on the TTC, as is also used for short-term safety. In addition, lane changes to a lane in which a vehicle is already present are removed from the action set. These actions are clearly unsafe, but this is not always reflected in the TTC, because

Algorithm 1. Safety computations for short-term and long-term safety.

Input:
$\mathbf{X}$: State
a_{prev}: Previous action

1: $A_{short} \leftarrow \emptyset$
2: $A_{long} \leftarrow \emptyset$
3: $a^* \leftarrow LK_c$
4: $\mathbf{X} \leftarrow f(\mathbf{X}, a_{prev})$
5: **for all** $a \in A$ **do**
6: **if** $TTC(X,a) \geqslant 1.5s$ **then**
7: add a to A_{short}
8: **end if**
9: **end for**
10: **for all** $a \in A_{short}$ **do**
11: **if** $TTC(X,a) \geqslant 1.5s \wedge a \in LCL_* \wedge \forall i \in \{1,\dots,n_v\} : l_i = l_e + 1 \Rightarrow x_i \notin [x_e, x_e + 30]$ **then**
12: add a to A_{long}
13: **else if** $TTC(X,a) \geqslant 1.5s \wedge a \in LCR_* \wedge \forall i \in \{1,\dots,n_v\} : l_i = l_e - 1 \Rightarrow x_i \notin [x_e, x_e + 30]$ **then**
14: add a to A_{long}
15: **end if**
16: **end for**
17: $a^* \leftarrow \text{argmax}_{a \in A_{long}} TTC(X,a)$

for TTC a constant heading is assumed. By removing the actions that do not meet the long-term criterion, the search space for finding the optimal action can be reduced significantly. The remaining actions are considered safe and will be used in the MDP to find the optimal (smart) action.

4.3 Algorithm

Algorithm 1 implements both short- and long-term safety checks. As input, it takes the measured state $\mathbf{X}$ and the previous action a_{prev}. First, the state is predicted one step ahead, to account for (computational) delays as explained above (line 4). This is done using a discrete-time linear model, given by function $f : \mathbb{X} \times A \to \mathbb{X}$, assuming constant velocities.

Next, the short-term safety check is executed for all actions (lines 5-9). Only actions that meet the criterion in line 6 are added to A_{short}, being the set of short-term safe actions in line 7.

All short-term safe actions are then checked for long term safety (lines 10-16). If an action has $TTC \geqslant 1.5s$ and meets the additional condition regarding a lane change, it is added to A_{long}. The additional condition checks whether there is an obstacle vehicle $\mathbf{x}_i$ present at the destination lane l_e+1 (l_e-1). Here, $LCL_*(LCR_*)$ denotes the set of actions corresponding to a lane change to the left (right) with any speed maneuver, e.g., $LCL_* = \{LCL_a, LCL_c, LCL_d\}$. This way, only the set of long-term safe actions, A_{long}, are considered for the MDP. Typically, 4 to 6 out of 9 actions pass the safety test. Last, the safest action is marked (a^*) in line 17. a^* is used as a 'warm start' for the optimality stage, which is explained in the next section.

Table 2. State quantization steps.

Variable	$x_i[m]$	$y_i[m]$	$v_i[m/s]$	$\theta_i[rad]$
Quantization size	8	0.5	1	0.01
Minimal value	−152	−2.0	0	−0.11
Maximal value	152	2.0	42	0.11

5. OPTIMALITY STAGE

The purpose of the optimality stage is to find the optimal action from the set of safe actions. Finding an action is modeled as a Markov decision process (MDP), see Sutton and Barto [1998]. An MDP is used because the model includes future actions into the decision making process and can handle stochastic changes in the state very well. In an MDP, each action incurs a reward. The goal is to maximize the reward over future actions. This closely resembles the driving process, where, for example, performing lane changes reduces comfort, but it might speed up the journey by overtaking slow vehicles. Hence, by proper selection of the reward, tradeoffs can be made. In this section, first a definition of MDP is recalled. Then, each element of the model is described for our purpose. Last, the algorithm to find the best action is introduced.

5.1 Markov decision process

An MDP (Sutton and Barto [1998]) is defined as a 4-tuple (S, A, T, R), where S is a set of states, A is a set of actions, $T : S \times A \times S \to [0,1]$, such that $\sum_{s' \in S} T(s,a,s') = 1$ for all $s \in S$ and $a \in A$, is a transition function representing the stochastic change of state, based on the chosen action in a time step. This relation is given by the transition function

$$T(s,a,s') = \Pr\big(s(k+1) = s' \mid s(k) = s, a(k) = a\big), \quad (4)$$

where $k \in \mathbb{N}$. $T(s,a,s')$ gives the probability that the state at time step $k+1$ is s' if, at time step k, action a is taken in state s. $R : S \times A \to \mathbb{R}$ is a function representing the immediate reward for taking an action, when in a state.

