Design of a decision maker for autonomous vehicles with safety and optimality considerations over a future horizon

Verbakel, Jeroen J.

Award date:
2019

Control System Technology Group
Department of Mechanical Engineering
University of Technology Eindhoven

Integrated Vehicle Safety Department
Traffic & Transport
TNO Helmond

Design of a decision maker for autonomous vehicles with safety and optimality considerations over a future horizon

MSc thesis

Author:
J.J. Verbakel

ID Number:
0835870

Eindhoven, June 28, 2019

CST2019.048

Contents

List of symbols

symbol	unit	name
$(\cdot)_{max}$		Maximal value for a quantized value.
$(\cdot)_{min}$		Minimal value for a quantized value.
$\bar{(\cdot)}$	$-$	Quantized variable.
A		Set of all actions.
A_s		Set of possible actions in s for an MDP.
a	$-$	Action of an MDP.
a^*	$-$	Optimal action for a node.
a_p	$-$	Parent action.
a_{prev}	$-$	Action taken during the last time step.
$B_l^k(d)$		Bad location predicate for location l (at iteration k)
c_0	m	Coefficient of y_{lane}^E
c_1	$-$	Coefficient of y_{lane}^E
c_2	$1/m$	Coefficient of y_{lane}^E
c_3	$1/m^2$	Coefficient of y_{lane}^E
$\mathcal{D}$		Domain for all variables in an FSM.
$d(\cdot)$		Quantization step for a variable.
d_0		Initial values of all variables in an FSM.
d_i	m	Distance between ego vehicle and vehicle i.
d_{trav}	m	Distance traveled by the ego vehicle.
$\dot{d}_i$	m/s	Rate of change of d_i.
DUC	m	Lateral distance upon intersection of paths.
$do\_optimal$	$-$	Boolean parameter for node selection in Anytime AO*.
E		Ego center-of-gravity coordinate frame
e		A transition in an FSM.
$\mathcal{E}$		Set of all transitions in an FSM.
$\mathcal{E}_c$		Set of all controllable transitions in an FSM.
$\mathcal{E}_u$		Set of all uncontrollable transitions in an FSM.
f_e		Update of transition e.
G		Tree graph.
G^*		Best partial graph.
g_e		Guard of transition e.
H	$-$	Horizon.
h	$-$	Heuristic function.
h_i	m	Length of vehicle i.

symbol	unit	name
L		Ego lane coordinate frame
l_0		Initial location for an FSM.
l_i	–	Lane number of vehicle i.
$\mathcal{L}$		Set of all Locations in a finite state machine (FSM).
$\mathcal{L}_f$		Set of all forbidden locations in an FSM.
$\mathcal{L}_m$		Set of all marked locations in an FSM.
N	–	Number of Samples.
$N_l^k(d)$		Non-blocking predicate for location l (at iteration k)
n_{LC}	–	Number of lane changes by the ego vehicle.
o_e		Origin location of transition e.
p	–	Probability of ending in a state after taking an action.
p_{opt}	–	Tuning parameter for node selection in Anytime AO*.
$Q^\pi(s)$		Action value function for π in an MDP.
$Q(s)$		Optimal action value function for an MDP.
$Q(s,d)$		Action value function for Anytime AO*.
R		Reward function of an MDP.
R_i		Reward feature i.
r	–	Immediate reward.
$\mathcal{R}$		Set of feature indices for rewards.
S		Set of all states of a Markov decision process(MDP).
S'_a		Set of all reachable states for action a.
s	–	State of an MDP.
s'	–	successor state of s for an MDP.
s_0	–	Initial state of an MDP.
s_p	–	Parent node.
T		Transition function of an MDP.
$\mathcal{T}$		Set of all found nodes for Anytime AO*.
t_e		Target location of transition e.
t_s	s	Sample time.
`tip`		Current tip for Anytime AO*.
TTC	s	Time to collision.
TTC_{max}	s	Maximal value of TTC.
$V^\pi(s)$		State value function for π in an MDP.
$V(s)$		Optimal state value function for an MDP.
$V(s,d)$		State value function for Anytime AO*.

symbol	unit	name
V^+	$-$	Value after updating node for Anytime AO*.
V^-	$-$	Value before updating node for Anytime AO*.
v_i	m/s	Speed of vehicle i.
v_x	m/s	Relative longitudinal velocity.
v_y	m/s	Relative lateral velocity.
$\bar{v}_{ref}$	$-$	Quantized reference speed.
w_i	m	Width of vehicle i.
w_l	m	Lane width.
x_B	m	Closest reachable value of x in one time step.
x_T	m	Farthest reachable value of x in one time step.
x_i	m	Longitudinal position of vehicle i.
x_{TTC}	m	Maximal value of x for using TTC.
$\tilde{x}_i$	m	Longitudinal distance between ego vehicle and vehicle i.
$\mathbf{X}$		Multi-vehicle system state.
$\mathbf{x}$		Vehicle vector.
$\mathcal{X}$		Longitudinal direction of coordinate frames
y^E_{lane}	m	Polynomial that describes the ego lane center
y_L	m	Leftmost reachable value of y in one time step.
y_R	m	Rightmost reachable value of y in one time step.
y_i	m	Lateral position of vehicle i.
y_{TTC}	m	Maximal value of DUC for assuming collision.
$\bar{y}_{LK}$	$-$	Maximal expected value of $\bar{y}$ for lane keeping.
$\tilde{y}_i$	m	Lateral distance between ego vehicle and vehicle i.
$\mathcal{Y}$		Lateral direction of coordinate frames
α_i	rad	Angle of d_i with respect to ego vehicle heading.
$\dot{\alpha}_i$	rad/s	Rate of change of α_i.
γ	$-$	Discount factor for an MDP.
δv	m/s^2	Change in speed for reward function.
$\delta\theta$	rad/s	Change in heading for reward function.
$\Delta(\cdot)$		Difference in a variable for ego vehicle and obstacle vehicle.
λ_i	$-$	Reward feature weight i.
$\pi(s)$		Policy for an MDP.
$\pi^*(s)$		Optimal policy for an MDP.
τ	$-$	Remaining horizon of a node in a tree graph.
θ_i	rad	Heading of vehicle i with respect to the lane center.

Summary

Modern cars are equipped with more and more Advanced Driver Assistance Systems (ADASs), such as (adaptive) cruise control and lane assist. As cars become more intelligent, automated vehicles (AVs) come closer to being commercially available. Many methods exist to design a decision maker, i.e., a controller that provides commands such as 'do a lane change to the right'. Every method has some disadvantages and there is no general agreement about which method is best suited to design a controller.

The purpose of this project is designing a decision maker that should choose desireable actions only from the set of safe actions. This decision maker should be able to work on all possible highway scenarios. The research question therefore is: *Given a state and goal, how can an autonomous vehicle make a discrete decision in a stochastic highway environment, such that safety is guaranteed?* To that end, this report proposes a new design of a controller that operates in two stages. In the first stage, the safety of all actions is checked and in the second stage, the optimal action is selected from the actions that are considered safe.

The safety stage uses discrete event modeling and supervisory controller synthesis to select the actions that are guaranteed to be safe one time step ($1s$) ahead. After that, the short-term safe actions are checked for long-term safety. Only the long-term safe actions are considered when searching the optimal action.

The process of finding this optimal action is modeled as a Markov decision process (MDP). To this end, the multi-vehicle state is quantized and a transition model based on simplified vehicle dynamics is used. Each action in an MDP incurs a reward, which is determined through a linear combination of reward features. These features are chosen by the designer, to model desireable behavior. To find the optimal action, multiple actions over a future horizon are considered, much like receding horizon control schemes. The optimal action to execute is found through Anytime AO*. It is an algorithm to find a (near) optimal action, within a predefined time.

The contribution of this project is a unique design, which combines guarantees on safety with optimal decision making. Moreover, the decision maker was designed for and validated on many different scenarios.

To validate the behavior of the decision maker, its performance is compared to a baseline controller according to three different criteria. The baseline controller is a rule-based controller that was developed at TNO. To show the broad applicability of the new controller, both controllers are compared on eight different scenarios.

From this validation, it can be concluded that the overall architecture and algorithm design is a viable methodology to implement scalable and generic decision making systems for autonomous driving. In addition, the decision maker shows safe behavior and consideration of the future impact of actions, for different scenarios.

1 Introduction

Modern cars are equipped with more and more Advanced Driver Assistance Systems (ADASs), such as (adaptive) cruise control and lane assist. As cars become more intelligent, automated vehicles (AVs) come closer to being commercially available. Additionally, with advanced vehicles, communication with other vehicles (V2V) or infrastructure (V2I) also becomes a realistic possibility. To speed up the arrival of AVs, researchers are investigating many different approaches to automated driving, as shown in [1]. In all approaches, a method is needed to determine what commands have to be provided to the vehicle, such that it can drive without the need of human aid.

A controller can provide these commands on several levels, which are introduced in [2], namely strategic, tactical and operational. For example, a strategic level command could be to take the next exit, due to traffic jams. On the tactical level, a command might be to switch lanes and overtake a slower vehicle. On the operational level, a vehicle might increase its acceleration to reach a desired speed. In this report, the focus is on the tactical level. Here, the controller subsystem that takes decisions on the tactical level is called the *decision maker*. It may also take decisions on other levels, but any subsystem on the tactical level is considered a decision maker. However, the algorithms proposed in this thesis to design a decision maker for the tactical layer may also be suitable to implement subsystems on other layers.

Many methods exist to design a decision maker. Often, these methods use some optimization program for taking a good action. In addition, some form of prediction of motion is used to obtain desireable behavior. Clearly, desireable behavior includes safety considerations. In the literature, two main approaches to incorporate safety considerations can be distinguished. First, safety is considered as a reward, along with other desiderata in an optimization scheme. This method is undesired, because the optimization algorithm may decide to sacrifice safety for other desiderata, such as urgency. Second, an auxiliary system is used to check the safety of an action after it is selected. Then the executed action will always be safe, but computational resources are wasted on finding an optimal action, which is then discarded because it is unsafe.

The purpose of this project is designing a decision maker that produces safe actions, while still taking desireable actions. This decision maker should be able to work on all possible highway scenarios. To this end, this report proposes a different design of a controller that operates in two stages. In the first stage, the safety of all actions is checked and in the second stage, the optimal action is selected from the actions that are considered safe.

The design that is proposed in this report uses various existing methods and algorithms. On the one hand, discrete event modeling and supervisory controller synthesis are used. On the other hand, Markov decision processes and tree search are used. These methods are closely related to dynamic programming, qualitative modeling and model predictive control. The contribution of this project is a unique design, which combines guarantees on safety with optimal decision making. Moreover, the decision maker was designed for, and validated on, many different scenarios.

This report is structured as follows. In Section 2, the current state of the art is given. In Section 3, the problem definition is formulated, and the two-stage controller design is introduced. Section 4 gives an overview of the techniques that are used in the design. Sections 5 and 6 explain each of the two stages of the controller. In Section 7 the results are presented and the performance is compared to the performance of a baseline controller. The baseline controller is a rule-based controller that was developed at TNO. Finally, conclusions and recommendations for future research are given in Section 8.

2 Literature review

In this section a representative set of controller designs for automated vehicles is discussed. First, three different control architectures are introduced. Next, several designs fitting these architectures are given. They are reviewed on safety and broader applicability, as mentioned in Section 1. Communication here means the need or possibility for the framework to include communication with other vehicles or a roadside unit. Many methods are tailored to a specific scenario, broader applicability refers to whether these methods also function in other scenarios. Safety of each method is reviewed in light of the definition of safety that is given in the paper which proposes the method.

In the future, communication can be included to improve the performance of the multi-vehicle system. For this thesis only one autonomous vehicle is considered, therefore, communication is not included. However, the possibility for communication should still be considered when evaluating methods.

2.1 Control architectures

In [1], a distinction between three different control architectures for AVs is made, these are shown in Figure 2.1. Namely, sequential planning, behavior-aware planning and end-to-end planning.

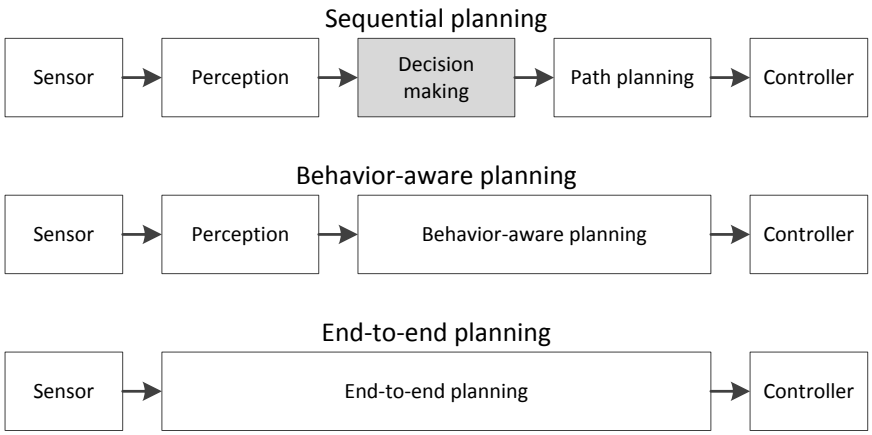

Figure 2.1: Three control architectures and the required components. The highlighted block shows the component to be designed in this project

Sequential planning separates perception, decision making and path planning into three sequential steps. A perception module converts raw input data into a usable state representation; how this is done is beyond the scope of this review. The state representation contains all the necessary information about the environment, e.g., the positions of all nearby vehicles. The decision maker uses this state representation to decide upon some discrete action from a set of possible actions, such as 'lane keeping'

or 'lane change to the left'. This action is then converted to a trajectory by the path planner, which is executed by low-level controllers. The algorithms used for path planning fall outside the scope of this project[1]. In this review the following methods are considered for sequential planning: manual programming, supervisor synthesis, game theory, Max-Plus algebra, temporal logic and Markov decision processes.

In behavior-aware planning, the motion plan is also (partially) generated by the decision maker. The input for the behavior-aware planner is a state representation generated by the perception module. The output is the trajectory that is used for the low-level controller. Note that the trajectory is actually an input to the vehicle on the operational layer. Since the controller decides what actions to take as well, it is still considered a decision maker. For behavior-aware planning methods, the following methods are considered: Game theory, MDP, temporal logic and Model Predictive Control.

If input data processing is incorporated in the decision making process as well, the controller is called an end-to-end controller. Usually, end-to-end controllers are made using machine learning, where the relation between input and output is inferred through real-world data or simulation, as is mentioned in [1].

2.2 Sequential planning

The simplest method is manual programming[2]. A rule-based program is written that outputs a decision, based on the inputs. Whether the program includes communication can be chosen by the programmer. Safety considerations for this method depend on the programmer and only situations that are accounted for can be safely handled. This also shows the lack of scalability of the solution, because only predefined situations can be handled. Clearly, this is not a feasible solution, since not all possible situations can be included. Furthermore, this method has a high chance of errors, since every line of code is written manually.

It is also possible to generate a rule-based decision maker automatically, based upon models of the system and requirements. For instance, in [6], supervisor synthesis is used to create a cruise control system, including human interaction. The safety of a decision is guaranteed by construction. This method is easily scalable, by adding more components to the model of the system. Similar to manual programming, communication can be included or excluded when modeling the system. Supervisor synthesis suffers from the state explosion problem, but methods exist to cope with this, such as the method given in [7]. A supervisor prevents actions that drive the system into undesired states, but it does not choose the best action to take, as explained in [8]. Besides that, it is impossible for a synthesized supervisor to not follow the specified requirements. But in traffic, situations might occur where traffic rules need to be broken to guarantee safety, e.g. driving on the wrong lane when avoiding a static obstacle. Modeling all these situations as requirements impedes broader applicability of supervisor synthesis. In [9], a different synthesis algorithm is proposed, which can

[1] Examples include [3, 4, 5]

[2] no reference is included, because most papers on AVs state that this is a non-preferred method.

break rules to achieve a goal. The rules are only broken if inevitable, and then only as long as required.

In [10], a two-player game-theory model is used for deciding to switch lanes or not. The authors assume all vehicles communicate their utility values, meaning this only works for CAVs. They give each vehicle a 'safety envelope' to model safety. The safety of a decision is calculated by a function based upon the overlap in 'safety envelopes'. The authors mention that a different strategy should be used for guaranteeing safe mandatory lane changes (such as taking an exit). When more players are introduced in the game, the complexity increases exponentially. This is because all players' actions are interdependent.

