CHALMERS
UNIVERSITY OF TECHNOLOGY

Decision Making for Automated Vehicles in Merging Situations

- using Partially Observable Markov Decision Processes

Master's thesis in Systems Control and Mechatronics

MALIN NILSSON

Department of Signals & Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2014

The research leading to these results has received funding from the European Commission Seventh Framework Programme under the project AdaptIVe, grant agreement number 610428. The author(s) would like to thank all partners within AdaptIVe for their cooperation and valuable contribution.

CHALMERS UNIVERSITY OF TECHNOLOGY

Department of Signal and Systems

Visiting address:
Hörsalsvägen 9-11, level 5-7
Göteborg, Sweden

Postal address:
Chalmers University of Technology
Instutitionen för Singaler och System
412 96 Göteborg

Telephone:
031-772 10 00

Telefax:
031-772 17 48

Web page:
http://www.chalmers.se/sv/institutioner/s2

Abstract

In this thesis a decision making algorithm for automated cars in lane change situations is presented. The algorithm accounts for situations when all objects detected by the sensor system cannot be classified. The algorithm is partially formulated as a Partially Observable Markov Decision Process which is solved approximately with means of Point Based Value Iteration.

The algorithm is implemented in a simulation environment and then tested and analyzed with simulation data. The results show that the decision algorithm is able to choose a proper gap to merge into and it is able to cancel a maneuver if the traffic situation changes so that a merging operation cannot be completed. If the execution time of the implemented Point Based Value Iteration algorithm could be decreased this would further improve the real world applicability of the decision making algorithm.

Keywords: Automated cars, merging, decision making, POMDP, MDP, Point Based Value Iteration.

Decision Making for Automated Vehicles in Merging Situations -
using Partially Observable Markov Decision Processes

Malin Nilsson

Author:	Malin Nilsson, Systems Control and Mechatronics Master's Program, Chalmers University of Technology
Advisor:	Stefan Solyom, Technical Expert - Autonomous Drive, Volvo Car Corporation
Examiner:	Jonas Sjöberg, Professor, Research Group Leader, Mechatronics, Chalmers University of Technology
Sponsor:	Volvo Car Corporation

Acknowledgements

For their help in the work that has resulted in this thesis, I would like to thank a few people.

First of all I would like to thank the people at the department of Automated Drive at Volvo Car Corporation for a great few months which have been very rewarding. In particular I would like to thank my advisor Stefan Solyom and my examiner Jonas Sjöberg for all the support I have been given. I also want to extend thanks to Julia Nilsson, PhD student at Volvo Car Corporation, for giving me valuable feedback especially in the beginning of the project.

Lastly I would like to thank Staffan Häglund for providing support and cheering me up whenever the algorithm troubled me.

Nomenclature

ADR: Average Discounted Reward, see (4.1).

Agent: The agent in an MDP or a POMDP is the decision maker. In the merging decision algorithm the agent is the host car.

Cardinality of X, $|X|$: The cardinality of X is the number of elements in the set X.

Gap: Defined in Definition 3.1.3.

Ghost-car: A ghost-car is an object that is falsely classified as a car. In the merging decision algorithm a ghost-car is an object that has been detected but that can be neglected in the context of merging.

Host car: The host car is the automated car for which the algorithm presented in this thesis is designed for.

Master function: Defined in Definition 3.1.4.

MDP: Markov Decision Process

Merging decision algorithm: The merging decision algorithm is the algorithm presented in this thesis and consists of four stages.

Merging lane: The merging lane is the lane that the host car will drive in when the merging maneuver has been completed.

PBVI: Point Based Value Iteration

POMDP: Partially Observable Markov Decision Process

Status of a suspected ghost-car: Defined in Definition 3.1.2.

Suspected ghost-car: Defined in Definition 3.1.1.

Time complexity: The time complexity of an algorithm, commonly described using the big O notation, describes the time it takes for an algorithm to run as a function of the length of the input. The big O notation is described asymptotically, hence lower order terms and coefficients are excluded. For example, a for-loop from 1 to N over an operation that is constant in time has complexity $O(N)$.

Contents

1

Introduction

The topic of this thesis concerns decision making for automated cars in merging situations where the automated car needs to change lane and join its path with vehicles driving in the adjacent lane. An algorithm, denoted *the merging decision algorithm*, is presented and its performance is evaluated in a simulation environment.