5.2 The state of an MDP for a DM

The state $s \in S$ is inferred from the state representation $\mathbf{X}$ of (2). To consider maneuver continuation as a condition for optimal behavior, the previous action is also included in the state. The algorithm that is used to find the optimal action in this paper requires that S is a finite set or countable infinite set. Therefore, the variables in $\mathbf{X}$ are quantized, see, e.g., Lunze [1994].

The continuous variables in $\mathbf{X}$ are quantized, the quantization steps and extremal continuous values are given in Table 2. These values were chosen because they capture the different states in highway driving and changes between those states. Choosing a smaller quantization step results in more computations. Let the set of possible quantized values for position be defined as $\bar{X}, \bar{Y}$, for speed as $\bar{V}$, for heading as $\bar{\Theta}$, and lane L, which is not quantized any further, all being subsets of $\mathbb{N}_0$. The set of all states is now a Cartesian product of these sets and the action set:

$$S = (\bar{X} \times \bar{Y} \times \bar{V} \times \bar{\Theta} \times L)^{n_v+1} \times A. \quad (5)$$

As the state space is large (10^{17} when considering the ego vehicle and two obstacle vehicles), the MDP is solved online, i.e., the optimal action is found for the current

state, at every time step. This is done because an offline solution would result in a high-dimensional mapping $S \rightarrow A$. An online solution considers only states (and values) that are reachable within the prediction horizon when finding an action. Note that searches start at the current state $s_0 \in S$ of the system at the current discrete-time step. A disadvantage of this state definition is that one-step ahead prediction is not used here. However, it is expected that using one-step ahead prediction will improve performance.

5.3 Action

The actions a describe how to control the system. Action set A contains all possible 'commands' that can be given to a system. The actions used in the MDP are given in Table 1. Not all actions have to be possible in every state, e.g., when being in the rightmost lane, a LCR command is not possible. Therefore, the set $A_s \subseteq A$ of available actions in state s is introduced. Typically, there are 6 actions in A_s. A_{long} contains all actions that were found to be safe in the safety stage for current state s_0. Note that $A_{long} \subseteq A_{s_0}$. For all future actions, A_s is based on the presence of a left or right lane. The presence of adjacent lanes is assumed constant over the horizon, i.e., no on-ramps or lane merges etc.

5.4 Transition function

Whenever the DM takes an action a, the state s changes to a next state s'. How the state changes depends not only on a, but might also depend on factors that cannot be controlled, such as other road users. In addition, uncertainties are introduced by sensor measurements and state quantization. To include these factors, a probabilistic relation is used. The transition function expresses the probabilities of reaching each possible configuration of vehicles, given an action. The transition function for the MDP created here is based on simplified vehicle dynamics and the chosen maneuver. Stochastic models of road-user behavior and sensor uncertainties are not included in the model below.

The vehicles are assumed to be point masses with constant velocity. The ego heading and speed change based on the chosen action. The speed of other road users is assumed to be constant and their heading (with respect to the lane) is assumed to be constant and 0. A more accurate representation could be obtained using a bicycle model, see [Mitschke and Wallentowitz, 1972, pp. 613-623], or even more detailed models. However, such models would be computationally more intensive.

Predictions are made based on the chosen action. When the maneuver is lane keeping, the quantized heading is changed such that the vehicle is guided towards the current lane center. When the maneuver is a lane change, the quantized heading is updated, depending on the direction of the lane change and the in-lane position. The heading and speed are changed by one quantization step, as this results in an accurate representation of a lane change in the quantized domain. Table 3 shows the change in quantized heading, depending on maneuver and position. Note that the boundaries on $\bar{y}$ seem asymmetric, but they are not, since the lane center is on the boundary between $\bar{y} = -1$ and $\bar{y} = 0$. The change of heading for a lane change is assumed to be dependent only on the lateral position.

Table 3. Change in heading per action.