At TNO, Max-Plus algebra was investigated as a possible approach to the decision maker, as shown in [11, 12]. Max-Plus algebra allows modeling discrete event systems in a framework similar to (discrete time) linear time invariant systems. The framework is especially suited for scheduling tasks, such as merging cars at a lane reduction. This does require V2I communication, since there is one global scheduler that communicates with the CAVs. The control strategy that is produced is cyclical, unlike the original system description. Because of the cyclic nature of the controller, it is assumed Max-Plus algebra is not suited for all general traffic situations, which, from the perspective of the driver, are not cyclic. It is possible to create controllers that are not cyclic, but Max-Plus algebra is best used for cyclical processes, as explained in [13].

In [14], Linear Temporal Logic(LTL) is used. LTL is a formal language to specify properties of systems, such as liveness and safety. Evaluating a LTL formula is a process that suffers from the state explosion problem. To prevent the state from becoming too large, the controller solves a sequence of smaller, short horizon problems while adhering the system requirements. The authors do not mention communication, it is not know whether this method can be extended to include communication. By dividing up the problem, this method is also feasible for larger problems. LTL allows for defining generic safety and liveness properties, such that the controller can easily be applied to other scenarios. The safety of the controller is guaranteed through the LTL formulae.

Another option is Markov decision processes (MDPs). MDPs are used in many different forms. An MDP has 4 basic elements: a set of states, a set of actions, a transition function and a reward function. A state represents the environment, it contains all necessary information to make a decision. The set of actions contains all actions that can be taken. The transition function gives the probability of ending in some state, upon taking an action from a (possibly different) state. The reward function gives the expected reward upon taking an action from a state. A more in-depth explanation is given in Section 4.3. An MDP controller determines what action to take, when in a specific state. The reward function should penalize unsafe decisions, but safety is not enforced. It can be imagined that other vehicles communicate intentions or information to create a more extensive state. This means MDP can be used for both communicating and non-communicating vehicles. The evaluation of safety and communication is similar for all MDP-based approaches, so these evaluations will not be explicitly mentioned for every MDP approach.

In [15], an MDP is used, which is solved offline. As a reward function, they use a weighted sum of several functions that are chosen by the designer, called features. These features represent a reward for important factors in driving, such as collision, desired velocity, fuel efficiency or driver comfort. The weights are found by an inverse reinforcement learning algorithm, based on expert demonstrations, this allows for choosing what features are relevant, and finding 'optimal' weights for them. This method is feasible for small state spaces, i.e. simple systems, but as the complexity of scenarios increases, computing or storing the policy might become infeasible.

In [16], the MDP is also solved offline. The transition model is based on multi-vehicle system predictions, which are based upon statistical data. Therefore, this approach works best in structured environments, where the prediction model is known. As a reward function, they use three features to represent efficiency and comfort, these are weighted manually. In addition, the vehicle receives a large negative reward upon collision with another vehicle.

An online solver for an MDP is used in [17], this means the optimal action is computed at each time-step. They also use a weighted feature reward function with manually designed weights. This allows penalizing unsafe decisions further, although this might decrease overall performance. The online solver can handle many different scenarios, if the reward functions are capable of capturing desiderata for all scenarios. In that paper, it is assumed that the entire state is known, this might not always be the case.

In [18], a partially observable MDP (POMDP) is used to handle uncertainties in the current state. They keep a set of probabilities for each state, based upon the observation made by the decision maker. Due to random sampling of the state space, this method remains feasible for complex systems. The reward function is an unweighted sum of several features, the features can be modified to weight the reward function.

2.3 Behavior-aware planning

In [19][3] a Stackelberg game is used. It is a game where vehicles take decisions one after another, which is more scalable than the system used in [10]. Since only the worst-case decision of other vehicles are considered as explained in [19]. They use a 'safe-mode' when getting too close to a predecessor to increase the safety of the system.

To handle uncertainties in the state, a point-based MDP(QMDP) is used in [20], where future rewards are calculated assuming only the first step is uncertain. The decision is made by assuming that enough random samples of future states are chosen to get a proper representation of the future state distribution. The scalability of QMDP is limited by the number of samples that is required to get a proper representation of the future state distribution. The authors make no mention of computation times, so little can be said about the broader applicability of QMDP related to other MDP methods. Unlike other MDP-based methods, this method is classified as behavior-aware, because the decision maker outputs a (quantized) speed reference for multiple time-steps ahead.

[3]The authors use controllers adapted from other papers, but these papers are unavailable.

Signal Temporal Logic(STL) is used in [21]. STL is a variant of temporal logic that allows propositions over real valued, continuous signals. In the paper, STL constraints on the position and velocity are defined. A receding horizon control scheme is used, this means that the condition *the system is always safe* turns into *the system is safe over the horizon*, otherwise the same guarantees as for LTL still hold. Similar to LTL, this method is scalable when using general formulae. In the paper, the environment is modeled as an external (uncontrollable) input. If communication is included, this can reduce uncertainties on the environment input.

For creating optimal paths, Model Predictive Control (MPC) can be used. In MPC, the optimal control inputs are computed for a (short) horizon, and the first control input is applied. This is called the receding horizon strategy, see e.g., in [22]. The control inputs are optimized for some (quadratic) cost function, satisfying constraints and system dynamics.

For example, in [23], a single lane ACC controller is designed through explicit MPC. Safety is measured through time headway, which is included in the reward function. Communication is not included in the paper, but could be included to increase the accuracy of the state. Explicit MPC is generally feasible for small states, (up to 5 variables, according to [24]), for larger states, different methods are needed.

In [25], implicit MPC is used to determine the (continuous) longitudinal speed and (discrete) highway lane. They use a special extension called Mixed-Logical-Dynamical (MLD) modeling, which allows the use of logical expressions in constraints. They consider safety as a set of constraints on the system, so the computed path is guaranteed to be safe. The authors do not mention communication, but it might be hard to implement this, for future trajectories of different vehicles are inter-dependent.

2.4 End-to-end methods

In [26], a deep neural network is proposed, which uses a discrete world occupancy grid. That paper focuses on the tuning of hyperparameter: parameters for the network, instead of the problem that is considered, such as preview horizon or what grid cells to consider as input. The hyperparameters are manually tuned to maximize the speed of the ego vehicle in simulation. If the ego vehicle observes squares it cannot normally percept, communication is necessary to gather all required information. To guarantee system safety, the authors use a separate (simple) collision avoidance system. It is assumed broader applicability depends on the coverage of different scenarios in the simulation.

Another end-to-end method is provided in [27], where no controller is modeled, but only lane changes are predicted from real-world data. It can be easily imagined that this prediction can be used as a control signal for the car. The system is based upon a five layer feed-forward neural network getting information about vehicles on its lane and both adjacent lanes. The authors give no guarantees with respect to safety, since the paper only discusses prediction of maneuvers. It is assumed communication can be used to give a more complete input layer, which improves the decision, but it is not

necessary. The current output is only a lane-change prediction, but it can be extended to predict other behavior, such as acceleration.

2.5 Safety

Above, it was mentioned for each method how the authors handled safety. There are also several papers related to AD, where safety is explicitly dealt with, which will be discussed below.

In [28], the safety of platooning vehicles is investigated. They define state-based safety regions. Inside these safety regions, either no collision occurs, or a collision occurs at a low relative velocity. In [29], invariant safe sets are computed based upon vehicle dynamics. When a system starts inside the invariant set, it will always stay there. The size of the invariant set, and therefore the safe region of operation, can increase if the controller or dynamical model is improved.

For cooperative driving, an interaction protocol is needed for safely navigating traffic. A merging protocol that allows for safely merging two platoons is proposed in [30]. In [31], another protocol is presented, based on finite-state machines without interleaving[4]. They use the concept of lane claiming to prevent two cars moving to the same space on a lane. This is an interesting solution to simultaneity, which is one of the largest problems encountered when implementing finite-state machines, as explained in [8].

The first two methods given above, based on state space divisions, are better suited for behavior-aware planning. The latter two methods are useful for both behavior-aware planning and sequential planning. They provide safe methods for communication, but are not complete solutions for guaranteeing safety.

2.6 Conclusion

For sequential planning, controller synthesis gives the best possibility for guarantees on the safety of the controller. However, there are still some problems, such as 'breaking rules' or choice, which have to be solved. Another versatile and promising method is MDPs. From the papers on MDPs, an approach that can handle uncertainty in the state is considered necessary, since the state cannot be fully known. POMDPs appear better than QMDPs, since the former includes information on the uncertainties. Some way to preclude unsafe actions has to be investigated to increase the possibility for safety guarantees. In addition, it has to be investigated if online solving is necessary, since the state space becomes very large (possibly even infinite), but there is only a small set of actions.

In discrete control, the decisions can easily be communicated, since a decision contains little data, such as a single integer decision ID. For trajectories, this is a more network intensive procedure, since there is a large amount of data, i.e., a vector of real values.

[4] interleaving means no two events occur simultaneously

Therefore, it is assumed the sequential planning methods can be extended to cooperative driving more easily than behavior-aware planning methods. This argument does not rely on exact values, but on the intuition that communication bandwidth is a finite resource, as demonstrated in [32].

At this moment, end-to-end methods are not considered as a feasible solution. The safety of the controller cannot be guaranteed, since the algorithm is a black-box model. A more elaborate study on opacity of machine learning algorithms is given in [33]. If one wants to guarantee safety, an additional safety system needs to be implemented. The influence of communication on neural networks is not known at this moment.

3 Problem definition

The decision makers mentioned in the previous section, give little guarantees with respect to safety of a decision. Some papers do not mention safety at all, others (e.g., [20, 15, 17]) consider safety as a reward, along with other desiderata. Often, an auxiliary system is used to guarantee safety, as in [10, 19, 17]. This shows that handling safety is an open problem. The goal of this project to solve this problem by proposing a decision maker that guarantees safety. This gives rise to the following research question:

Given a state and goal, how can an autonomous vehicle make a discrete decision in a stochastic highway environment, such that safety is guaranteed?

As mentioned in [17], highways provided a structured environment and consistent behavior on a short prediction horizon. Therefore, considering all possible trajectories is not necessary when making a decision. Because of that, sequential planning is used for this project. Sufficiently advanced perception and path planning algorithms have already been developed at TNO and are considered given.

To properly answer the research question and find a decision maker, a problem definition is required. This problem definition specifies the input, output and expected behavior of the system. The input to the system is given in the form of a state, it is introduced in Section 3.1. Because the system is intended to be implemented on a real vehicle, the origin of values is considered in that section as well. The output of the system is a discrete action, how these are formed is explained in Section 3.2. The expected behavior is measured based upon three criteria, these are given in Section 3.3. Based upon these definitions, a decision maker architecture was designed. This architecture is introduced in Section 3.4.

3.1 State

All different vehicle parameters and coordinate systems are shown in Figure 3.1. Parameters are given a superscript for their coordinate systems. Coordinate system E is centered on ego vehicle center-of-gravity E, with Cartesian axes $\mathcal{X}^E$, in the same direction as the ego-velocity and $\mathcal{Y}^E$, which is perpendicular to $\mathcal{X}^E$ in counterclockwise direction. Coordinate system L is centered on the intersection between $\mathcal{Y}^E$ and $y^E_{lane}(x)$, its $\mathcal{X}^L$ direction is tangential to the road. It is assumed that the road curvature is small, which holds for highway scenarios, such that L can be assumed Cartesian.

The state is divided in two parts, road information and a system state of all vehicles. The road information consists of a 3$^{\text{rd}}$ order polynomial that describes the path of the ego-lane in ego vehicle center-of-gravity coordinates (E). The polynomial is expressed using 4 variables:

$$y^E_{lane}(x^E) = c_0 + c_1 \cdot x^E + c_2 \cdot {x^E}^2 + c_3 \cdot {x^E}^3 \tag{1}$$

Typically, this polynomial is inferred from camera data, e.g., in a lane tracking system.

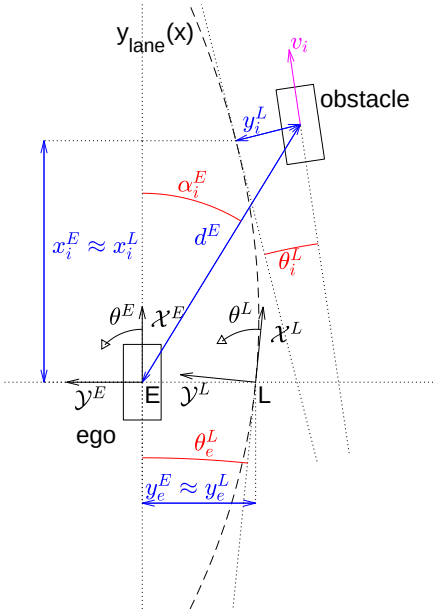

Figure 3.1: Coordinate systems.

The multi-vehicle system state $\mathbf{X}$ contains a vector $\mathbf{x}_i, i = e$ or $i \in \mathbb{N}_{[1,n_v]}$,with vehicle information of the ego vehicle and each of the n_v vehicles that are detected by on-board sensors, respectively.

$$\mathbf{x}_i = [x_i^L, y_i^L, v_i, \theta_i^L, l_i, h_i, w_i]^\top \in \mathbb{R}^7 \tag{2}$$

$$\mathbf{X} = [\mathbf{x}_e, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_v}] \in \mathbb{X} = \mathbb{R}^{7\times(n_v+1)} \tag{3}$$

In each $\mathbf{x}_i$, x_i^L and y_i^L are the vehicle position in ego-lane coordinates (L), with $x_e^L = 0$, v_i is the vehicle speed, $\theta_i^L \in (-\pi, \pi]$ is the heading of the vehicle w.r.t. the ego-lane, $l_i \in \mathbb{N}$ is the lane number, $h_i, w_i \in \mathbb{R}_+$ are the length and width of the vehicle. The state is associated to a discrete time step, the state at time step k is given by $\mathbf{X}(k)$.

For the ego-vehicle, the state is derived as follows.

- x_e^L : As mentioned above, $x_e^L = 0$ by definition.
- y_e^L : By substituting $x = x_e^L$ in (1), $y_e^L = c_0$.
- v_e : Ego speed is given. Typically, this is given by the vehicle speed sensor.
- θ_e^L : Since (1) is given relative to the vehicle heading, it can be determined through:

$$\theta_e^L = -\tan^{-1}\left(\frac{\mathrm{d}y_{lane}^E}{\mathrm{d}\mathcal{X}^E}|_{x_e^E}\right) = -\tan^{-1}(c_1) \tag{4}$$

  Note the minus sign, to get the right direction for the heading.
- l_e : Lane number is given. Typically this is done by the lane tracking system, which also gives $c_0, \ldots, c_3$ in (1).
- h_e : Vehicle length is a given constant.
- w_e : Vehicle width is a given constant.

For obstacle vehicles, the state is derived as follows:

- x_i^L and y_i^L : Target tracking returns the vehicle position relative to the ego vehicle in polar coordinates (d_i^E, α_i^E), as shown in the figure. This is translated to Cartesian coordinates in E:

$$\tilde{x}_i^E = d_i^E \cos(\alpha_i^E) \tag{5}$$
$$\tilde{y}_i^E = d_i^E \sin(\alpha_i^E) \tag{6}$$

  Note that there is a difference between x (distance along lane center) and $\tilde{x}$ (longitudinal distance from ego vehicle). The same holds for y (distance to lane center) and $\tilde{y}$ (lateral distance from ego vehicle). Next, y_i^L is found by taking the minimal distance between $(\tilde{x}_i^E, \tilde{y}_i^E)$ and $\big(x, y_{lane}^E(x)\big)$:

$$y_i^L = \min_{x \in \mathbb{R}} \sqrt{(x - \tilde{x}_i^E)^2 + (y_{lane}^E(x) - \tilde{y}_i^E)^2} \tag{7}$$

  This equation also gives $x_i^E = x$. To obtain x_i^L this distance should be taken along the lane-center, however, $x_i^L \approx x_i^E$ for small θ_e and road curvature.
- v_i : The relative velocity of the vehicle is given in the same polar coordinates as the position $(\dot{d}_i^E, \dot{\alpha}_i^E)$. These are converted to Cartesian coordinates in E as well and the speed is obtained as

$$v_x^E = v_e + \dot{d}_i^E \cos(\alpha_i^E) - \dot{\alpha}_i^E d_i^E \sin(\alpha_i^E) \tag{8}$$
$$v_y^E = 0 + \dot{d}_i^E \sin(\alpha_i^E) + \dot{\alpha}_i^E d_i^E \cos(\alpha_i^E) \tag{9}$$
$$v_i = \sqrt{(v_x^E)^2 + (v_y^E)^2} \tag{10}$$

- θ_i^L : Using velocity vector (v_x^E, v_y^E), the heading is found using (4):

$$\theta_i^L = \tan_2^{-1}(v_y^E, v_x^E) - tan^{-1}(\frac{\mathrm{d}y_{lane}^E}{\mathrm{d}\mathcal{X}^E}|_{x_i^E}) \tag{11}$$

  Where $\tan_2^{-1}$ is the four quadrant inverse tangent, such that all angles lie between $-\pi$ and π.
- l_i : The lane is found by taking $l_e + \text{round}(\frac{y_i^L}{w_l})$ where l_e is the ego lane and w_l is the width of the lane $(3.5m)$.
- h_i : Vehicle length is a given constant.
- w_i : Vehicle width is a given constant.