1.1 Thesis Outline

This chapter describes the problems involved in decision making in general and automated decision making in merging situations in particular. It also presents an overview of the final solution and states the contributions of the thesis. In Chapter 2 the theory that the algorithm is based on is presented where the main focus is on Markov Decision Processes and Partially Observable Markov Decision Processes. In Chapter 3 the merging decision algorithm is described in detail and in Chapter 4 the results are presented followed by a discussion and concluding remarks in Chapter 5.

1.2 Context of the Thesis Project

This thesis project is part of the Volvo Car Group pilot project named ”Drive Me” where the goal is to produce one hundred automated Volvo cars by the year of 2017. The cars are to be used on typical commuter roads in and around the city of Gothenburg [1]. The automated cars must be capable of handling a large number of different traffic situations including the merging situation where decision making is fundamental in order to safely perform the merging maneuver. The decisions that need to be made concern where to merge, when to merge and whether to continue to merge or cancel a merging maneuver.

1.3 The Merging Situation and the Optimal Merging Maneuver

Figure 1.1 shows a typical merging situation where the automated car - the host car - is in the left lane and there are five cars in the right lane such that every two subsequent cars create a gap. The merging situation may also involve a vehicle in front of the host car in the same lane, denoted $vehicle_{Front}$. The goal for the host car is to merge into the right lane - the merging lane - before the *end-point* (also illustrated in Figure 1.1) and without violating any traffic rules. Optimally an *optimal merging maneuver* should be performed.

Definition 1.3.1 - Optimal merging maneuver
For a given merging situation, an optimal merging maneuver is a maneuver where the host car manages to merge into the merging lane before the end-point without violating any traffic rules and with as small impact as possible on all vehicles involved in the merging situation, including the host car itself.

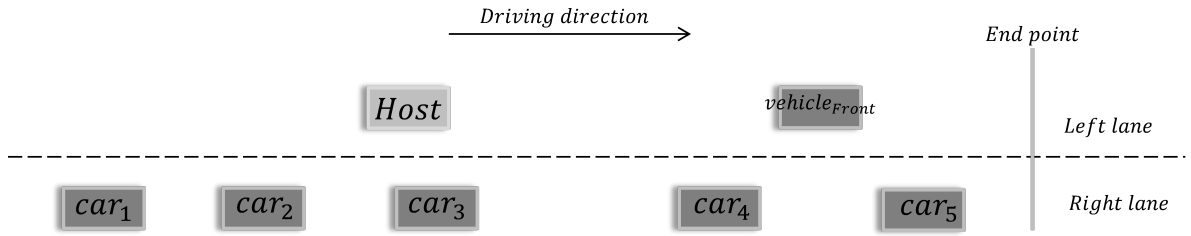


Figure 1.1: A typical merging situation with the host car in the left lane, a vehicle in front of the host car in the left lane and five cars in the right lane. The host car should merge into the right lane before the end-point. All vehicles are traveling to the right in the figure.

With this definition a merging maneuver involving low positive or low negative acceleration (longitudinally or laterally) is preferred to a merging maneuver with high acceleration values. The highest impact on vehicles involved in the merging maneuver corresponds to a car crash and sometimes this may be the only option available in order to merge into the right lane before the end-point. However, since an intentional car crash would violate traffic rules, an optimal merging maneuver would not be possible to perform in this case.

As stated, decision making is fundamental when performing a merging maneuver and next section discusses decision making in general terms.

1.4 Decision Making

Decision making can be broken down into two categories; decision making when the outcome is deterministic and decision making when the outcome is stochastic. An outcome appears to be stochastic when there is limited knowledge available or there is limited access to the information that is needed in order to determine the outcome. Out of all possible outcomes some are usually preferred to others, hence the outcomes have different *utilities*. Also, some outcomes are more

likely than others. The likelihood of an outcome may not be known due to limited knowledge on how to determine the probability of an outcome. Also it is not possible to determine the probability of an outcome if all data that is needed to determine the probability is not available. If this is the case, estimations of the probabilities can be made.

Good decision making weights the probabilities of all outcomes with the utilities of these outcomes. However, since there is not just a single way to weight the utilities it can be difficult to tell whether a decision is good or not or if one decision is better than another. There are some situations where all rational people would agree that one decision is better than another but this is not always the case and then there may not be an objective method available that can be used to measure the quality of a decision. Moreover, if the probability of an outcome is estimated differently from using different models or judgments this makes it even harder to determine the quality of a decision.