Lane keeping			
	$\bar{y} < -1$	$-1 \leqslant \bar{y} \leqslant 0$	$0 < \bar{y}$
$\bar{\theta} > 0$	$+0$	-1	-1
$\bar{\theta} = 0$	$+1$	$+0$	-1
$\bar{\theta} < 0$	$+1$	$+1$	$+0$
Lane changing			
	$\bar{y} < -1$	$-1 \leqslant \bar{y} \leqslant 0$	$0 < \bar{y}$
left	-1	$+1$	$+1$
right	$+1$	-1	-1

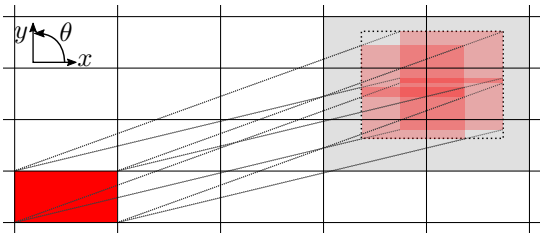

Fig. 4. Computation of reachable region. The ego vehicle is in the bottom left red cell. Given the current action, it can go to the shaded red area in one time step. This is translated to a probability for each of the gray cells.

A visualization of the transition function is given in Fig. 4. The grid represents the quantization intervals of the continuous position. The origin cell (dark red) shows the initial position of the ego vehicle. By combining all possible speeds and headings for this quantized position, a region is found where the ego vehicle can end up in one time step. Each cell inside this region is assigned a probability, related to the area, of ending up in that cell. As such, the state quantization and underlying state dynamics lead to a probabilistic transition function as given in (4), see Lunze [1994]. Typically, one action can lead to between 100 and 200 different next states.

5.5 Reward

After every action, a reward is obtained, given by the reward function $R : S \times A \rightarrow \mathbb{R}_{\geqslant 0}$. Note that this is an immediate reward; it only depends on the present action. We define the reward function as a linear combination of chosen features $R_i : S \times A \rightarrow [0, 1]$, i.e.

$$R(s, a) = \sum_{i \in \mathcal{R}} \lambda_i R_i(s, a), \tag{6}$$

where $\mathcal{R}$ is the set of feature indices and $\lambda_i \in \mathbb{R}_{\geqslant 0}$ are the weights corresponding to each feature. The used features are given below:

Speed R_v: Deviations from a given reference speed v_{ref} are penalized quadratically, to motivate overtaking slow vehicles. A quadratic penalty was chosen to give a larger penalty for larger deviations. In addition, urgency is enforced by linearly penalizing negative deviation from the desired speed. It is not possible to gain a higher urgency reward by exceeding v_{ref}.

Activity penalty R_a: Taking unnecessary actions is penalized for driver comfort and fuel efficiency. A reward

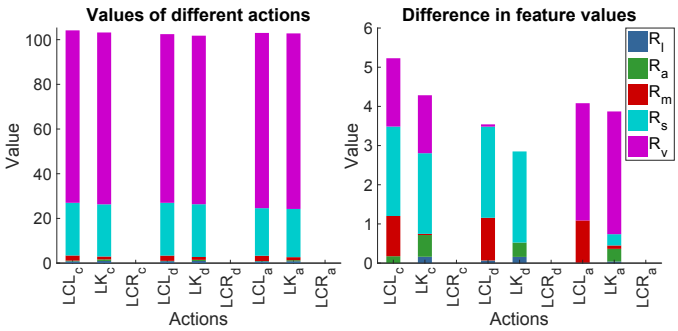

Fig. 5. Values for different actions, divided by feature. Left: total value for different actions at a given time step. Right: difference between feature contributions at the same time step.

is given if the chosen action is lane-keeping or when the chosen speed is constant.

Safety R_s: The time to collision is used to give preference to behavior that is even safer than prescribed by the safety components.

Lane preference R_l: A reward is given for being in the rightmost lane or moving towards it.

Maneuver continuation R_m: Similar to the unnecessary action penalty, a reward is given if a lane change is continued, to promote consistent behavior.

All these rewards are combined using weights λ_i, which are chosen manually based on observed behavior in simulations. Some relations between individual weights were found, e.g., between the activity penalty R_a and maneuver continuation R_m.

An example of the decision making process is visualized in Fig. 5. On the left-hand side of the figure the expected rewards are given for each action at a certain time step and in a certain state. Each individual bar is colored according to the portion of the value that originates from different features. Note that speed and safety have large contributions to the value, whereas the others have not. It is not easily seen that LCL_c is the best action.

To more clearly show the impact of all features, the minimal value over allowed actions of a feature is subtracted from the values on the right hand side of Fig. 5. This is most clearly seen for safety, since LCL_a had the lowest contribution for R_s, it now appears as 0. Note that the contribution of R_m has a significant impact on the best action (LCL_c), whereas it was barely visible in the left hand figure.