The state can be extended using road curvature and obstacle position estimates can be improved by considering distance along the road (x_i^L) instead of forward distance (x_i^E).

3.2 Action

Based upon the state, an action is chosen. The decision maker has to choose one action every time step of $1s$. An action is defined by the combination of a lateral maneuver

and a speed update. There are three lateral maneuvers (lane change left (LCL), lane keeping (LK) and lane change right (LCR)) and three speed values (increased speed (a), current speed (c) and reduced speed (d)). The increased and reduced speed are offset from the current speed by one quantized step ($1m/s$, see also Section 6.1).

Table 3.1: Actions as a combination of lateral maneuver and speed update

	LCL	LK	LCR
increased speed	LCL_a	LK_a	LCR_a
constant speed	LCL_c	LK_c	LCR_c
reduced speed	LCL_d	LK_d	LCR_d

This results in set A of all possible actions, with 9 different elements, as shown in Table 3.1. The decision maker has to select one action $a \in A$ for each timestep.

3.3 Expected behavior

As mentioned above, the focus is on safety. Therefore, the controller should produce safe behavior. Of course, the ego vehicle should also follow traffic rules and move towards a destination. Lastly, the controller should do all this without too much activity, as to create a comfortable user experience. From these demands, three criteria are selected to check the performance: *safety, liveness* and *activity*.

All of these measures are used to validate the behavior of the controller in simulations. A discrete time simulation model, that was developed at TNO, is used. Each simulation has a simulation time of $40s$, discrete sample time $t_s = 0.01s$ and thus number of samples $N = 4001$.

Safety is measured by the time to collision (TTC) as introduced by [34]. TTC is a common safety measure for two vehicles, it gives the time until a collision will occur given current velocities. A higher TTC suggests a safer situation. TTC is found using the method given in [35]. This method is given in Appendix A. For each sample of the simulation, the minimal TTC for all obstacle vehicles is determined. The TTC is bound on $[0s, 15s]$, since negative TTC and very distant collisions are not considered relevant. To get a single value, the root mean square (RMS) over all samples is used. By using RMS instead of mean, lower TTCs are penalized more.

$$safety = 15 - \text{RMS}(15 - TTC)$$

This measure is chosen such that a higher number means a safer controller and it is always positive.

Obviously, it is considered a perfectly safe solution not to move at all[5]. However, this is not a desirable solution. To prevent such solutions, *liveness* is introduced. It is defined by reaching the destination at some point. Because no explicit destination

[5]halting robot problem

is defined, it is measured by the distance traveled by the ego-vehicle d_{trav}. This is computed by approximately integrating the speed with the Riemann sum

$$d_{trav} = \sum_{k=1}^{N} v_e(k) t_s$$

where $v_e(i)$ is the ego speed at sample k.

Activity is measured by the number of lane changes. The low level controllers receive a binary signal (LC) to determine whether a lane change has to be executed, the rising edges on this signal are counted to get the number of lane changes.

$$n_{LC} = \sum_{k=2}^{N} \max\big(LC(k) - LC(k-1), 0\big)$$

Note that for this criterion, lower is better.

3.4 Decision maker architecture

As explained before, the decision maker selects an action in two stages, first safe actions are selected, next the best safe action is selected. Safety is checked by a supervisor, which is synthesized from models. An action is selected by searching for the optimal action in a Markov decision process (MDP). Using the current method, the supervisor cannot check long-term safety, this is further explained in Section 5.

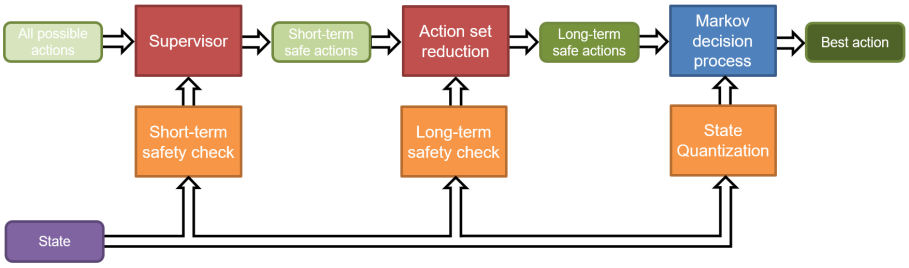

Figure 3.2: Controller design.

To overcome this challenge of the method, the safety check is split into two phases: a long-term phase and a short-term phase, as shown in Figure 3.2. Each of these steps, short-term safety, long-term safety and optimality, is executed by its own component of the decision maker. The safety-related components are shown in red, the optimality-related component is shown in blue. The orange components modify the state that was given above for other components. The arrows show the flow of information between components, what information is transferred between components is given in the rounded rectangles in the figure.

The supervisor only considers the safety of a single timestep, guaranteeing only the safety of the first action. After that, the *action set reduction* selects only the actions that also pass a long-term safety check. The safety checks are done by the orange

blocks. The two red components together produce the set of safe actions, from which an optimal action can be chosen. To find the optimal action, the state is quantized and modeled as an MDP, for which the optimal action will be computed. This is further explained in Section 4.

In Section 2 it was mentioned that an approach that can handle uncertainties in the state was needed. However, this was not chosen, because those methods require a significant amount of additional computational resources. Also, it is assumed the state can be measured with sufficient accuracy. Furthermore several assumptions on behavior of vehicles are made to reduce the uncertainties, as given in Section 6.

As explained above, the decision maker has to output an action every second. Therefore, the time for safety and optimality computations together is limited to $1s$. First safety computations are done, these are relative fast, as is explained in Section 5. The remaining time is used to find the optimal action.

In the figure, two inputs to the decision maker are shown on the lefthand side, namely the set of possible actions (A) and the current state of the vehicle and its environment. The set of all possible actions is fixed, the state changes over time, as described above. Based upon the state, one of the possible actions is chosen. This action is the output of the system.

4 Preliminaries

The controller design from the previous section mentions several methods that were used for the decision maker. This section elaborates on the modeling formalism that are used. In addition, the algorithms that were used are explained.

First, the discrete event modeling formalism, as is used for the safety stage, is given. The safety stage uses a supervisor which is automatically synthesized, how this is done is given next. Then the modeling formalism for the optimality component, i.e. a Markov decision process (MDP), is given. Next, it is explained how this model gives an action to execute. The algorithm that is used to find the optimal action is Anytime AO*. To be able to find the optimal action, the MDP is viewed as a tree graph.

4.1 Discrete event modeling

To be able to synthesize a supervisor, the vehicle is modeled as a discrete event system. This means that all dynamics are abstracted such that only discrete states and transitions between them remain. The plant, i.e., the system that is to be controlled, is divided into components for clarity. Each component is modeled as a finite state machine (FSM).

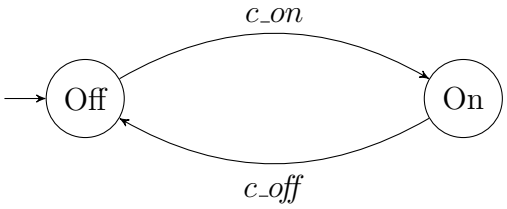

Figure 4.1: example of a finite state machine (FSM).

An FSM consists of a finite number of locations and transitions between those locations. An example of an FSM is given in Figure 4.1. Each location represents a state of the component, in the example, the states are 'On' and 'Off'. The initial location of the FSM is 'Off', as denoted by the incoming arrow. Each transition represents an event, such as turning on a system or selecting a maneuver. Transitions are denoted as arrows in the figure.

To obtain a model of the full system from its component models, all separate models are combined using *parallel composition*. Parallel composition consists of combining FSMs such that a transition for a shared event can only be taken if it is possible in all FSMs using that event. An explanation of this procedure can be found in [36, pp. 305-307].

It is also possible to include variables. Variables have an initial value and a finite domain. They can be updated upon taking a transition. Transitions can now be disabled for certain variable-based conditions, these are called guards. An FSM with variables is called an extended FSM. Note that combinations of locations and variable

values form states of an extended FSM.

4.2 Supervisory controller synthesis

Given the plant and the requirements, a supervisor can be generated using the supervisory controller synthesis algorithm. As mentioned in [37], a synthesized supervisor is guaranteed to have the following properties:

- The supervised plant will always adhere to the requirements.
- The supervisor will only disable controllable events. Uncontrollable events are never disabled. This is called controllability.
- From every state the supervised plant can be in, it is possible to reach a marked location. This is called non-blocking.
- The supervisor disables as little behavior as possible. This is called maximally permissive.

The explanation of the supervisor synthesis algorithm is based on [38]. For this explanation we consider a single extended FSM as a tuple $(\mathcal{L}, \mathcal{D}, \mathcal{E}, l_0, d_0, \mathcal{L}_m)$, where $\mathcal{L}$ is the set of all possible locations, $\mathcal{D}$ is the cartesian product of all domains of variables, $\mathcal{E}$ is the set of transitions, $l_0 \in \mathcal{L}$ is the initial location, $d_0 \in \mathcal{D}$ are the initial values of all variables and $\mathcal{L}_m \subseteq \mathcal{L}$ is the set of all marked locations. Each transition $e \in \mathcal{E}$ is a tuple (o_e, t_e, g_e, f_e), with origin $o_e \in \mathcal{L}$, target $t_e \in \mathcal{L}$, guard $g_e : \mathcal{D} \to \mathbb{B}$ and update $f_e : \mathcal{D} \to \mathcal{D}$. $\mathcal{E}$ can be partitioned into controllable events $\mathcal{E}_c$ and uncontrollable events $\mathcal{E}_u$. Transition e is possible when current location $l = o_e$ and $g_e(d) = true$ for current variable values d. When e is executed, l becomes t_e and d changes according to $f_e(d)$. The requirements are given as a set of forbidden locations $\mathcal{L}_f \subset \mathcal{L}$, as explained in [38] this can be done without loss of generality.

Supervisory controller synthesis is an iterative algorithm. In each iteration of the algorithm, some unwanted behavior is removed, until only desired behavior remains. The algorithm consists of the following steps:

- **Compute the non-blocking predicates**
  Non-blocking predicates, i.e., logical conditions, are assigned to locations to specify under which conditions a marked location can be reached.
- **Compute the bad state predicates**
  Bad state predicates extend the non-blocking predicates by checking if the enabled uncontrollable events do not lead to a bad state.
- **Update the guards on all edges**
  Based on the bad states, the guards on controllable events are updated, preventing the supervisor from going there.
- **Repeat if any guard changed**
  If no guards changed, synthesis is done. Otherwise, all steps above are repeated using the system with updated guards.

Each of these steps is explained below, except for the last step, which is trivial.

4.2.1 Non-blocking predicates

Each location $l \in \mathcal{L}$ is given a non-blocking predicate $N_l(d)$ for all $d \in \mathcal{D}$. $N_l(d)$ is computed iteratively, the value in iteration k is given by $N_l^k(d)$. The initial non-blocking predicates are $true$ for all marked locations $l \in \mathcal{L}_m$ and $false$ otherwise:

$$N_l^0(d) := \begin{cases} true, & \text{if } l \in \mathcal{L}_m \\ false, & \text{if } l \notin \mathcal{L}_m \end{cases} \qquad \forall l \in \mathcal{L}, d \in \mathcal{D}$$

Here, and in the rest of this document, := denotes assignment. The non-blocking predicates are updated iteratively using:

$$N_l^{k+1}(d) := N_l^k(d) \vee \bigvee_{\{e|o_e=l\}} \left[g_e(d) \wedge N_{t_e}^k\big(f_e(d)\big) \right] \qquad \forall l \in \mathcal{L}, d \in \mathcal{D}$$

Informally, a state is non-blocking if the location l is non-blocking, or if any non-blocking location is reachable through an enabled event for the variable values d. This process is repeated until no predicate changed in the current iteration.

4.2.2 Bad location predicates

Next, the bad location predicates $B_l(d)$ are computed in a similar manner to $N_l(d)$. A location is marked as a bad location if is forbidden, blocking or uncontrollable. An uncontrollable location has an uncontrollable event for which the target location is blocking or forbidden. Bad location predicates are computed iteratively, with the initial value given by:

$$B_l^0(d) := \begin{cases} true, & \text{if } l \in \mathcal{L}_f \\ \neg N_l(d), & \text{if } l \notin \mathcal{L}_f \text{ for the first iteration} \\ \neg N_l(d) \vee B_l(d), & \text{if } l \notin \mathcal{L}_f \text{ otherwise} \end{cases} \qquad \forall l \in \mathcal{L}, d \in \mathcal{D}$$

Note that $B_l(d)$ from a previous iteration of the supervisor synthesis algorithm is used. Therefore, special case is included for the first iteration.

The bad location predicates are updated iteratively:

$$B_l^{k+1}(d) := B_l^k(d) \vee \bigvee_{\{e|o_e=l, e\in\mathcal{E}_u\}} \left[g_e(d) \wedge B_{t_e}^k\big(f_e(d)\big) \right] \qquad \forall l \in \mathcal{L}, d \in \mathcal{D}$$

Informally, a location is bad when it was marked as bad before, or if an uncontrollable event $e \in \mathcal{E}_u$ is enabled ($g_e(d) = true$) which leads to a bad state with location t_e and variable values $f_e(d)$. If no bad location predicate changes, this process ends.

4.2.3 Update guards

The last step is updating the guards. Only the guards on controllable events can be adapted, for the supervisor should not disable uncontrollable events.

$$g_e(d) := g_e(d) \wedge \neg B_{t_e}\big(f_e(d)\big) \qquad \forall e \in \mathcal{E}_c, d \in \mathcal{D}$$

Each guard is tightened by including the bad location predicate for the target state of the corresponding event.

If no guard is updated, the synthesis is finished. Otherwise, the algorithm is executed again from the first step.

4.3 Markov decision process

The process of finding this optimal action is modeled as a Markov decision process (MDP). An MDP is used because the model includes future actions into the decision making process. In addition, it can handle stochastic changes in the state very well. In an MDP, actions are taken, each action incurs a rewards, the goal is to maximize the reward over future actions. This closely resembles the driving process, where, for example, doing lane changes reduces comfort, but it might speed up your journey by overtaking slow vehicles.

The following definition of an MDP is based on [39]. An MDP is defined as a 4-tuple $(S, A, T(s, a, s'), R(s, a))$, where S is any set of states, A is any set of actions, $T : S \times A \times S \to [0, 1]$, such that $\sum_{s' \in S} T(s, a, s') = 1$ is a transition function representing the stochastic change of state, based on the chosen action in a time step, $R : S \times A \to \mathbb{R}$ is a function representing the immediate reward (or cost) for taking an action, when in a state.

4.3.1 State

The state $s \in S$, captures relevant information about the current state of the system. After taking an action, the system transitions to a next (possibly different) state. The state should contain all information that is needed to select an action. This means the decision can be made without knowledge of past states and actions. The algorithm that is used to find the optimal action in this work requires that S is a finite set or countable infinite set. However, this is not generally necessary for MDPs, as explained in [40, p. 18].

4.3.2 Action

Where the state describes the system itself, the actions describe how to control this system. Action set A contains all possible 'commands' that can be given to a system. Not all actions have to be possible in every state, therefore, the set $A_s \subseteq A$ of available actions in state s is introduced.