Also, by only considering the outcome of a decision it is not possible to determine the quality of the decision since a good decision does not necessarily have a good outcome and conversely, bad decisions do not imply bad outcomes. As an example, if the host car is close to the end-point and it is next to a gap that is both wide and is getting larger, one can argue that a good decision would be to merge into that gap. If the host car starts to merge into that gap but the driver in the car in front of the host car in the merging lane suddenly breaks hard so that the size of the gap quickly decreases, the outcome is bad. Still, the decision can be regarded as good because in considering all information that was available at the time when the decision was made, the gap chosen was a good gap to merge into. This means that an optimal merging maneuver does not always have to correspond to good decision making but the single best decision making method to use when performing a merging maneuver is to consistently make good decisions.

1.5 Automated Decision Making in Merging Situations and its Difficulties

Automated decision making in merging situations is in similarity to decisions made by human drivers associated with stochastic outcomes, since it is impossible to determine the future behavior of all vehicles involved in the situation. Even though a merging maneuver is an interaction maneuver, which means that the driver can influence the behavior of surrounding vehicles to some extent, this only means that the likelihood for specific outcomes can be increased or decreased, not determined.

Decisions that need to be made in merging situations concern:

1. Which gap to try to merge into.
2. When to try to merge into the gap.
3. If an initiated merging maneuver should be canceled.

As stated in Section 1.4 the decisions should be based on the probabilities as well as the utilities of the outcomes. The probabilities of the outcomes can only be estimated since in a merging situation, like in any complex traffic situation, it is not known how to exactly determine the probabilities of the outcomes. Also the sensor system is not perfect which means that the information available is limited and consequently only estimations of the probabilities of the outcomes can be made. The sensor system is limited in the way that it may not detect all relevant objects and/or it might detect relevant objects twice. The sensor system is also associated with uncertainties regarding the velocities of the objects and the distances between the objects and the automated car. Moreover the sensor system is limited when it comes to classification of objects. Some objects can be classified as cars with high probability whereas other objects can only be classified as cars with low probability. These objects may be trucks, motorbikes, traffic signs etc. or just noise. Since an automated merging maneuver is a real time application the decisions also need to be made in real time. This means that the execution time for a decision making algorithm is very limited and this can have an impact on the quality of the decisions.

A merging situation is associated with decision making where the exact quality of the decisions cannot be determined objectively since the probabilities of the outcomes are estimated and there is not just one way to weight the utilities of the outcomes. Therefore the quality of a decision in a merging situation can only be evaluated through rational reasoning.

1.6 Problem Definition and Scope of Thesis

The algorithm presented in this thesis - the *merging decision algorithm* - can be used to make the decisions that are associated with a merging maneuver and that are listed in Section 1.5. Hence, the algorithm decides:

1. Which gap the automated car should try to merge into.
2. When the automated car should try to merge into the gap.
3. If the automated car should cancel an initiated merging maneuver.

The algorithm tries to make as good decisions as possible from using the sensor data. However, since there is no formula available that objectively quantifies the quality of decisions made in merging situations, the quality of the decisions made by the algorithm will only be evaluated through rational reasoning.

Some assumptions that simplify the decision making are made. It is assumed that there will be a gap available that the host car can merge into before the end-point and that the sensor system is perfect except when it comes to the classification of detected objects. It is also assumed that all vehicles in the right lane are cars which means that all objects detected are either cars or objects that can be neglected in the context of merging.

1.7 Overview of the Solution

The merging decision algorithm consists of four stages, illustrated in Figure 1.2, where decisions 1 and 2 listed in Section 1.6 are handled in Stage 1 and decision 3 is handled in Stage 2 and 3.

In the first stage, Stage 1, the task is to choose which gap to merge into. A heuristic described in Section 3.2.1.3 creates the foundation for the decision about which gap to choose. Once a gap has been chosen and this gap is considered good enough to try to merge into (according to criteria described in Section 3.2.1.3), the system transitions to Stage 2 where the host car shows its intention to merge by signaling with the indicators and laterally moving towards the line that separates the left lane from the right lane. The host car should give signal for some time, allowing surrounding vehicles to be aware of its intention to merge. Therefore the system stays in Stage 2 for some time before transitioning to Stage 3 where the lane change maneuver takes place, that is, the host car moves laterally to the middle of the right lane. Finally, in Stage 4 the merging maneuver has been completed and the host car follows the car in front of it in the right lane. The system may transition from Stage 2 to Stage 1 if it turns out that the gap is not suitable (according to criteria described in Section 3.3), e.g. due to lack of cooperativeness from surrounding vehicles. Also the system may transition from Stage 3 to Stage 1 if something unexpected happens that requires the merging operation to be canceled. This transition is only an emergency transition. Figure 1.2 also illustrates the transitions between the stages.