5.6 Anytime AO*

The best action, i.e., the action that has the highest expected reward, is found through Anytime AO*, as introduced in Bonet and Geffner [2012]. This algorithm was chosen because it can handle large state spaces and it can find a near-optimal action in a predefined time. This predefined time is equal to the remaining time, after safety computations, as explained in Section 3. Note that safety computations take much less than $1s$, which means that

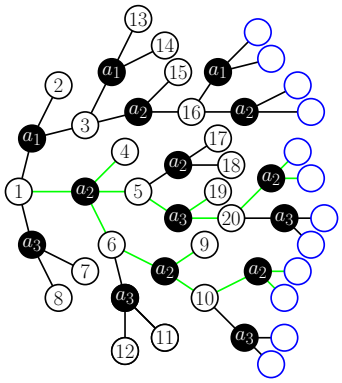

Fig. 6. Visualization of tree structure, states in white, possible actions in black. Blue circles are beyond the horizon. Node 1 is the initial state. Green edges denote a possible optimal subgraph, i.e., the subgraph that is obtained by choosing the action with the highest reward (rewards are not shown).

almost one second remains for finding the optimal action. Estimation function h is chosen such that it accurately approximates the future rewards, as in (6), with little computational complexity. With each iteration, the value estimates of explored nodes get closer to the real values. The action with the highest expected reward is selected as the optimal action. A visualization of the tree structure is given in Fig. 6. For the MDP, as given above, the structure is much larger than in the figure. Below, the deviations from the algorithm in Bonet and Geffner [2012] are described.

The function that is used to estimate the value of a state s_k is based on the reward for the action a_{k-1} that was taken to end up in this state.

$$h(s_k, \tau) = \sum_{i=0}^{\tau-1} \gamma^i R(s_{k-1}, a_{k-1}) \tag{7}$$

Note that previous action a_{k-1} is included in s_k, as explained in Section 5.2. The rewards are assumed constant for the remaining horizon τ. This function was chosen because it approximates the value using R, without having to compute all possible future paths and rewards. It is assumed h always returns the same value for a particular state and action, such that no averaging is needed. In Bonet and Geffner [2012], averaging is used to get more accurate predictions.

The algorithm continues until the entire tree is expanded or the time runs out. Since the duration of computations cannot be measured during simulation, a fixed number of iterations is used.

Selecting a node to expand starts at the root node. Repeatedly, a successor state is selected, until a tip node is reached. The probability of selecting an action in the best partial graph is computed differently from the original algorithm in Bonet and Geffner [2012]. For one iteration of the algorithm, only optimal actions are selected with some probability p_{opt}, otherwise only random actions are selected. This means the best action could still be selected. Therefore, the actual probability of selecting a node inside the best partial graph is slightly higher than p_{opt}.

Table 4. Scenario desciptions and results, based on the criteria given in Section 2.3.

Scenario name	Description	Expected behavior	Rule-based			Two stage		
			safety	d_{trav}	n_{LC}	*safety*	d_{trav}	n_{LC}
1. Empty road	No obstacle vehicles	Maintaining speed and lane keeping	15.00	778	0	15.00	778	0
2. Normal overtake	A slower vehicle before the ego vehicle	Overtaking the slower vehicle	12.90	778	2	13.13	780	2
3. Fast overtake	A slightly slower vehicle before the ego vehicle	Lane keeping and decelerating.	13.30	764	2	13.30	742	0
4. Double overtake	Two slower vehicles in front of the ego vehicle, close together	Overtaking both vehicles at once	9.92	725	4	11.39	729	2
5. Single overtake	Two slower vehicles in front of the ego vehicle, far apart	Overtaking both vehicles separately	12.05	778	4	10.52	698	3
6. No overtake	Two slower vehicles on different lanes in front of the ego vehicle	Lane keeping on either lane	10.03	655	1	11.69	647	0
7. Overtaken	The same as normal overtake, with a faster vehicle on the left lane	Waiting before overtaking	11.99	774	2	11.70	712	2
8. Overtake interrupt	The ego vehicle is left of a vehicle, with a faster vehicle behind	Accelerate to promote safety	9.35	667	1	12.14	701	1

Note that this method has similarities to model predictive control (MPC), where an optimal sequence of control inputs (actions) is sought too. After finding such a sequence, only the first input is applied, and the search starts again.