4.3.3 Transition function

Whenever the decision maker takes an action a, the state s changes to a next state. How the state changes is dependent not only on a, but might also depend on factors that cannot be controlled, such as other road users. To include these factors, a

probabilistic relation is used. This relation is given by the transition function:

$$T(s, a, s') = \Pr\big(s(k+1) = s' \mid s(k) = s, a(k) = a\big) \tag{12}$$

With time step $k \in \mathbb{N}$. T 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. It is not necessary for all time steps to be equally long, or for different actions to take the same time. However, for simplicity all time steps in the model are assumed equal.

4.3.4 Reward

After every action, a reward is gained. How high this reward is, depends on s and a, it is given by reward function $R(s, a)$. Note that this is an immediate reward, it is only dependent on the present action. For this project, the reward was chosen such that is is always positive. It is also common that all rewards, or costs, are negative. However, in the general formulation of an MDP, the reward can have any value.

4.4 Finding an action

To specify what decisions should be taken, often a *policy* is defined. A policy $\pi : S \to A$ such that $\pi(s) \in A_s$ is a function specifying what action to take in each state[6]. A simple policy would be to always take the action with the highest reward.

$$\pi(s) = \underset{a \in A_s}{\operatorname{argmax}}\, R(s, a)$$

However, when making a decision in autonomous driving, it is necessary to look at long-term returns, multiple time steps ahead. Otherwise, the vehicle might choose actions that are very beneficial for a short time, but result in problems on a longer horizon. For example, a vehicle can accelerate to the desired speed quickly, because there is no vehicle in front of it. However, if there is a vehicle on the left lane that just started a lane change to the right, a large deceleration is required a short time later, which could have been prevented.

Since the future is not known beforehand and can only be predicted, the future rewards should be valued less than immediate rewards. This is expressed using a discount factor $\gamma \in [0, 1)$. For larger γ, future values are valuated more. Since the predictions are uncertain and several simplifications were used, γ is chosen to be 0.5.

If an arbitrary policy $\pi(s)$ is used, the action is fixed $a = \pi(s)$ and the outcome of $T(s, \pi(s), s')$ is only dependent on the states. Therefore we can define state value $V^\pi(s)$ as the expected returns when using policy $\pi(s)$ from s:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s')$$

[6] In [39] a richer definition of a policy is used, but that is not necessary here.

Note that this definition is recursive, which is forms the basis for dynamic programming (DP) algorithms, as mentioned in [40]. Using $V^\pi(s)$, we can also define the action value $Q^\pi(s,a)$, for taking a in s and using $\pi(s)$ afterwards.

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\pi(s')$$

This is the desire for taking an action, using the long-term (even infinite-term) returns. Note that if $a = \pi(s)$ is substituted, we can observe:

$$Q^\pi(s, \pi(s)) = V^\pi(s) \tag{13}$$

The best action is now clearly the action with the highest value $Q(s,a)$. So we can define the optimal policy π^*, which always returns the best action:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A_s} Q(s,a) \tag{14}$$

with

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V(s') \tag{15}$$

Where $Q(s,a)$ and $V(s)$ are the action value and state value for following $\pi^*(s)$, respectively. The optimal state value $V(s)$ is the expected return for the best action, so the highest value over all actions.

$$V(s) = \max_{a \in A_s} Q(s,a). \tag{16}$$

(15) and (16) are called the Bellman (optimality) equations, as introduced in [41]. For an explanation on these equations, see [39, Ch. 3]. Note that they do not depend on $\pi(s)$ anymore.

In the general definition of an MDP, (15) and (16) have an infinite recursion. In the algorithm that is used to find the best action, the values are computed only for a fixed number of recursions, i.e., the horizon H. This is similar to the receding horizon strategy used in model predictive control, e.g., in [22]. Where a set of future inputs is computed, and only the first input is executed on the system. In both methods, the problem of not knowing the entire future is dealt with by assuming there is nothing beyond the horizon.

4.5 Tree graph representation of an MDP

To find the optimal action, the MDP is viewed as a tree graph. Given the current state s_0 of the system, there is a set of actions A_s that can be taken, these branch out from s_0. Following from transition function T, each action has multiple possible future states. In these states, an action can be selected once again. It should be clear

that this creates an ever branching succession of actions and states, a structure which can be captured in a tree graph.

A tree graph is an undirected graph in which any two nodes are connected by a single path. Here, the tree G is explored incrementally, starting from an initial node called the *root node*. The root node is a tuple (s_0, H), where s_0 is the current state of the real system and H is the horizon, as defined in Section 4.4. This node is an *OR-node*, meaning a choice has to be made, i.e., the controller has to choose an action, presumably this is the optimal action. For each possible action, there is a child *AND-node* (a, s, τ) of OR-node (s, τ) where $a \in A_s$ is the action, s is the state from which a is taken and τ is the remaining horizon of the tree. In an AND-node, there is no choice, this differs AND-nodes from OR-nodes. AND-nodes (a, s, τ) get child OR-nodes $(s', \tau - 1)$, for which $T(s, a, s') > 0$. Since the controller cannot choose what state will follow when taking an action, actions get AND-nodes. When an OR-node has $\tau = 0$, i.e., the remaining horizon is 0, it is called a *terminal node* and no children are added. There are no terminal AND-nodes for the MDP.

As mentioned, the tree is explored incrementally. During one iteration, one OR-node (or state) is explored by adding all AND-nodes. All children of the AND-nodes are added to tree G as well. OR-nodes without children are called *tip nodes*. It is possible that there are OR-nodes with $\tau > 0$, but which do not yet have children.

A value can be associated to each node. These values were chosen such that they represent the values given by the Bellman equations for Q and V, as given in (15) and (16). The value of a terminal node is 0. The value of a non-terminal OR-node is equal to highest value of all child AND-nodes.

$$V(s, \tau) = \begin{cases} 0, & \text{if } \tau = 0 \\ max_{a \in A_s} Q(s, a, \tau), & \text{otherwise} \end{cases} \tag{17}$$

The value of an AND-node (a, s, τ) is defined as the reward for taking the action plus the sum of all values of children $(s', \tau - 1)$ weighted by the probability of occurring and discount factor γ:

$$Q(s, a, \tau) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s', \tau - 1) \tag{18}$$

Once again, note the similarity to the bellman equations (15).

Given these values, the best partial graph G^* can be defined. G^* is defined here as the subgraph of G in which, starting from the root node, only AND-nodes with the highest value are selected, and all subsequent OR-nodes are selected. The optimal action is now the action that is taken from the root node.

4.6 Anytime AO*

The best action is found through a tree search algorithm called Anytime AO*, as introduced in [42]. Tree search algorithms are used to find the optimal path through a

tree graph. 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 3.4. In Section 5, it is mentioned that safety computations take much less than $1s$.

The values of unexplored states are estimated to decide what part of the state space should be explored, the estimation function $h(s, \tau)$ is called a heuristic. The function h should be chosen such that it accurately approximates $V(s, \tau)$ from (17), with little computational complexity. It is undesired to use the Bellman equation (16) directly, because it has a high computational complexity.

Usually, search algorithms require admissible heuristics, this means the heuristic always returns a higher reward than actually is received upon taking the action. The Anytime AO* algorithm also works for non-admissible heuristics to predict the value of an action. This means it is possible to use functions that estimate the value, but not always over-estimate it, as is required for an admissible heuristic. Every time a state is explored, it's value is computed using (17) and (18), instead of estimated using h. Therefore, the values for nodes at or near the root become increasingly accurate as more states are explored.

The Anytime AO* algorithm works using a receding horizon strategy. Not only the current action is considered, but also several actions into the future. After the predefined time, only the first action is executed. For the next action, the algorithm is executed again.

The solution is found iteratively by expanding (or, exploring) a non-terminal tip node in G. With a probability of $1 - p_{opt}$, a node is selected to be in G^*, otherwise it is selected to be outside of G^*. $0 \leqslant p_{opt} \leqslant 1$ is a parameter which determines the trade-off between exploring actions that are not in G and refining the current value estimate of the best action.

Once a node is selected, it is expanded in 5 steps. A visual representation of this process, as executed on the root node (s_0, H) is given in Figure 4.2. Each step is explained below.

1. **Add AND-nodes**
   For each possible action $a \in A_s$ add node (a, s, τ) as child of (s, τ). In the figure actions a_1, a_2 and a_3 are possible for the root node.
2. **Add OR-nodes**
   For each $a \in A_s$ and s' such that $T(s, a, s') > 0$, add node $(s', \tau - 1)$ as child of (a, s, τ). In the figure, 7 nodes are added, the probabilities are given next to the edges. Note that the actual amount of nodes will be much larger. This is the main reason for using a search algorithm instead of dynamic programming.
3. **Initialize values**
   For each node $(s', \tau - 1)$ that was added, estimate the value $V(s', \tau - 1) := h(s', \tau - 1)$, or $V(s', \tau - 1) := 0$ if it is a terminal node (i.e., $\tau = 0$). The values are given in blue in the figure, they are found by evaluating $h(s, \tau)$ for each node.
4. **Update ancestors**
   Recursively update the values for all parents according to (17) and (18). Q and V

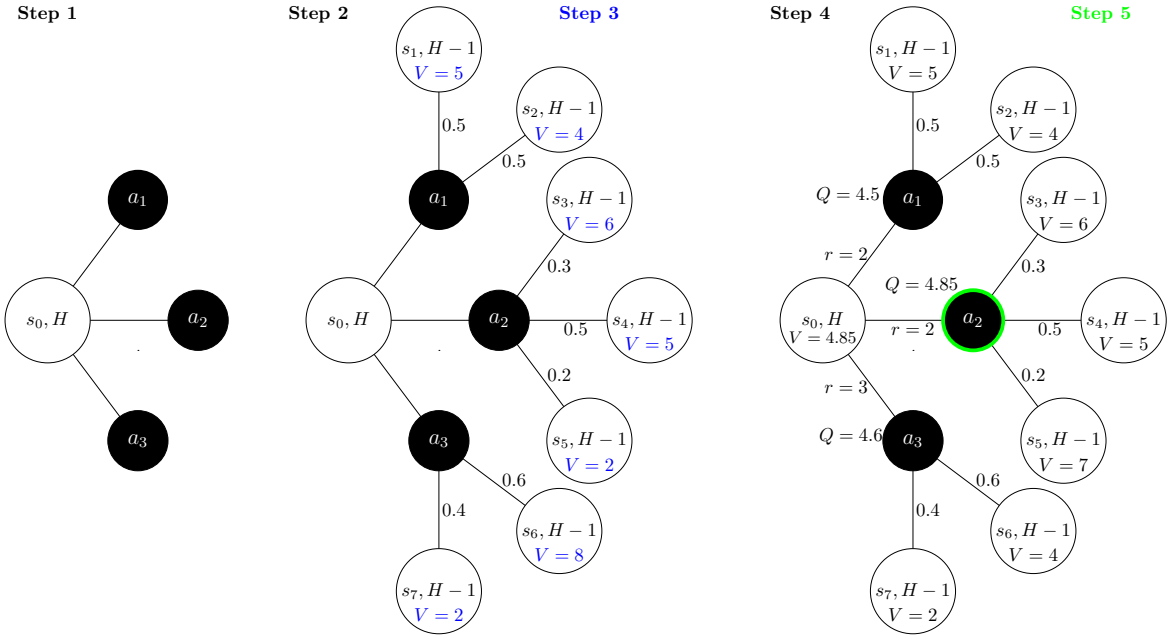

Figure 4.2: visualization of node expansion. OR-nodes are shown in white, AND-nodes are shown in black. Step 3 is given in blue in the center graph. Step 5 is given in green in the righthand graph.

determine the best action and the best partial graph. In the figure, the immediate reward for each action is given by r. Here, Q is added for each action, V is only updated for the root node. Note that the immediate reward is highest for a_3, whereas a_2 has the highest value for Q.

5. **Mark best action**
   In each ancestor OR-node (s, τ), mark the best action a^* for which $Q(s, a, \tau) = \max_{a \in A_s} Q(s, a, \tau)$, note that this value is equal to $V(s, \tau)$, as defined above. In the figure, a_2 is marked in green.

Once the node is expanded, the best partial graph is updated by following the marked actions. This sequence of finding a node, expanding, and updating the best partial graph is repeated until no more non-terminal tip nodes exist or the time runs out. If no more non-terminal tips nodes exist, the tree is fully explored and the heuristic is no longer used. Therefore, the best action that was found, is the optimal action for this tree. The best action is given at any time by the marked action in the root node. Whenever the value $V(s, \tau)$ of a leaf node (s, τ) is required for computations, the heuristic is computed as explained in step 3, and the value is averaged with previous values, which might be different.

How the next node for expansion is selected is not fixed for the algorithm. In [42], it is stated that the performance can significantly be improved if a good criterion is used. For the decision maker, the next node is selected by taking an action at random and then selecting a successor state based on the probability of ending there.

5 Safety stage

The first stage of the decision maker is the safety stage. The safety stage consists of a supervisor which checks the short-term safety and *action set reduction* to check the long term safety. The supervisor is automatically synthesized from plant models and requirement models, as introduced by [37]. In this section, first the plant components are introduced, next the requirements are given. From these plant and requirements, a supervisor is generated using the synthesis algorithm. As explained above, the supervisor only checks the safety of one action. At the end of this section, it is explained why a supervisor cannot check long-term safety and how action set reduction is used to overcome this challenge.

5.1 Plant components

The system is modeled using two plant components: one for enabling the autonomous driving (AD) and one for modeling the possible actions.

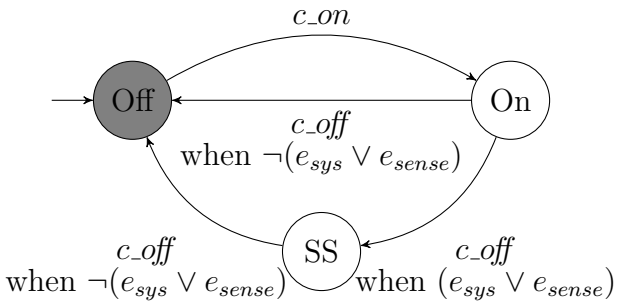

Figure 5.1: Plant model for the autonomous driving (AD) capabilities. e_{sys} and e_{sense} are error signals that are provided to the supervisor.

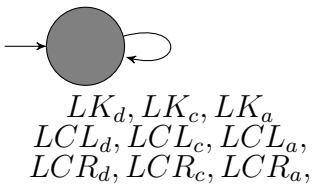

Figure 5.2: Plant model for all possible actions in the decision maker.

The autonomous driving behavior is modeled using a three-state automaton, as shown in Figure 5.1. The leftmost location represents the AD capabilities being off, this location is also marked, as it is considered idle. Marked locations are needed for the synthesis algorithm [38], they are used to denote idle states or states where the supervisor can safely end. The leftmost location represents the AD capabilities being on, it is not marked. The bottom location is a safe state (hence SS), which is present to allow for different behavior when something is wrong. Note that this location is not marked, for the system is not yet safe in this location. This location is used to allow the vehicle to get to safety.

The automaton can switch between these locations using controllable events *c_on* and *c_off*. *Controllable* means the supervisor can choose whether to execute this event, as opposed to *uncontrollable* events, with cannot be controlled by the supervisor, such as a sensor turning on or off. There are two transitions leaving location *On* for the *c_off* event, with different guards. The plant goes to a safe state whenever the supervisor is

notified of an internal error e_{sys} or unexpected measurement e_{sense}. When the problem is resolved, the system turns off as normal, until the user turns it back on. Note that these signals should represent errors that are not resolved by low level components, such as the camera turning off.

All possible actions, as given in Table 3.1 are represented using an automaton with only one location. From this location, each action is represented as a controllable event loop. This is shown in Figure 5.2, for compactness, all nine loops are depicted as a single loop.

5.2 Requirements

Using these plant models, the requirements can be formulated. AD can be enabled when lane keeping (LK) is possible and the driver wants to activate AD. Once AD is activated, different conditions are used for different maneuvers. LK is enabled when the current lane is detected by the lane tracking system, and the chosen action is safe. LCR is enabled when the current lane is known, the right lane is detected and the chosen action is safe. LCL is enabled when the current lane is known, the left lane is detected and the chosen action is safe.