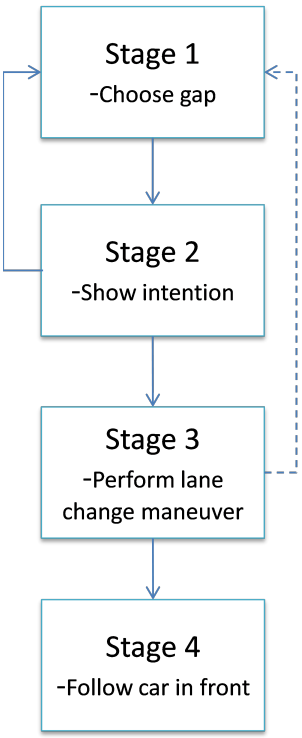


Figure 1.2: Overview of the solution. The merging decision algorithm is divided into four stages, each one responsible for a specific task.

In Stage 1 the merging situation is described with means of a set of states where each state corresponds to a position of the host car relative to the objects detected in the right lane. The state description makes it possible to use the framework of Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) which are explained in Chapter 2.

1.8 Contributions

The contributions of this thesis concerns decision making in merging situations for automated cars. The algorithm presented, denoted the *merging decision algorithm*, uses Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs) where each state in the MDP model and the POMDP model corresponds to a position of the host car relative

to the objects detected in the lane into which the host car should merge. The POMDP model, which is capable of making decisions in merging situations also when all objects detected by the sensor system cannot be classified, is solved with means of Point Based Value Iteration (PBVI) where the belief point selection strategy is specifically designed to suit the POMDP model. The algorithm is tested in a simulation environment using simulation data as well as data collected from a real traffic scenario.

2

Theory and Related Work

This chapter presents related work on decision making in traffic situations, more specifically decision making in merging situations and it outlines the theory which the merging decision algorithm is based on. In Section 2.1 the related work is presented and in Section 2.2 it is described how to design and solve a decision making problem formulated as a Markov Decision Process (MDP). There are many kinds of MDPs but only the one used in the merging decision algorithm in Stage 1 (see Section 1.7) will be discussed. In Section 2.3 it is described how to set up and interpret a Partially Observable Markov Decision Process (POMDP), also used in the merging decision algorithm in Stage 1. Section 2.3 also includes a description of a method called Point Based Value Iteration (PBVI), which is used to find approximate solutions of POMDPs.

2.1 Related Work on Decision Making in Merging Situations

Decision making for automated cars in merging situations has been investigated by several researchers. Nilsson and Sjöberg present in [2] an algorithm for decision making for lane change maneuvers where a mixed logical dynamical system is computed and then solved with means of model predictive control. The model predictive control framework is also used by Cao, Mukai and Kawabe in [3] which describes the generation of a merging path when entering a road from a ramp. In [3] no more than one car in the main lane and no more than one car in the merging lane is considered, also sensor noise is neglected.

In [4], Brechtel, Gindele and Dillman describe a general framework for decision making in traffic situations. They use an MDP model derived from discretization of a continuous state space resulting in high computational time complexity. The problem of high computational time complexity is handled by Ulbrich and Maurer in [5] with means of two so-called signal processing networks that are intended to simplify a POMDP model. The algorithm developed by Ulbrich and Maurer is used for lane change maneuvers in automated cars and was successfully tested in an automated car in real world traffic. However, the majority of the decision problem is solved

using the signal processing networks, not the POMDP. Moreover, the algorithm only consider one gap, hence the automated car does not choose between several gaps.

An MDP framework is also used in [6] by Wei et al. for single-lane autonomous driving where the host car is only controlled longitudinally. A so called point-based MDP is derived in which sensor limitations as well as the uncertain behavior of surrounding vehicles are considered. The algorithm does not deal with the merging problem but it shows how the MDP framework can be implemented and used for decision making in a specific traffic situation.