6. RESULTS

The behavior of the DM is validated through simulation in 8 important scenarios (see Table 4) in highway driving. Certain behavior is expected for each scenario, as given in the table. It can easily be checked whether this behavior does indeed occur. A schematic view of relative vehicle positions for all scenarios is given in Fig. 7. All obstacle vehicles use lane keeping with adaptive cruise control. The dynamics of all obstacle vehicles and the ego vehicle are modeled using a single-track bicycle model, see [Mitschke and Wallentowitz, 1972, pp. 613-623]. Note that the model used for simulation is more realistic than the model used for the DM. Each simulation has a simulation time of $40s$ and sample time $t_s = 0.01s$.

The performance of the two-stage DM is compared to the performance of a manually programmed rule-based controller. The performance of both DMs is given in Table 4, based on the criteria given in Section 2.3. As can be seen, the two-stage DM has an equal or higher safety score for most scenarios (except 5 and 7). However, this results in slightly less distance covered.

From the table, it can be observed that both DMs successfully perform a *normal overtake* (2), but the two-stage DM is slightly safer. However, the rule-based DM overtakes a slightly slower vehicle as well, whereas the two-stage DM reduces its speed, as can be seen from n_LC and d_{trav} for *fast overtake* (3). As a result, the two-stage DM covers $22m$ less distance, but also executes two fewer lane changes.

In Fig. 8, the speed profiles for the *Overtake interrupt* scenario (8) are given. The rule-based DM is shown in the upper part, the two-stage DM is shown in the lower part. When the background is shaded, a lane change to the right is performed. The two-stage DM accelerates above the preferred speed ($16.7m/s$). This increases the safety with respect to the approaching vehicle that is much faster. After finishing its overtake, the DM slows down to slightly above the desired speed. This happens because the reward gained for driving the desired speed does not outweigh the cost for deceleration. The rule-based DM maintains its

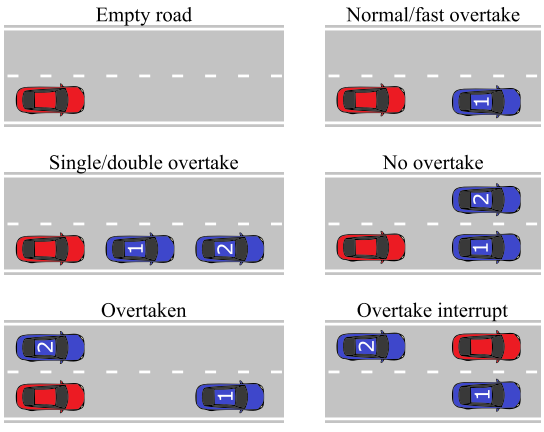

Fig. 7. Schematic view of initial vehicle positions in the scenarios. Ego vehicle in red, obstacles in blue. See Table 4 for a textual description of the scenarios.

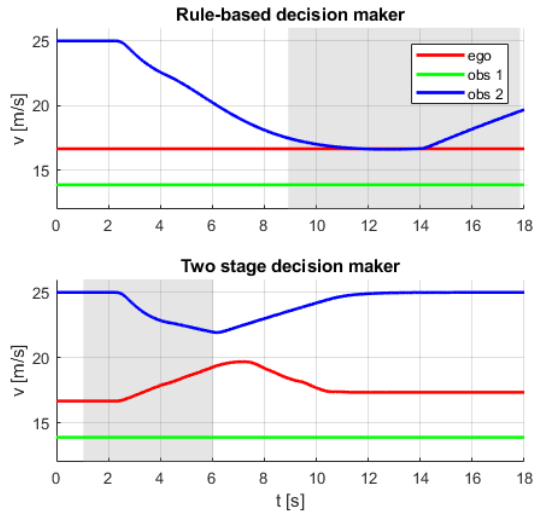

Fig. 8. Speed plots for the *overtake interrupt* (8). Top: Rule-based DM. Bottom: Two-stage DM. A gray background denotes LCR.

desired speed, causing a large deceleration for the obstacle vehicle, which then accelerates again after passing.

7. CONCLUSIONS

This paper presents a generic DM that incorporates safety considerations for highway scenarios, using scalable algorithms. The DM is based on two-stage architecture. In the first stage, the safety of all possible actions is evaluated, while in the second stage, the optimal action is selected from these safe actions. The safety of 1 time step is checked first, yielding short-term safe actions, after which the long-term safety of these actions is checked, leading to the set of safe actions. Then, in the second stage, an MDP is used to find the best action taking into account both the safety of actions and the future evolution of the multi-vehicle system. The MDP uses a quantized state and manually weighted reward features involving safety, liveness and activity. It is solved using a variation of the Anytime AO* algorithm.