Safety is measured here by taking the time to collision (TTC). It is computed in the same manner as for measuring safety in Section 3.3. A higher TTC suggests a safer situation. The chosen speed update for an action is included by assuming the change in speed is immediate, such that the new speed can be used for TTC computations. Here, the TTC is bound on $[0s, 6s]$, the upper bound was chosen because a maneuver takes approximately six seconds.

An action is safe if the TTC exceeds a certain threshold for all vehicles. Therefore, only the lowest TTC for all obstacle vehicles has to be considered.

5.3 Supervisory controller synthesis

Given the plant and the requirements, a supervisor can be generated using the supervisory controller synthesis algorithm. As mentioned before, a synthesized supervisor is guaranteed to always adhere to the requirements, it is controllable, non-blocking and maximally permissive.

When applying supervisor synthesis to the models given in this section, no changes are made to the system. This means that the synchronous product of the plant and requirements already satisfies the properties mentioned above. Clearly, the system is always controllable, for there are no uncontrollable events. Synthesis shows that the requirements are non-blocking. For large systems, methods exist to check whether synthesis is needed to guarantee these properties. However, this is not necessary for small systems, where the synthesis algorithm finishes within seconds. For those systems, a supervisor can directly be synthesized.

5.4 Horizon of safety check

First, the supervisor is used to check the safety of a single action, which takes one time step ($1s$).

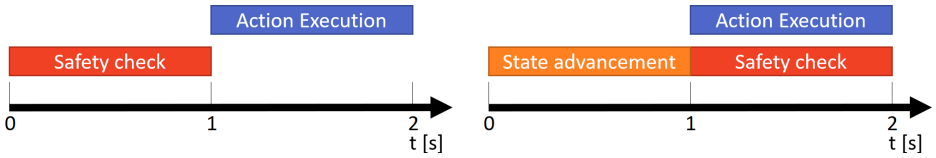

Figure 5.3: Horizon of safety check. Left: old check. Right: new check.

However, the action that is selected by the decision maker, is executed after the current time step, as shown on the left hand side in Figure 5.3. Therefore, the safety check is computed for actions one time step further, as shown on the right hand side in the figure. Given that the action for the first time step is already fixed and should be safe, this additional time step is exactly the same for all possible actions that are considered. Therefore, the state is advanced by 1 time step of predictions and that state is used for the safety computation. This happens in the orange short-term safety check block in Figure 3.2. Note that the safety computations take much less than $1s$[7] all remaining time is used for finding the best action. The error signals e_{sys} and e_{sense} are used directly, because errors need an immediate response.

5.5 Action set reduction

The algorithm for synthesis in Section 4.2 allows for all variable types. However, current tools[8] that implement this algorithm only allow boolean or integer variables in updates. In addition, only simple arithmetic functions can be used, such as addition and subtraction. Because of that, the dynamical behavior of the continuous state cannot be included in the supervisor. As explained before, this challenge is overcome by only considering short-term safety, i.e., choosing only a single action. By checking the safety for only one time-step, the controller does not need real valued variables, i.e., the state, in updates. Long-term safety is handled by the *action set reduction* component. This way, long-term safety is considered before choosing the optimal action.

The action set reduction component evaluates a long-term safety criterion for all short-term safe action. All actions that do not meet this criterion are removed. The criterion that is used is a threshold on the TTC, as is also used for short-term safety. In future works, a more fitting method should be selected. In addition, lane changes to a lane where 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. By removing these actions, the search space for finding the optimal action can be reduced significantly. The remaining actions are considered safe and will be used to find the optimal action.

[7] It is estimated that safety computations take up to $1ms$.
[8] E.g., CIF3 [43] or supremica [7].

6 Optimality stage

The second stage is the optimality stage. Its purpose is to find the optimal action from the set of safe actions. The process of finding this optimal action is modeled as a Markov decision process (MDP). An MDP is defined as a 4-tuple $(S, A, T(s, a, s'), R(s, a))$, as was explained in Section 4.3. In this section, each of the elements of the model that was used are given. After that, it is explained how the algorithm to find the best action was implemented.

6.1 State

The state $s \in S$ of the MDP is inferred from the state representation $\mathbf{X}$ of (3). The state contains information on the position and velocity of the ego vehicle and n_v obstacle vehicles. To consider maneuver continuation as a condition for optimal behavior, the previous action is also included.

As mentioned above, the algorithm requires that the state contains no continuous variables. Therefore, the variables are quantized. This is similar to the work on qualitative modeling by Lunze, e.g. in [44]. There, the continuous state is partitioned (quantized) to obtain different regions that require different control strategies. These regions are not necessarily regularly spaced, i.e., like the grid that is used for this project. In addition, the signals can be discretized by only considering transitions between regions, as opposed to using a fixed time step, as is done here. Further studies are required to find out which strategy suits the problem best.

Each of the continuous variables in $\mathbf{X}$ is quantized, the quantization steps are given in Table 6.1. The extremal continuous values that were used are also given in the table.

Table 6.1: state quantization steps

Variable	x_i	y_i	v_i	θ_i
Quantization size	$8m$	$0.5m$	$1m/s$	$0.01rad$
Minimal value	$-152m$	$-2.0m$	$0m/s$	$-0.11rad$
Maximal value	$152m$	$2.0m$	$42m/s$	$0.11rad$

All variables except θ_i are quantized by flooring, θ_i is quantized by rounding. If θ_i was floored as well, the transition function would be biased towards the right, as is explained in Section 6.3. Let the quantized sets be defined for position $\bar{X} \subseteq \mathbb{Z}$ and $\bar{Y} \subseteq \mathbb{Z}$, speed $\bar{V} \subseteq \mathbb{N}_0$, heading $\bar{\theta} \subseteq \mathbb{Z}$, and lane $L \subseteq \mathbb{N}_0$, which is not quantized any further. Note that vehicles moving in reverse is assumed not to occur, but standstill is allowed, e.g., due to traffic jams. The set of all states is now a Cartesian product of these sets and the action set:

$$S = (\bar{X} \times \bar{Y} \times \bar{V} \times \bar{\Theta} \times L)^{n_v+1} \times A. \tag{19}$$

As the state space is large(10^{17} states for 3 vehicles), the MDP is solved online, i.e., the optimal action is found for the current state, at every time step. This means that

only states (and values) that are reachable within the finite horizon are considered when finding an action. Searches start at the current state $s_0(k)$ of the system.

6.2 Action

The decision maker produces a decision every time step of $1s$, except for emergency situations. The low level controllers require an input every $0.01s$, this is realized by holding the last decision until the next one is found. The actions used in the MDP are given in Table 3.1. $A_s \subseteq A$ is defined as the set of available actions in state s, because not all actions are available in every state, as some actions might be unsafe. For the first time step, A_s is given by the action set refinement from Section 5. 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.

6.3 Transition function

The transition function for the MDP is based on simplified vehicle dynamics and the chosen maneuver. It is stochastic due to uncertainties introduced by state quantization. Sensor uncertainties can be included as well. Also, road-user behavior can be captured in the stochastic transition function, if modeled in a stochastic fashion.

The transition function is used to compute the probabilities of reaching each possible configuration of vehicles, given an action. The ego heading and speed change based on the chosen action. Obstacle speed is assumed to be constant and heading (with respect to the lane) is assumed to be constant and 0.

To predict the position after one time step, the vehicle is assumed to be a point mass with constant velocity. A more accurate representation could be obtained using a bicycle model, where the vehicle is a rigid body with wheels, like a bicycle. This model can more accurately represent vehicle maneuvers, as long as no tight turns are involved. For an explanation of the model see, e.g., [45, pp. 613-623]. However, the bicycle model is computationally more intensive. In addition, the increase in accuracy for highway scenarios is not assumed significant enough to justify using the bicycle model.

Table 6.2: Heading change for lane keeping (left) and lane changing (right)

LK	$\bar{\theta} < 0$	$\bar{\theta} = 0$	$\bar{\theta} > 0$
$\bar{y} \geqslant 1$	$+0$	$+0$	-1
$-1 \leqslant \bar{y} \leqslant 0$	$+1$	$+0$	-1
$\bar{y} \leqslant -2$	$+1$	$+0$	$+0$

	LCL	LCR
$\bar{y} > \bar{y}_{LK}$	$+1$	$+1$
$-\bar{y}_{LK} \leqslant \bar{y} \leqslant \bar{y}_{LK}$	$+1$	$q-1$
$\bar{y} < -\bar{y}_{LK}$	-1	-1

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 lane center, as shown on the lefthand side in Table 6.2. In the table and in the rest of

this section, $\bar{(\cdot)}$ denotes a quantized value. 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, as shown on the righthand side in the table. $\bar{y}_{LK}$ is the maximal quantized value of y that is expected during lane keeping. 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.

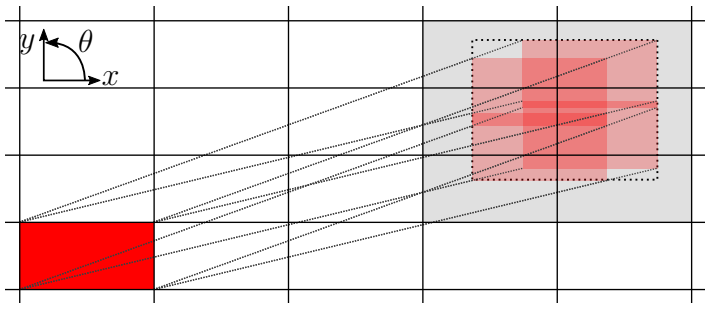

Figure 6.1: Computation of reachable region.

A visualization of the transition function is given in Figure 6.1. The grid represents the quantized intervals of the continuous x- and y-values. The origin cell (dark red) shows the initial position of the ego-vehicle. By combining all possible speeds with all possible headings for this quantized state, a region is found where the ego-vehicle can end up in one time step.

In Figure 6.1, the possible end positions for the highest and lowest values for speed and heading are shown in light red. All possible quantized end positions are given in light grey. It can be seen that from a single quantized position, several end positions can be attained, this produces stochastic transitions.

To reduce the complexity of the reachable region, it is assumed the region is rectangular, such that only the highest and lowest lateral and longitudinal positions have to be computed. This means the real probabilities for partially covered regions will be at most the computed probabilities. There is a small possibility of a grid cell in a corner of the reachable region being included while it is actually not reachable. It is assumed this difference in real and computed probabilities is negligible.

The positions are now calculated by the following steps:

1. **Find the extremal continuous values corresponding to the quantized state.**
   Each state variable was quantized by flooring, so the minimum value can be found by multiplying the quantized value with the quantization step size. This value plus one step size gives the maximum value.

$$x_{min} = \bar{x} \cdot dx$$

$$x_{max} = \bar{x} \cdot dx + dx$$

   $d(\cdot)$ is the quantization step for each variable. This is done for each of the state variables x, y, v and θ.

2. **Find the extreme (continuous) lateral and longitudinal values that are reachable.**

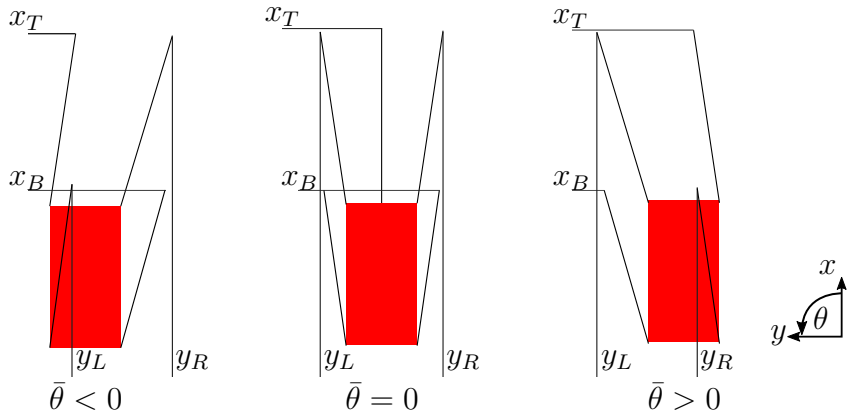

Figure 6.2: Extreme values for different $\bar{\theta}$.

The extreme values are dependent on the sign of $\bar{\theta}$, because the largest absolute heading determines the extreme values. In Figure 6.2 a visualization of the extreme values is given. Note that the figure is not to scale. x_B and x_T are the nearest and furthest reachable longitudinal positions, respectively. y_L and y_R are the leftmost and rightmost reachable lateral positions, respectively. How these values are obtained is given below.

If $\bar{\theta} > 0$ and hence $|\theta_{min}| < |\theta_{max}|$,

$$\begin{aligned} x_B &= x_{min} + v_{min} \cdot \cos(\theta_{max}) \\ x_T &= x_{max} + v_{max} \cdot \cos(\theta_{min}) \\ y_L &= y_{min} + v_{min} \cdot \sin(\theta_{min}) \\ y_R &= y_{max} + v_{max} \cdot \sin(\theta_{max}) \end{aligned}$$

If $\bar{\theta} < 0$ and hence $|\theta_{max}| < |\theta_{min}|$,

$$\begin{aligned} x_B &= x_{min} + v_{min} \cdot \cos(\theta_{min}) \\ x_T &= x_{max} + v_{max} \cdot \cos(\theta_{max}) \\ y_L &= y_{min} + v_{max} \cdot \sin(\theta_{min}) \\ y_R &= y_{max} + v_{min} \cdot \sin(\theta_{max}) \end{aligned}$$

If $\bar{\theta} = 0$ and hence $\theta_{max} = -\theta_{min} = \frac{d\theta}{2}$,

$$\begin{aligned} x_B &= x_{min} + v_{min} \cdot \cos(\theta_{max}) \\ x_T &= x_{max} + v_{max} \cdot \cos(0) \\ y_L &= y_{min} + v_{max} \cdot \sin(\theta_{max}) \\ y_R &= y_{max} + v_{max} \cdot \sin(\theta_{min}) \end{aligned}$$

Where $d\theta$ is the step size for quantization of θ. These values give the dotted black rectangle in upper right corner of Figure 6.1. Note that for $\bar{\theta} = 0$, the reachable y values $[y_L, y_R]$ are centered on the original value of y, as can be seen in Figure 6.2. This is because $\bar{\theta}$ was rounded when quantizing. Otherwise, θ_{min} would have been 0, which results in a reachable y range that extends to the left of the original value, similar to $\bar{\theta} > 0$ in the figure.

3. **Estimate the probabilities of ending in a region.**

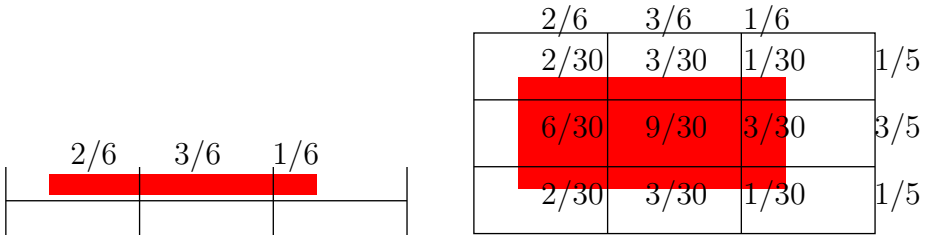

Figure 6.3: Probabilities for one direction (left) and region (right).

The lateral and longitudinal probabilities are computed separately. Each reachable quantized value is assigned a probability proportional to the reachable part, as shown on the lefthand side of Figure 6.3. This is done the same for lateral and longitudinal positions. Afterwards, the probabilities are multiplied to get the probability of ending up in each region. This is shown on the righthand side of the figure. The values outside the grid show the lateral and longitudinal probabilities. When these are multiplied, the values inside the grid are obtained. These values are the probabilities for each cell.

6.4 Reward

The reward function is a linear combination of chosen features R_i, i.e. functions of the state representation, such as speed or time to collision.

$$R(s, a) = \sum_{i \in \mathcal{R}} \lambda_i R_i(s, a) \tag{20}$$

Where $\mathcal{R}$ is the set of feature indices and λ_i are the weights corresponding to each feature. The weights for each feature are chosen or can be learned from data through, e.g., inverse reinforcement learning, as was done by [15]. The used features are given below.[9] Note that all reward features are bound on the domain $[0, 1]$.