2.2 Markov Decision Processes

This section outlines the theory used when formulating the MDP model used in Stage 1 in the merging decision algorithm, a so called infinite-horizon discounted reward MDP. For a detailed description of other kinds of MDPs the reader is referred to [7].

2.2.1 MDPs and their Applications

Markov Decision Processes are widely used within artificial intelligence for modeling sequential decision-making in environments with probabilistic dynamics where an autonomous decision maker - the *agent* - has to determine actions to execute in order to achieve an objective. The applications of MDPs also extend to other areas such as operations research, computational finance, gambling theory, control theory and computational neuroscience [7, pp. 3, abstract].

Within the MDP framework it is assumed that the agent has perfect sensors. In the case of noisy sensors, the MDP can be extended to a Partially Observable Markov Decision Process (POMDP) [7, pp. 2], discussed in Section 2.3.

2.2.2 Infinite-Horizon Discounted Reward MDP

An *infinite-horizon discounted reward MDP* can be described as a tuple $\langle \mathcal{S},\mathcal{A},\mathcal{T},\mathcal{R},\gamma \rangle$ [8],[7, pp. 8, 16],[9] where:

- $\mathcal{S}$ is the finite set of all possible states of the system.
- $\mathcal{A}$ is the finite set of all actions the agent can execute.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ is the transition function where $\mathcal{T}(s,a,s')$ gives the probability of ending up in state s' when starting in state s and executing action a.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is a reward function where $\mathcal{R}(s,a,s')$ gives the utility for ending up in state s' when starting in state s and executing action a. The $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ reward function can be converted to a $\mathcal{S} \times \mathcal{A}$ function by taking the expectation over the next state,

$$\mathcal{R}(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')\mathcal{R}(s,a,s'). \tag{2.1}$$

- γ is the discount factor, $0 < \gamma < 1$.

The set of states describes the different situations that the system can be in and the set of actions describes all actions the agent can take in order to move between states. In the merging situation a state can for example describe the position of the host car relative to surrounding vehicles and the set of actions can consists of different driving commands. An MDP model typically describes an environment in which the outcome of an action is not deterministic, that is, when executing a non deterministic action the agent does not transition from one state to another with probability one. These uncertainties are described in the transition function. In the merging situation, a non-deterministic action would be "change lane". This action is non-deterministic since it is not possible to tell beforehand whether the surrounding vehicles will be cooperative when the host car tries to change lane, hence, a lane change may not succeed. The reward function describes the rewards associated with each state and each action executed. The objective for the agent is to maximize the sum of rewards gained over an infinitely long sequence of actions. The rewards are discounted with means of the discount factor γ so that rewards that can be gained closer to present time are favored to rewards that can be gained later.

Example 2.2.1 Figure 2.1 illustrates an infinite-horizon discounted reward MDP with four states and two actions. The probability p of going from one state to another state when executing an action and the reward r obtained for a successful transition are given by a tuple (p,r). For example, action a_1 takes the agent from state s_3 to state s_4 with probability 0.8 and 4 points are then received in reward.

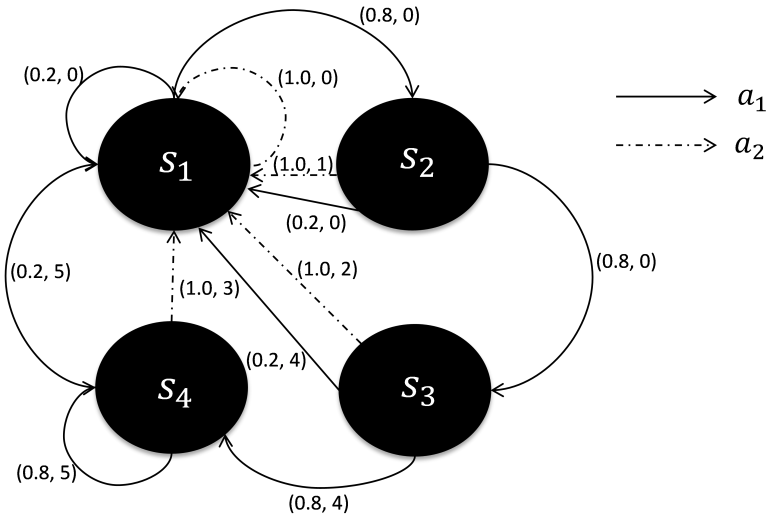


Figure 2.1: Illustration of an infinite-horizon discounted reward MDP with four states and two actions. The transition probabilities p and the rewards r are given by tuples (p,r).