The performance is compared to a rule-based DM according to three different criteria. To show the broad applicability, both DMs are compared in 8 relevant highway scenarios. From this comparison, it is concluded that the overall architecture and algorithm design is a viable methodology to implement a scalable and generic DM for AVs in a highway setting. It can be observed that the proposed DM design considers both the safety and the future effects of its actions. However, to assess the efficiency of the new controller design, a comparison to the performance of a human driver is needed.

As future work, a different method for the long-term safety check could be selected, e.g., using a measure that accounts for the probability of a collision, as in van Nunen et al. [2011]. In addition, a richer class of scenarios can be considered, e.g., with multiple vehicles or with a different number of lanes. The possibility of using a partially observable MDP (POMDP) should be investigated as well, since the state and transition function of the MDP are not fully known. Besides that, the optimality stage can be extended to include sensor uncertainties and road-user behavior, to make some of the assumptions less restrictive.

REFERENCES

P Abbeel and AY Ng. Apprenticeship learning via inverse reinforcement learning. In *Proc. 21st Int. Conf. Machine learning*, page 1. ACM, 2004.

B Bonet and H Geffner. Action selection for MDPs: Anytime AO* versus UCT. In *Proc. 26th AAAI Conf. AI*. AAAI, 2012.

F Fabiani and S Grammatico. Multi-vehicle automated driving as a generalized mixed-integer potential game. *IEEE Trans. Intell. Transp. Systems*, 2019.

P Falcone, F Borrelli, J Asgari, HE Tseng, and D Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE Trans. control systems technology*, 15(3): 566–580, 2007.

Y Guan, SE Li, J Duan, W Wang, and B Cheng. Markov probabilistic decision making of self-driving cars in highway with random traffic flow: a simulation study. *J. of Intell. and Conn. Veh.*, 1(2):77–84, 2018.

J Hayward. *Near misses as a measure of safety at urban intersections*. Pennsylvania Transportation and Traffic Safety Center, 1971.

C Kim and R Langari. Game theory based autonomous vehicles operation. *Int. J. Vehicle Design*, 65(4):360–383, 2014.

K Kim and PR Kumar. An MPC-based approach to provable system-wide safety and liveness of autonomous ground traffic. *IEEE Trans. Automatic Control*, 59(12): 3341–3356, 2014.

T Korssen, V Dolk, JM van de Mortel-Fronczak, MA Reniers, and WPMH Heemels. Systematic model-based design and implementation of supervisors for advanced driver assistance systems. *IEEE Trans. Intell. Transp. Systems*, 19(2):533–544, 2017.

A Laureshyn, A Svensson, and C Hydén. Evaluation of traffic safety, based on micro-level behavioural data: Theoretical framework and first implementation. *Accident Analysis & Prevention*, 42(6):1637–1646, 2010.

W Liu, SW Kim, S Pendleton, and MH Ang. Situation-aware decision making for autonomous driving on urban road using online POMDP. In *Proc. 2018 IEEE Intell. Veh. Symposium*, pages 1126–1133. IEEE, 2015.

J Lunze. Qualitative modelling of linear dynamical systems with quantized state measurements. *Automatica*, 30(3):417–431, 1994.

M Mitschke and H Wallentowitz. Lineares einspurmodell, objektive kenngrößen, subjektivurteile. In *Dynamik der kraftfahrzeuge*, chapter 4. Springer, 1972.

F Mohseni, E Frisk, J Åslund, and L Nielsen. Distributed model predictive control for highway maneuvers. *IFAC-PapersOnLine*, 50(1):8531–8536, 2017.

W Schwarting, J Alonso-Mora, and D Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:187–210, 2018.

RS Sutton and AG Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

E van Nunen, T van den Broek, M Kwakkernaat, and D Kotiadis. Implementation of probabilistic risk estimation for VRU safety. In *8th Int. Workshop on Intell. Transp.*, 2011.

T Wongpiromsarn, U Topcu, and RM Murray. Receding horizon control for temporal logic specifications. In *Proc. 13th ACM Int. Conf. Hybrid systems: computation and control*, pages 101–110. ACM, 2010.

S Zhou, Y Wang, M Zheng, and M Tomizuka. A hierarchical planning and control framework for structured highway driving. *IFAC-PapersOnLine*, 50(1):9101–9107, 2017.