- **Speed**
  Deviations from reference speed v_{ref} are penalized quadratically, to motivate

[9]These functions are defined for a single obstacle vehicle. For multiple vehicles, the sum, min or max could be taken.

overtaking slow vehicles and yielding to faster vehicles. A quadratic penalty was chosen to give a larger penalty for larger deviations.

$$R_v = \left(\frac{\bar{v}_e - \bar{v}_{ref}}{\bar{v}_{ref}} \right)^2$$

- **Lane change and acceleration penalty**
  Taking unnecessary actions is penalized for driver comfort and fuel efficiency. A reward is given if the chosen action is lane-keeping or if the chosen speed is constant.

$$R_{aL} = \begin{cases} 1, & \text{if } a = LK_* \\ 0, & \text{otherwise} \end{cases}$$

$$R_{aV} = \begin{cases} 1, & \text{if } a = *_c \\ 0, & \text{otherwise} \end{cases}$$

  Note that $*$ is used to denote three actions $a \in A$ with the given longitudinal or lateral component.
- **Safety**
  The time to collision is used to give preference to behavior that is even safer than prescribed by the safety components. To increase the speed of computations, TTC is approximated here. It is approximated by only considering longitudinal movement.

$$\begin{aligned} \Delta v &= (\bar{v}_o - \bar{v}_e - \delta v) dv \\ \Delta x &= (\bar{x}_o - \bar{x}_e) dx + \Delta v \\ TTC &= -\frac{\Delta x}{\Delta v} \end{aligned}$$

  Where δv is the change in speed following from the chosen action, $(\cdot)_e$ corresponds to ego vehicle variables and $(\cdot)_o$ is used for obstacle vehicle parameters. However, even when longitudinal TTC is low, the lateral distance might be large enough to avoid a collision. This is measured by distance upon collision DUC:

$$\begin{aligned} \Delta y &= (\bar{y}_o - \bar{y}_e) dy + w_l (l_o - l_e) \\ \Delta \theta &= (\bar{\theta}_o - \bar{\theta}_e - \delta\theta) d\theta \\ DUC &= \Delta y - \Delta\theta \Delta x \end{aligned}$$

  Where $\delta\theta$ is the change in heading following from the chosen action. Note that the linearization $\tan(\theta) \approx \theta$ is used, which is valid for small angles. The safety reward is then computed for each existing vehicle ($\bar{v} > 0$) by:

$$R_s = \begin{cases} \frac{TTC}{TTC_{max}}, & \text{if } \Delta x < x_{TTC} \wedge DUC < y_{TTC} \\ 1, & \text{otherwise} \end{cases}$$

Where $TTC_{max} = 15s$ is the maximal TTC that is considered, it is used for normalization. $x_{TTC} = 40m$ is the maximal longitudinal distance for which TTC is used and $y_{TTC} = 2m$ is the maximal lateral distance for which the vehicles are assumed to collide, it should be equal to or greater than the vehicle width.

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

$$R_l = \begin{cases} 1, & \text{if } l = 1 \\ 0.5, & \text{if } l > 1 \wedge a = LCR_* \\ 0, & \text{otherwise} \end{cases}$$

- **Urgency**
  Urgency is enforced by penalizing negative deviation from the desired speed. Note that urgency, contrary to speed, is penalized linearly. This was done because the vehicle should not show increasingly hasty behavior if the speed decreases.

$$R_u = \min\left(\frac{\bar{v}_e}{\bar{v}_{ref}}, 1\right)$$

- **Maneuver continuation**
  Similar to the unnecessary action penalty, a reward is given if a maneuver is continued. Also, a minor reward is given when starting or finishing a maneuver. Note that this reward does not explicitly reward completing a lane change.

$$R_c = \begin{cases} 0.5, & \text{if } a = LK_* \vee a_{prev} = LK_* \\ 1, & \text{if } a = LCR_* \wedge a_{prev} = LCR_* \\ 1, & \text{if } a = LCL_* \wedge a_{prev} = LCL_* \\ 0, & \text{otherwise.} \end{cases}$$

- **Lane centering**
  A penalty is given for deviating too far from the lane center, to motivate finishing a maneuver before switching back to LK.

$$R_{lc} = 1 - \min\left(0, \frac{(|\bar{y}| - \bar{y}_{LK}) \cdot dy}{\frac{w_l}{2}}\right)$$

  Where $\bar{y}_{LK}$ is the limit for LK as above, dy is the quantization step size for y and w_l is the lane width. Note that for smal values $|\bar{y}| < \bar{y}_{LK}$, this reward does not change, to allow for some deviations during lane keeping.

All these rewards are combined using weights λ_i, these were manually tuned by trial and error in simulations. No formal procedure was found to tune the weights. Some relations between individual weights were found, e.g., between activity penalty λ_{aL} and maneuver continuation λ_c.

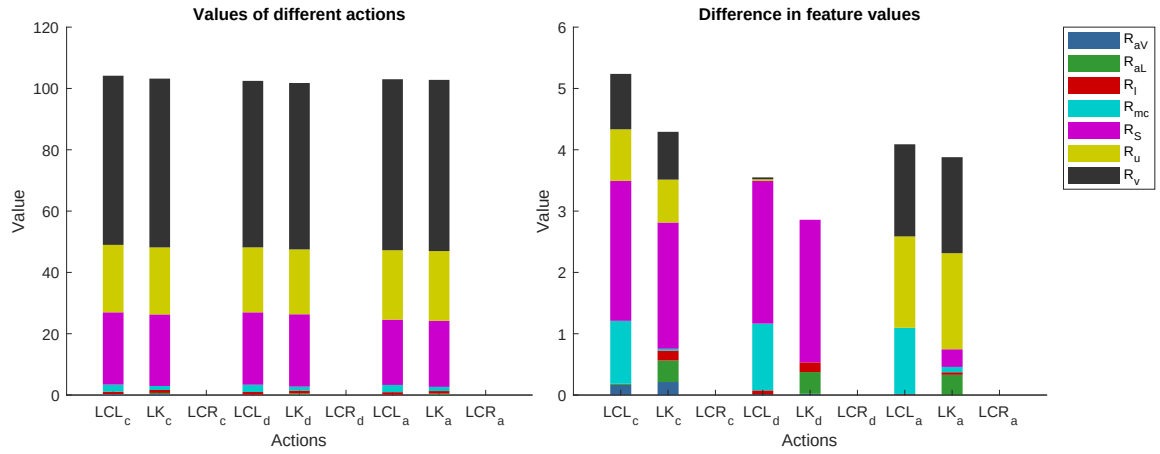

Figure 6.4: Values for different actions. Left: values for different actions. Right: difference between features.

An example of the decision making process is visualized in Figure 6.4. On the left hand side of the figure the value $Q(s, a)$, from (15), is given for each action from a certain state s. Each individual bar is colored according to the portion of the value that originates from different features. Note that R_s, R_u and R_v have large contributions to the value, whereas the others do not. It is not easily seen that LCL_c is the best action. The values for each feature are given in Appendix B.

To more clearly show the impact of all features, only the difference between different actions is shown on the right hand side of the figure.

$$Value = \sum_{i \in \mathcal{R}} \left(\lambda_i R_i(s, a) - \min_{a' \in A_s} \lambda_i R_i(s, a') \right)$$

It can best be seen for R_s in LCL_a, that action had the lowest contribution for R_s, and now appears as 0. Note that the contribution of R_{mc} has a significant impact on the best action (LCL_c), whereas it was barely visible in the left hand figure.

6.5 Anytime AO*

To find the optimal action, Anytime AO* is used, as explained Section 4.6. Several design choices were made in the implementation of the algorithm.

As explained in Section 4.6, the algorithm iteratively selects and expands nodes in the tree representation of the MDP. Expanding nodes happens in 5 steps:

1. adding AND-nodes (for actions);
2. adding OR-nodes (for states);
3. estimating V for new OR-nodes, using a heuristic;
4. update the values Q and V of ancestors, using (18) and (17);
5. mark the best action for all ancestors.

Node selection is done based upon the probability of encountering a state using randomly selected actions.

The heuristic that is used is based on the immediate reward for taking the same action. That reward is gained for every future time step.

$$h(s,\tau) = R(s,a_{prev}) + \gamma R(s,a_{prev}) + \gamma^2 R(s,a_{prev}) + \ldots + \gamma^{\tau-1} R(s,a_{prev})$$

This heuristic was chosen because it approximates the value using the actual reward function R, without having to compute all future values.

To reduce the memory footprint, the expanded tree G is stored separately from the unexpanded tip nodes $\mathcal{T}$. The tip nodes are triples (s, V, p). They are much smaller than expanded nodes. All values are stored either in a tip or node. Therefore, only values that are needed for selecting a tip are stored in tips. All other values are stored in expanded nodes.

Nodes in G are stored as a 7-tuple $(\tau, s_p, a_p, a^*, A_s, [S'_a], [Q_a])$. Here τ is the remaining time horizon, s_p is the parent node and a_p is the parent action, these are all related to the OR-node. The AND-nodes are stored implicitly as a variable inside the OR-nodes. This was done because They are expanded at the same time as OR-nodes. The AND-node contains set $[S'_a]$, of all reachable states for each allowed action $a \in A_s$, set $[Q_a]$, with $Q_a = Q(s,a)$, for each allowed action $a \in A_s$.

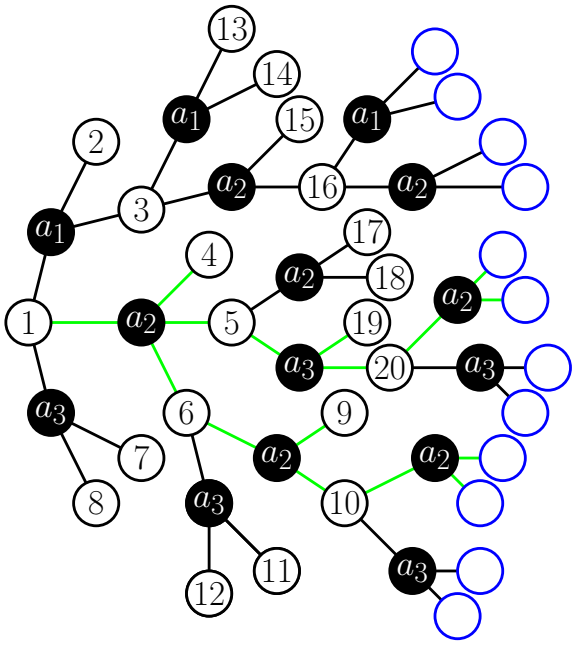

Figure 6.5: Visualization of G. OR nodes in white, AND nodes in black, terminal nodes in blue. Green arrows denote G^*.

Table 6.3: Data storage for expanded nodes in the implementation, A_s and $[Q_a]$ are not shown. IDs correspond to the numbers in Figure 6.5.

Nodes

ID	τ	s_p	a_p	a^*	S'_1	S'_2	S'_3
1	3			2	2,3	4,5,6	7,8
6	2	1	2	2		9,10	11,12
3	2	1	1	2	13,14	15,16	
16	1	3	2	2	$\emptyset$	$\emptyset$	
10	1	6	2	2		$\emptyset$	$\emptyset$
5	2	1	2	3		17,18	19,20
20	1	5	3	2		$\emptyset$	$\emptyset$

Table 6.3 shows how the structure of a tree G, as shown in Figure 6.5 is stored. Note that not all values in expanded nodes are shown, only those related to the structure of the tree. In the figure, OR- and AND-nodes are shown in white and black respectively,

terminal nodes are shown in blue. After seven iterations of the algorithm, there are seven nodes and many times more tip nodes than iterations, 20 for this example. For the actual MDP, as given above, there are typically 6 actions leaving a state. And there are between 100 and 200 states that follow a single action[10].

```
1  G := ∅
2  tip := (s_0, 0, 1)
3  T := {tip}
4  A_s := A_ASR
5  τ := H
6  s_p := ∅
7  a_p := ∅
8  loop 500
9      ([Q_a], [S_a], T) := add_nodes(A_s, T)
10
11     V⁻ := tip.V
12     V⁺ := max_{a∈A_s}(Q_a)
13     a* := argmax_{a∈A_s}(Q_a)
14     tip.V := V⁺
15     update tip in T
16     add (τ, s_p, a_p, a*, A_s, [S'_a], [Q_a]) to G
17
18     p := tip.p
19     while s_p ≠ ∅
20         P := G(s_p)
21         P.Q_{a_p} := P.Q_{a_p} + γp(V⁺ − V⁻)
22         P.a* := argmax(P.Q_a)
23         update P in G
24         V⁻ := T(s_p).V
25         V⁺ := max(P.Q_a)
26         if V⁻ = V⁺
27             break
28         T(s_p).V := V⁺
29         p := T(s_p).p
30         s_p := P.s_p
31         a_p := P.a_p
32
33     do_optimal := true with probability p_opt, false otherwise
34     (tip, A_s, τ) := select_tip(G, T, do_optimal)
35     if no tip was found
36       (tip, A_s, τ) := select_tip(G, T, ¬do_optimal)
37     if no tip was found
38       break
39
40 return G(s_0).a*
```

Algorithm Section 1: Pseudo code for the implementation of AO*

The implemented algorithm is shown in Algorithm Section 1. Lines 1 through 8 are initialization. Note that the expanded tree G is initially empty whereas $\mathcal{T}$ contains one tip, which represents the initial node. `tip` is the tip that is currently being expanded, A_s and d are the allowed actions and depth of `tip`. s_p and a_p are its parent state and action. Note that these are both empty for the root node, for it has no parents.

The algorithm is repeated until the entire tree is expanded or when the time runs out. As the duration of computations cannot be measured during simulation, a fixed number of iterations (500) is used.

The remainder of the algorithm is divided in three parts:

[10] In a simulation with three vehicles and 2 lanes.

1. adding nodes (steps 1, 2 and 3 of expansion), this is done in line 9;
2. updating ancestors (steps 4 and 5), this is done in lines 11 through 31;
3. selecting the next tip for expansion, which is done in lines 33 through 38.

Each of these parts will be explained below.

6.5.1 Adding nodes

```
1  input: A_s, T
2  output: [Q_a], [S'_a], T
3
4  for a ∈ A_s
5     Q := R(s, a)
6     if τ − 1 ⩽ 0
7        Q_a := Q
8        S'_a := ∅
9     else
10       S'_a := {s' ∈ S | T(s, a, s') > 0}
11       for s' ∈ S'_a
12          p := T(s, a, s')
13          V := h(s', τ − 1)
14          add (s', V, p) to T
15          Q := Q + γpV
16       Q_a := Q
```

Algorithm Section 2: Pseudo code for `add_nodes`

Adding successor states is done using the function shown in Algorithm Section 2. The function uses a loop over all allowed actions. For each allowed action a, the immediate reward $R(s, a)$ is computed. The action value is set as $Q = R(s, a)$ as well. In Figure 6.5, there are three actions from the initial node.

Terminal nodes are never expanded. If $\tau - 1 = 0$, the successor states are terminal nodes, and $V(s', \tau - 1) = 0$ for all s'. This means the value of this action will be

$$Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} (T(s, a, s') \cdot 0) = R(s, a).$$

The immediate rewards are already known so no new tips are created. This can also be seen in lines 6 through 8. In the Figure, terminal nodes are shown in blue, these are not included in the list of tip nodes.

If the successor nodes are non-terminal, the set of reachable states S'_a for action a is computed. A loop over these states is started in line 11. For each successor state s' with probability p, the value $V = h(s', \tau - 1)$ is computed. It is assumed the heuristic for a particular state always returns the same value, such that no averaging is needed. A tip (s', V, p) is created is for the successor state. A reference to this tip node will be stored in its parent node, as shown in Table 6.3. The action value for the current action Q is updated iteratively. After the loop over all reachable states, the action value is stored and the next action is considered.

6.5.2 Updating ancestors

After this has been done for all actions, the value of parent (s, τ) is updated and the best action is marked. This is done in lines 11 through 31 of Algorithm Section 1.

First, the old value $V^-$of the tip is stored for later. Next, the value of the expanded tip is updated, and the best action found as shown in line 12 and 13. The expanded tip is added to the tree in line 16, using values that were computed previously.

Next, all ancestors are updated one by one, moving towards the root node, as shown in lines 18 through 31. The parent P of the node is selected and the action value of the parent is updated. Note that all values of other states remain constant during one iteration. Therefore, only V^- has to be subtracted in the sum of (18) and V^+ has to be added, as shown in line 21. Note that Q_{a_p} is the stored value of $Q(s_p, a_p)$ and p is the stored value of $T(s_p, a_p, s)$. The best action is updated by finding the action with the highest Q_a. After updating Q_{a_p}, V^- is stored for the next iteration and V^+ is computed. If V did not change, further ancestors will not be updated and the loop is ended. If V did change, this value is updated in the tip, p, s_p and a_p are updated and the next parent is updated, until the root node is reached.