2.2.3 A Policy - the Solution to an MDP

The solution to an MDP is called a *policy*, denoted π.

Definition 2.2.1 - Policy for an infinite-horizon discounted reward MDP.
A policy in an infinite-horizon discounted reward MDP maps states to actions, $\pi : \mathcal{S} \rightarrow \mathcal{A}$ [7, pp. 11].

Finding the optimal solution to an infinite-horizon discounted reward MDP means finding the *optimal policy*, π^*, which is the policy that maximizes the expected sum of discounted rewards. The optimal policy corresponds to the *optimal value function*, $V^*(s)$.

Definition 2.2.2 - Optimal policy and optimal value function for an infinite-horizon discounted reward MDP.
The optimal policy π^* and the optimal value function V^* for an infinite-horizon discounted reward MDP are defined implicitly as,

$$\begin{cases} \pi^*(s) & = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \Omega(s,a), \forall s \in \mathcal{S}, \\ V^*(s) & = \underset{a \in \mathcal{A}}{\max}\, \Omega(s,a), \forall s \in \mathcal{S}, \end{cases}$$

where $\Omega(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V^*(s')]$.

Hence, the optimal policy with respect to the value function defines the best action to execute in each state [7, pp. 17].

2.2.4 Solution Methods of MDPs

Value iteration, policy iteration and *linear programming* are three of the most popular techniques used for solving infinite-horizon discounted reward MDPs. However, policy iteration and linear programming are computationally expensive techniques when solving problems with large state spaces because they require solving systems of linear equations of the same size as the state space. Value iteration on the other hand has a recursive approach that does not suffer from this drawback and this technique is used in Stage 1 in the merging decision algorithm. The remainder of this section focuses on the value iteration algorithm [8].

2.2.4.1 The Value Iteration Algorithm

The value iteration algorithm is a dynamic programming algorithm which improves the approximation V_n of the optimal value function V^* for every iteration step n until some predefined accuracy has been reached. As the number of iteration steps n tends to infinity, V_n converges to V^*. For each iteration step the previous approximation V_{n-1} is used to compute the new approximation V_n. This update is known as a *Bellman backup* or *Bellman update* and for an infinite-horizon discounted reward MDP it is given by,

$$V_n(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V_{n-1}(s')]. \tag{2.2}$$

There are no specific requirements for the initialization step in value iteration so V_0 can be set arbitrarily for each state. When the required accuracy has been met the policy can be found by using,

$$\pi^{V_n}(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V_n(s')]. \tag{2.3}$$

The term ϵ-consistency is usually used for describing the accuracy of V_n. A value function approximation is ϵ-consistent when the maximum residual Res^{V_n} is less than ϵ,

$$Res^{V_n} = \max_{s \in \mathcal{S}} |V_n(s) - V_{n-1}(s)|. \tag{2.4}$$

A value function that is ϵ-consistent remains ϵ-consistent for all subsequent iterations [7, pp. 54, 40-43]. If $Res^{V_n} < \epsilon$ then the error $|V_n(s_0) - V^*(s_0)|$ that results from following the policy corresponding to V_n from the initial state s_0 rather than following the optimal policy from the initial state s_0 is no greater than $2\epsilon\gamma/(1-\gamma)$ [10, pp. 84].

2.3 Partially Observable Markov Decision Processes

Partially Observable Markov Decision Processes (POMDPs) is the framework to use when the agent cannot fully observe which state it is in [7, pp. 158,159]. As in the case of MDPs there are many types of POMDPs but this thesis only considers the infinite-horizon discounted reward POMDP which is the type of POMDP used in Stage 1 in the merging decision algorithm.