6.5.3 Selecting nodes

```
1  input: G, T, do_optimal
2  output: tip, A_s, τ
3
4  loop 50
5    s_p := s_0
6    label deeper
7    P := G(s_p)
8    if do_optimal
9      a_p := P.a*
10   else
11     a_p := random action from P.A_s
12   s_p := prob. weighted state from P.S'_{a_p}
13   tip := T(s_p)
14   if tip is expanded
15     goto deeper
16   if tip is non-terminal
17     A_s := actions(tip.s)
18     τ := P.τ − 1
19     break
```

Algorithm Section 3: Pseudo code for `select_tip`

Selecting a node to expand is done using the subroutine `select_tip` in Algorithm Section 3. The search starts at the root node. Repeatedly, an action and successor state are 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 [42].

Prior to selecting nodes, in line 33 of Algorithm Section 1, a variable *do_optimal* is set to true with probability p_{opt}, this determines whether optimal actions are selected. If *do_optimal* is true, the best action is chosen, otherwise a random action is chosen, this is shown in lines 10 through 13. This means the best action could still be selected if *do_optimal* is false. Therefore, the actual probability of selecting a node inside the

best partial graph is slightly higher[11] than p_{opt}.

After selecting an action, the next node is found by randomly selecting a successor node. This random selection is weighted by the probability of a successor state occurring.

If the successor node is already expanded, the search continues from that node. If the node is non-terminal, τ and $A(s)$ are computed and the node is selected for expansion. Otherwise, the node is terminal and the search is restarted from the root node. If the search was restarted 50 times, it is assumed no non-terminal tip nodes exist inside or outside of G^* and the search is restarted in the other part of G. If still no tip is found, it is assumed the tree is fully explored and the algorithm finished. This can be seen in lines 35-38 of Algorithm Section 1.

[11] $p = p_{opt} + (1 - p_{opt})p_r$, where p_r is the probability of randomly selecting the best actions, which is dependent on allowed actions and explored nodes in the best partial graph.

7 Results

The behavior of the controller is validated by comparing its performance to the performance of a baseline controller. The baseline controller is a rule-based controller that was developed at TNO. Since the controller should be able to handle all possible highway scenarios, different scenarios are considered. First, the scenarios that were considered are introduced. Next, the controllers are compared based upon the criteria given in Section 3.3.

7.1 Scenarios

To be able to compare the performance of the decision maker to the baseline controller, multiple scenarios are considered. This was done to show the robustness of the decision maker. In addition, the decision maker should work in all highway scenarios, so a variety of scenarios is used. The current implementation of the controller is not suitable for all highway scenarios. For example, the controller does not take upcoming exits into account when choosing an action. This can be integrated in future works, e.g., through a distance to the exit in the state, and a reward feature that discourages lane changes to the left close to an exit.

The scenarios that are used were chosen because they show a variety of different behaviors. Certain behavior is expected for each scenario. It can easily be checked whether this behavior also occurs.

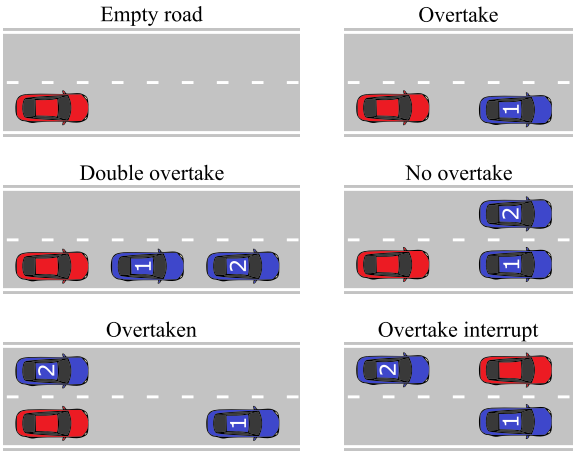

Figure 7.1: Schematic view of vehicle positions. Ego vehicle in red, obstacles in blue.

Table 7.1: Scenario parameters

Name	$v_{0,e}$	$v_{0,1}$	$v_{0,2}$
Empty road	70		
Overtake	70	50	
Fast overtake	70	65	
Double overtake	70	50	50
Single overtake	70	50	50
No overtake	70	50	50
Overtaken	70	50	80
Overtake Interrupt	60	50	90

A schematic view of relative vehicle positions for all scenarios is given in Figure 7.1. The initial/desired speeds are given in Table 7.1. The fast overtake scenario uses the overtake positions from the figure, the single overtake uses the double overtake positions. This should not affect the behavior of the controller, though the reward feature weights might require additional tuning for higher speeds. Unless mentioned otherwise, all obstacle vehicles are lane keeping with adaptive cruise control.

- **Empty road**
  There are no obstacle vehicles in this scenario. It is used to check default behavior, such as maintaining speed and not doing unnecessary maneuvers.
- **Normal overtake**
  There is one obstacle vehicle in this scenario. It is driving at a constant speed, which is significantly lower than the desired speed of the ego vehicle. The ego vehicle starts behind the obstacle and it is expected it will overtake the slower vehicle.
- **Normal overtake fast**
  This scenario is the same as the previous scenario, except the obstacle speed is much closer to the ego speed. Simulating an overtake at different speeds shows the balance between inactivity and speed. It is expected that no overtake will take place.
- **Double overtake**
  There are two slower vehicles in front of the ego vehicle. The vehicles are relatively close together, to promote a double overtake.
- **Single overtake**
  The same as above, except with increased inter-vehicle distance. Simulating this scenario twice shows the balance between inactivity and right lane alignment. The vehicle is expected to overtake each vehicle individually.
- **No overtake**
  The ego vehicle is stuck behind two vehicles driving at a low speed on different lanes. The ego vehicle is unable to overtake, but should still continue to behave correctly. It is debatable whether the vehicle should move to the left lane, as to indicate the desire to overtake a slower vehicle. Both lanes are considered correct behavior, as long as the controller does not continue switching lanes.
- **Overtaken**
  This is the same as the overtake scenario but with an additional vehicle that is executing a double overtake. Due to simulation considerations, the obstacle vehicle will continue lane keeping on the left after the double overtake. The ego vehicle should wait for the first vehicle to pass, before overtaking the slower vehicle itself.
- **Overtake interrupt**
  Here the ego vehicle is halfway through an overtake (at reduced speed), when a faster vehicle approaches from behind. The ego vehicle might consider increasing its speed to promote safety and urgency for the approaching vehicle. To prevent the obstacle vehicle from crashing into the ego vehicle.

7.2 Controller comparison

Both the baseline controller and new controller were tested on all scenarios, to be able to compare them. The simulation time for each simulation is $40s$, and thus with $t_s = 0.01s, N = 4001$.

Table 7.2: Comparison of optimality criteria for baseline and new controller.

	Baseline			New		
Scenario	*safety*	d_{trav}	n_{LC}	*safety*	d_{trav}	n_{LC}
Empty road	15.00	778	0	15.00	778	0
Normal overtake	12.90	778	2	13.13	780	2
Normal overtake fast	13.30	764	2	13.30	742	0
Not passing	10.03	655	1	11.69	647	0
Overtaken	11.99	774	2	11.70	712	2
Overtake interrupt	9.35	667	1	12.14	701	1
Double overtake	9.92	725	4	11.39	729	2
Single overtake	12.05	778	4	10.52	698	3

The results for both controllers are given in Table 7.2. As can be seen for most scenarios, the new controller has an equal or higher safety score. This also results in slightly less distance covered, as can also be seen in the table. Below, several scenarios that show interesting behavior are highlighted.

From the table, it can be observed that both controllers successfully perform a *normal overtake*, the new controller is slightly more safe. However, as seen for the *fast overtake*, the baseline controller overtakes a slightly slower vehicle as well, whereas the new controller chooses to follow this vehicle at a slightly lower speed than was desired. As a result, the new controller covers $22m$ less distance, but also executes two fewer lane changes.

In the Figures 7.2, 7.3 and 7.4, the speed profiles for both controllers are given. The baseline controller is shown on the lefthand side, the new controller is shown on the righthand side. Maneuvers are shown using the background shading, dark gray denotes an LCL, light gray denotes an LCR. Whenever only two speed profiles are visible, the observers have equal speed profiles.

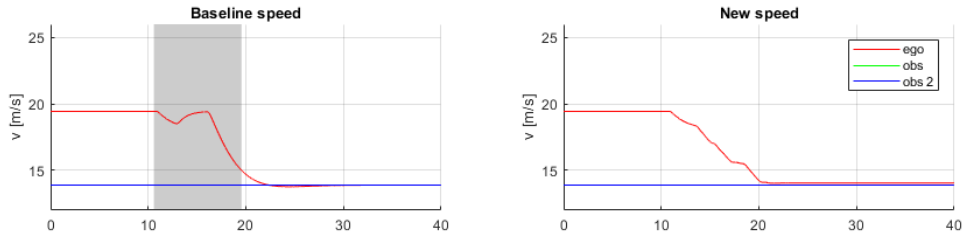

Figure 7.2: Results for scenario *not passing*. Left: Baseline controller. Right: New controller. Dark gray background denotes LCL. The speed of both obstacle vehicles is equal.

Figure 7.2 shows scenario *not passing*. It can be seen that both controllers slow down, but the baseline controller first does a LCL, and then stays on the left lane. The new controller stays on the right lane, because there is a reward for right alignment of the

vehicle.

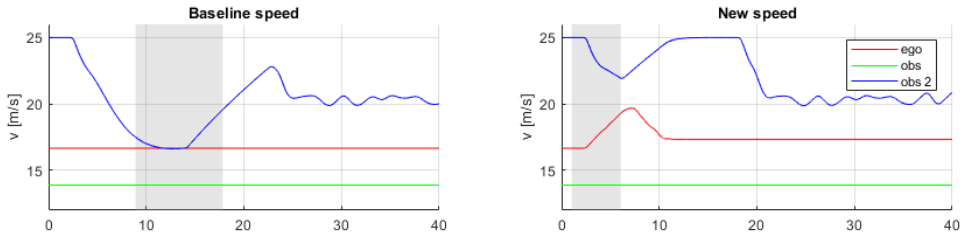

Figure 7.3: Results for scenario *overtake interrupt*. Left: Baseline controller. Right: New controller. Light gray background denotes LCR.

For the *Overtake interrupt*, the new controller accelerates to above the preferred speed, which is equal to the initial speed, as can be seen in Figure 7.3. This increases the safety with respect to the approaching vehicle, which is much faster. After finishing its overtake, the ego vehicle 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 baseline controller maintains it desired speed, causing large deceleration for the obstacle vehicle, which then accelerates again after passing. Note that the obstacle vehicle uses a lower reference speed for the second half of the scenario, because there is a strong curvature in the road. This behavior has no influence on the execution of the scenario for either controller.

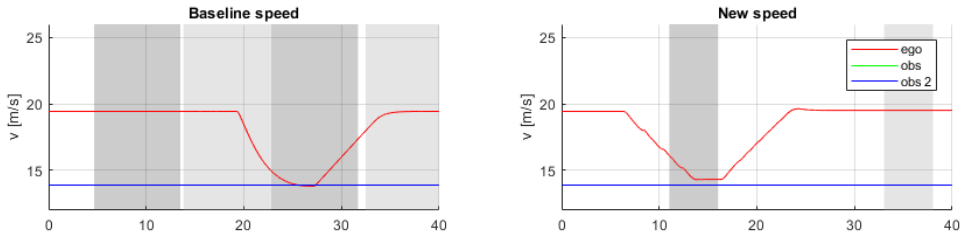

Figure 7.4: Results for scenario *double overtake*. Left: Baseline controller. Right: New controller. Dark gray background denotes LCL, light gray denotes LCR. The speed of both obstacle vehicles is equal.

It was observed during simulations that the new controller will usually slow down before an overtake. This can be seen in Figure 7.4, for the *double overtake* scenario. The ego vehicle slows down just before the double overtake and accelerates again after switching lanes. Compare this to the behavior of the baseline controller, which maintains the original speed, overtakes only the first vehicle and then rapidly decelerates when encountering the second vehicle. This also results in a slightly higher distance covered for the new controller in this scenario.

An explanation of this behavior can be that slowing down increases the TTC, and therefore the safety of a maneuver. However, the time spent close to the vehicle also increases, which decreases the safety measure for the entire scenario. This is why the new controller has a lower distance and safety score than the baseline controller for the *single overtake*.

7.3 Discussion

Given current results, it can be concluded that the overall architecture and algorithm design is a viable methodology to implement scalable and generic decision making systems for autonomous driving. For the scenarios where the new controller successfully executes maneuvers, its performance is similar to or better than the baseline controller. It can be observed that the new controller considers the safety of the ego vehicle in interaction with other vehicles. Compared to the baseline controller, the new controller shows behavior that considers more actively the future effects of an action. This is because the MDP also takes rewards of future actions into account.

8 Concluding remarks

In this section, first a brief conclusion to this report is given. Second, recommendations for improving the decision maker proposed here and future research directions are given.

8.1 Conclusions

The goal of this project was to design a scalable and generic decision maker that incorporates safety considerations. A decision maker is proposed, which uses a two-stage architecture to accomplish this. First, the safety of all possible actions is evaluated, next the optimal action is selected from the actions that are deemed safe.

The safety of actions is guaranteed by a supervisor and an action set reduction component. The supervisor is automatically generated from plant models and requirement models. By automatically generating a supervisor, it is guaranteed to adhere to the requirements, to disable as little behavior as possible and to always allow return to an idle state. Due to limitations in the tooling, the supervisor can only check the safety of 1 time step. Therefore, action set reduction is used to check the safety of future time steps. Action set reduction checks long term safety using TTC and then selects only actions which exceed a certain safety threshold.

A Markov decision process (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 is obtained by quantizing the multi-vehicle system state. Transitions are based on the chosen action and the probability of reaching different positions in the next time step. Rewards are based on eight different reward features that reflect relevant behavior. These features are weighted according to manually tuned weights.

The MDP is solved using a search algorithm named Anytime AO*. This algorithm searches for the best actions over a fixed horizon, whilst maintaining the best action found so far. That way, the algorithm always produces a (near-optimal) result, even when the algorithm has not completely finished yet.

To validate the behavior of the decision maker, its performance is compared to a baseline controller according to three different criteria. To show the broad applicability, both controllers are compared on 8 different scenarios. From this comparison, it is concluded that the controller design produces decisions that are safe and considers the impact of this action on the future multi-vehicle system, for the given scenarios.

8.2 Recommendations

As mentioned above, the newly proposed controller shows promising behavior. However, many improvements can be made to the controller, which are given below.

In the state, obstacle position estimates can be improved by considering the real distance along the road (x_i^R) instead of forward distance (x_i^E). The state can also

be extended using road curvature to give a better representation of the real world. Currently, it is assumed the number of lanes does not change. If information on lane reductions or oncoming ramps is included in the state, the decision maker can take these into account when deciding upon an action.

The long-term safety check for action set reduction uses TTC, which is also used in the short-term safety check for the supervisor. To better accommodate the probabilistic nature of future traffic, long-term safety should be measured using a probabilistic measure. For example, the measures proposed in [46] or [47] can be considered.

To keep computation times low, several assumptions are made in the transition function for the MDP. Below, it is explained how some of these assumptions could be eliminated.

- **Heading of obstacle vehicle is always 0**
  This is clearly too restrictive. Since predicting maneuvers of vehicles is a complex problem [48], it is necessary to assume that obstacle vehicles do not start a maneuver. However, if a vehicle is halfway through a lane change, it is safe to assume it will finish the maneuver. This can easily be used to generate more accurate predictions in specific scenarios.[12]

- **Obstacle vehicle speed is constant**
  Similar to constant heading, this is also too restrictive. If communication with other vehicles is possible, obstacles can communicate their intent. This produces more accurate and realistic predictions, resulting in a safer system.

- **x and y positions are independent stochastic variables**
  The positions are not independent, since they are related through the heading and speed. Moreover, there is a lane which a vehicle will probably follow. It is not known whether this assumption yields realistic values, because the difference between computed probabilities and actual probabilities is unknown. This should be further investigated.

- **Distributions of quantized state values are uniform**
  The continuous state of a vehicle will have some equilibrium within a quantized interval, e.g., the speed will be inclined towards the speed limit. However, this inclination towards one value is not modeled, since the intentions of vehicles are not taken into consideration. If the quantized is small enough, the distribution can be accurately represented as uniform. It should be investigated whether the chosen quantization is small enough.

The results show decisions that fall in line with expectations from a human driver. However, for actual highway speeds (over $100 km/h$) or for different vehicles, the rewards might have to be tuned again. A more structured method for finding weights should be used, for example, reinforcement learning.

[12] In current simulations, other vehicles do not start maneuvers, but the possibility exists and should be accounted for.

The observed behavior was adequate, but might be sub-optimal, since several assumptions were made to ease the implementation of the algorithm finding the optimal action. For each assumption, the validity and possible improvements are discussed below.

- **Heuristic function**
  The currently used reward function is deterministic, all randomness in the long-term reward comes from the transition model. In the heuristic, this randomness is removed by assuming the reward is constant over the remaining horizon. This is an oversimplified view of the MDP, but the heuristic is only used to give an estimate, so using a constant heuristic is considered a valid assumption. In [42], it is suggested that the heuristic value should be the long-term reward when following some base policy.

- **Computational time**
  A chosen number of fixed iterations is assumed to represent a reasonable computation time. During simulation the exact computation time cannot be measured, so this assumption cannot be validated. It is estimated from simulations that computing one time step using 500 iterations for the controller takes less than the allowed time of $1s$. When implementing this algorithm on actual hardware, the algorithm should be bound by time, not number of iterations.

- **Terminating the algorithm**
  It is assumed the algorithm has finished if no non-terminal tip is found by randomly selecting a tip 100 times. The entire tree consists of millions of nodes, so it is assumed an expandable node, i.e., a non-terminal tip, can always be found for the chosen number of iterations. Therefore, this assumption is considered valid. During simulations, it has not yet occurred that the algorithm finished within the given number of iterations without finding a non-terminal tip. This assumption can be removed if an efficient way to check whether all nodes were expanded is found.

- **Number of iterations**
  This algorithm produces a near optimal solution, given enough iterations are done. However, if the decisions are according to expectations, it can be assumed the number of iterations is sufficient. It should be investigated how many iterations are needed to produce the optimal action.

- **Tip selection method**
  In [42], it is also mentioned that tip selection can significantly improve performance. The tip selection criterion that is used is based on probability of occurrence of a state. This criterion can be changed to improve performance. For example, in [42] tips are selected based on potential impact on the value of an action.

If another decision maker is designed using the same architecture, the possibility of using a partially observable MDP (POMDP) should be investigated. This means the

method for finding the optimal action will be different. Therefore, a different solver than Anytime AO* might also be necessary, such as DESPOT, which was introduced in [49].

For future research, it can be investigated how communication influences the performance of the controller. In addition, scenarios with more vehicles can be considered. It might be interesting to study the effect on traffic flow of having multiple AVs using this decision maker in highway scenarios.

References

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

[2] J. Ploeg, E. Semsar-Kazerooni, A. I. M. Medina, J. F. de Jongh, J. van de Sluis, A. Voronov, C. Englund, R. J. Bril, H. Salunkhe, Á. Arrúe, *et al.*, "Cooperative automated maneuvering at the 2016 grand cooperative driving challenge," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 4, pp. 1213–1226, 2018.

[3] F. Borrelli, P. Falcone, T. Keviczky, J. Asgari, and D. Hrovat, "Mpc-based approach to active steering for autonomous vehicle systems," *International Journal of Vehicle Autonomous Systems*, vol. 3, no. 2-4, pp. 265–291, 2005.

[4] G. Schildbach, M. Soppert, and F. Borrelli, "A collision avoidance system at intersections using robust model predictive control," in *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pp. 233–238, IEEE, 2016.

[5] X. Xu, J. W. Grizzle, P. Tabuada, and A. D. Ames, "Correctness guarantees for the composition of lane keeping and adaptive cruise control," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 3, pp. 1216–1229, 2018.

[6] T. Korssen, V. Dolk, J. van de Mortel-Fronczak, M. Reniers, and M. Heemels, "Systematic model-based design and implementation of supervisors for advanced driver assistance systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 533–544, 2017.

[7] R. Malik, K. Åkesson, H. Flordal, and M. Fabian, "Supremica–an efficient tool for large-scale discrete event systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5794–5799, 2017.

[8] M. Fabian and A. Hellgren, "Plc-based implementation of supervisory control for discrete event systems," in *Proceedings of the 37th IEEE Conference on Decision and Control*, vol. 3, pp. 3305–3310, IEEE, 1998.

[9] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *Proceedings of the 16th International conference on Hybrid systems: computation and control*, pp. 1–10, ACM, 2013.

[10] C. Kim and R. Langari, "Game theory based autonomous vehicles operation," *International Journal of Vehicle Design*, vol. 65, no. 4, pp. 360–383, 2014.

[11] M. Fusco, E. Semsar-Kazerooni, J. C. Zegers, and J. Ploeg, "Decision making for connected and automated vehicles: A max-plus approach," in *Proceedings on the 88th IEEE Vehicular Technology Conference*, pp. 1–5, IEEE, 2018.

[12] M. Fusco and M. Alirezaei, "Report - decision making for autonomous vehicles: a max-plus approach," unpublished.

[13] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity: an algebra for discrete event systems.* John Wiley & Sons Ltd, 1992.

[14] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pp. 101–110, ACM, 2010.

[15] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the 21st international conference on Machine learning*, p. 1, ACM, 2004.

[16] Y. Guan, S. E. 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," *Journal of Intelligent and Connected Vehicles*, vol. 1, no. 2, pp. 77–84, 2018.

[17] S. Zhou, Y. Wang, M. Zheng, and M. Tomizuka, "A hierarchical planning and control framework for structured highway driving," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9101–9107, 2017.

[18] W. Liu, S.-W. Kim, S. Pendleton, and M. H. Ang, "Situation-aware decision making for autonomous driving on urban road using online POMDP," in *Proceedings of the 2018 IEEE International Conference on Intelligent Vehicles Symposium*, pp. 1126–1133, IEEE, 2015.

[19] N. Li, D. W. Oyler, M. Zhang, Y. Yildiz, I. Kolmanovsky, and A. R. Girard, "Game theoretic modeling of driver and vehicle interactions for verification and validation of autonomous vehicle control systems," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 5, pp. 1782–1797, 2017.

[20] J. Wei, J. M. Dolan, and B. Litkouhi, "Autonomous vehicle social behavior for highway entrance ramp management," in *Proceedings of the 2013 IEEE Intelligent Vehicles Symposium*, pp. 201–207, IEEE, 2013.

[21] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th international conference on hybrid systems: Computation and control*, pp. 239–248, ACM, 2015.

[22] E. F. Camacho and C. Bordons, *Model predictive control. Advanced textbooks in control and signal processing.* Springer-Verlag, London, 2007.

[23] G. Naus, J. Ploeg, M. Van de Molengraft, W. Heemels, and M. Steinbuch, "Design and implementation of parameterized adaptive cruise control: An explicit model predictive control approach," *Control Engineering Practice*, vol. 18, no. 8, pp. 882–892, 2010.

[24] F. Borrelli, *Forecasts, Uncertainty and Control in Autonomous Systems.* At the occasion of Manfred Morari's 65th birthday and retirement from ETH, May 2016.

[25] F. Fabiani and S. Grammatico, "A mixed-logical-dynamical model for automated driving on highways," in *Proceedings of the 2018 IEEE Conference on Decision and Control*, pp. 1011–1015, IEEE, 2018.

[26] L. Fridman, B. Jenik, and J. Terwilliger, "Deeptraffic: Driving fast through dense traffic with deep reinforcement learning," *arXiv preprint arXiv:1801.02805*, 2018.

[27] J. Zheng, K. Suzuki, and M. Fujita, "Predicting drivers lane-changing decisions using a neural network model," *Simulation Modelling Practice and Theory*, vol. 42, pp. 73–83, 2014.

[28] L. Alvarez and R. Horowitz, "Safe platooning in automated highway systems part I: Safety regions design," *Vehicle System Dynamics*, vol. 32, no. 1, pp. 23–55, 1999.

[29] J. A. Ligthart, J. Ploeg, E. Semsar-Kazerooni, M. Fusco, and H. Nijmeijer, "Safety analysis of a vehicle equipped with cooperative adaptive cruise control," *IFAC-PapersOnLine*, vol. 51, no. 9, pp. 367–372, 2018.

[30] E. Semsar-Kazerooni and J. Ploeg, "Interaction protocols for cooperative merging and lane reduction scenarios," in *Proceedings of the 18th IEEE International Conference on Intelligent Transportation Systems*, pp. 1964–1970, IEEE, 2015.

[31] G. v. Bochmann, M. Hilscher, S. Linker, and E.-R. Olderog, "Synthesizing controllers for multi-lane traffic maneuvers," in *Proceedings of the International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pp. 71–86, Springer, 2015.

[32] I. Llatser, A. Festag, and G. Fettweis, "Vehicular communication performance in convoys of automated vehicles," in *Proceedings of the 2016 IEEE International Conference on Communications*, pp. 1–6, IEEE, 2016.

[33] J. Burrell, "How the machine thinks: Understanding opacity in machine learning algorithms," *Big Data & Society*, vol. 3, no. 1, pp. 1–12, 2016.

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

[35] A. Laureshyn, Å. Svensson, and C. Hydén, "Evaluation of traffic safety, based on micro-level behavioural data: Theoretical framework and first implementation," *Accident Analysis & Prevention*, vol. 42, no. 6, pp. 1637–1646, 2010.

[36] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[37] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM journal on control and optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[38] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, 2011.

[39] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*, vol. 135. MIT press Cambridge, 1998.

[40] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

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

[42] B. Bonet and H. Geffner, "Action selection for MDPs: Anytime AO* versus UCT," in *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, AAAI, 2012.

[43] D. A. van Beek, W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. Van De Mortel-Fronczak, and M. A. Reniers, "Cif 3: Model-based engineering of supervisory controllers," in *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 575–580, Springer, 2014.

[44] J. Lunze, "Qualitative modelling of linear dynamical systems with quantized state measurements," *Automatica*, vol. 30, no. 3, pp. 417–431, 1994.

[45] M. Mitschke and H. Wallentowitz, "Lineares einspurmodell, objektive kenngrößen, subjektivurteile," in *Dynamik der kraftfahrzeuge*, ch. 4, Springer, 1972.

[46] E. van Nunen, T. van den Broek, M. Kwakkernaat, and D. Kotiadis, "Implementation of probabilistic risk estimation for VRU safety," in *Proceedings of the 8th International Workshop on Intelligent Transportation*, 2011.

[47] M. Althoff, O. Stursberg, and M. Buss, "Model-based probabilistic collision detection in autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 2, pp. 299–310, 2009.

[48] F. Remmen, I. Cara, E. de Gelder, and D. Willemsen, "Cut-in scenario prediction for automated vehicles," in *Proceedings of the 2018 IEEE International Conference on Vehicular Electronics and Safety*, pp. 1–7, IEEE, 2018.

[49] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," in *Advances in neural information processing systems*, pp. 1772–1780, 2013.

A Time to collision measure

In [35], a generic method for computing the time to collision (TTC) between to vehicles is given. The computations are based on the observation that a two-vehicle collision always consists of a corner of one vehicle hitting the side of another vehicle. This gives 32 possible collisions (4 corners $\times$ 4 sides $\times$ 2 vehicles), assuming the vehicles are rectangular. The TTC is then found by taking the lowest TTC for a moving line section (side) and a moving point (corner).

A.1 Collision of line and point

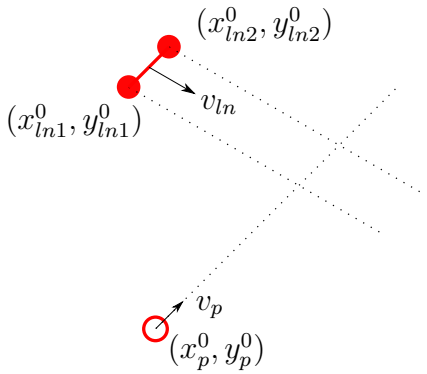

Figure A.1: A moving line and point, adapted from [35].

Given a moving point (x_p, y_p) with velocity vector $[v_{px} v_{py}]^\top$, the position at time t can be found by:

$$\begin{cases} x_p = x_p^0 + v_{px} t \\ y_p = y_p^0 + v_{py} t \end{cases} \tag{21}$$

where (x_p^0, y_p^0) is the original position of the point. Note that the velocity vector is given in Cartesian coordinates here, whereas the state from (3) contains polar coordinates. This conversion is considered trivial.

Similarly, for the endpoints of a moving line section between (x_{ln1}, y_{ln1}) and (x_{ln2}, y_{ln2}), with velocity $[v_{lnx} v_{lny}]^\top$ and initial positions (x^0_{ln1}, y^0_{ln1}) and (x^0_{ln2}, y^0_{ln2}), the positions at time t are given by:

$$\begin{cases} x_{ln1} = x^0_{ln1} + v_{lnx} t \\ y_{ln1} = y^0_{ln1} + v_{lny} t \end{cases} \quad \text{and} \quad \begin{cases} x_{ln2} = x^0_{ln2} + v_{lnx} t \\ y_{ln2} = y^0_{ln2} + v_{lny} t \end{cases} \tag{22}$$

Both the point and line section are shown in Figure A.1.

The slope k of the line can be found from the initial positions of the line section:

$$k = \frac{y^0_{ln2} - y^0_{ln1}}{x^0_{ln2} - x^0_{ln1}} \tag{23}$$

The point p intersects the line when:

$$\frac{y_p - y_{ln1}}{x_p - x_{ln1}} = k \tag{24}$$

Because the slope for the line section between $ln1$ and p is equal to the slope for the line, and $ln1$ is known to be on the line.

By substituting (21) and (22) in (24), the possible time of collision t_{coll} can be found.

$$t_{coll} = \frac{(y_p^0 - y_{ln1}^0) - k(x_p^0 - x_{ln1}^0)}{-(v_{py} - v_{lny}) + k(v_{px} - v_{lnx})}$$

Note that this equation is undefined if the line is vertical, for then $k = \infty$. In that special case, a different equation is used:

$$t_{coll} = \frac{-(x_p^0 - x_{ln1}^0)}{(v_{px} - v_{lnx})}$$

Only relevant values for t_coll, i.e., $0 \leqslant t_coll \leqslant t_{max}$ with $t_{max} = 6$ are considered to find the TTC.

The final positions of the line section and point are computed by substituting $t = t_{coll}$ in (21) and (22). A collision only occurs when the point crosses the line within the line section:

$$\begin{cases} x_{ln1} \leqslant x_p \leqslant x_{ln2}, & \text{if } x_{ln2} \geqslant x_{ln1} \\ x_{ln2} \leqslant x_p \leqslant x_{ln1}, & \text{otherwise} \end{cases} \quad \text{and} \quad \begin{cases} y_{ln1} \leqslant y_p \leqslant y_{ln2}, & \text{if } y_{ln2} \geqslant y_{ln1} \\ y_{ln2} \leqslant y_p \leqslant y_{ln1}, & \text{otherwise} \end{cases} \tag{25}$$

B Reward table

Below is a table showing the values of each reward feature in Figure 6.4.

Table B.1: Reward feature values for different actions. Each value is multiplied by λ_i. R_{lc} is not shown, for λ_{lc} is 0.

a	Q	$\lambda_v R_v$	$\lambda_{aL} R_{aL}$	$\lambda_{aV} R_{aV}$	$\lambda_s R_s$	$\lambda_l R_l$	$\lambda_u R_u$	$\lambda_c R_c$
LCL_c	104.16	55.176	0.019	0.295	23.574	0.796	21.996	2.300
Lk_c	103.21	55.052	0.351	0.342	23.350	0.958	21.855	1.303
LCL_d	102.47	54.304	0.000	0.143	23.617	0.859	21.182	2.363
Lk_d	101.78	54.272	0.354	0.150	23.616	0.955	21.155	1.273
LCL_a	103.01	55.775	0.003	0.135	21.291	0.806	22.645	2.352
Lk_a	102.80	55.838	0.330	0.130	21.579	0.838	22.722	1.360