2.3.1 Formulation of the Infinite-Horizon Discounted Reward POMDP

The infinite-horizon discounted reward POMDP differs from the infinite-horizon discounted reward MDP in that it also has a set of observations $\mathcal{O}$, an observation function $\mathcal{Z}$ and an initial belief point b_0 which all stem from the fact that all states in the state space cannot be fully observed. An infinite-horizon discounted reward POMDP can be described as a tuple $\langle \mathcal{S},\mathcal{A},\mathcal{T},\mathcal{R},\mathcal{O},\mathcal{Z},\gamma,b_0 \rangle$ where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$ and γ have the same interpretations as in an infinite-horizon discounted reward MDP (see Section 2.2.2) and the remaining parts are defined as,

- $\mathcal{O}$ is the finite set of all observations that the agent can sense using its sensors.
- $\mathcal{Z}$ is the observation function where $\mathcal{Z}(s,a,o)$ gives the probability of observing o when the system is in state s and the last action executed was a.
- b_0 is the initial belief point which identifies the initial distribution over states such that each element, $b_0(s)$, indicates the initial probability of being in state s [11].

Since all states cannot be fully observed in a POMDP, the agent will only be able to determine the probability of being in a specific state and these probabilities are comprised in the belief point

b. If the POMDP has a state space of size $|\mathcal{S}|$ then each belief point $b = (b(s_1), b(s_2), \ldots, b(s_{|\mathcal{S}|}))$ has dimension $|\mathcal{S}|$ and each element $b(s)$ indicates the probability of being in state s. Hence every belief point is a probability distribution where every element in b is non-negative and the sum of all $|\mathcal{S}|$ elements equals unity, $\sum_{s\in\mathcal{S}} b(s) = 1$. This means that there is an infinite number of belief points[1] where the *set of belief points* is denoted Δ. A POMDP can be seen as an MDP but with a continuous state space where each state in the continuous MDP is a belief point [7, pp. 145], [12].

Example 2.3.1 A belief point in a two state POMDP has two elements where $b(s_1) = p$ indicates the probability of being in state s_1 and $b(s_2) = 1 - p$ indicates the probability of being in state s_2. Since there are only two states, once the probability of being in one of the states is known, so is the probability of being in the other state. The entire space of belief points can therefore be represented with a line ranging from zero to one, where every point on the line represents the probability of being in state s_1, see Figure 2.2. In a POMDP with many states, the representation will instead of a line be given by a hyperplane.

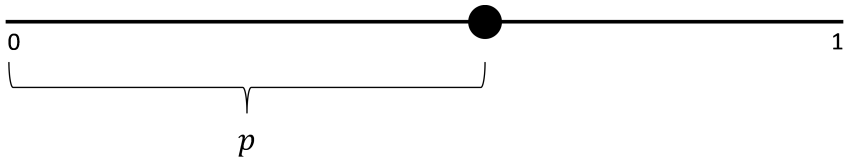


Figure 2.2: Representation of the belief space for a two state POMDP. The black dot represent the probability p of being in state s_1.

For every action that the agent executes, the probability of being in a specific state may change, hence the system may transition from one belief point to another. A belief point cannot transform to an arbitrarily belief point in just a single transition but there is a finite number of possible belief points that can be reached by a single transition. These points can be determined with means of the finite number of actions and observations. Every new belief point is associated with a combination of an action and an observation. The initial belief point may transform to the same next belief point for more than one action-observation-pair, but there will be at most $|\mathcal{A}||\mathcal{O}|$ new possible next belief points. The set of belief points that will ever be possible to reach is called the *set of reachable belief points*, denoted $\bar{\Delta}$ where $\bar{\Delta} \subseteq \Delta$ [12]. The belief point reached from b when executing action a and obtaining observation o is denoted b_a^o and is defined in (2.9).

Example 2.3.2 Figure 2.3 illustrates the scenario where a belief point (in the two state POMDP) represented with a big black dot is transformed to a new belief point represented with a small black dot. There are two actions $\mathcal{A} = \{a_1, a_2\}$ and three observations $\mathcal{O} = \{o_1, o_2, o_3\}$ available, so in total there are six new possible next belief points. The arcs in Figure 2.3 represent the transformations where the dotted arcs are associated with the transformations that result from executing action a_2 [12].

[1] The cardinality of the set of belief points is uncountably infinite since belief points are drawn from the unit hypercube $[0,1]^{|\mathcal{S}|} \subset \mathbb{R}^{|\mathcal{S}|}$ and normalized and $|\mathbb{R}^{|\mathcal{S}|}| = \left|\{b = (b(s_1), b(s_2), \ldots, b(s_{|\mathcal{S}|})) \text{ s.t. } \sum_{s\in\mathcal{S}} b(s) = 1\}\right| = \aleph_1$.

Next section describes how to solve a POMDP.

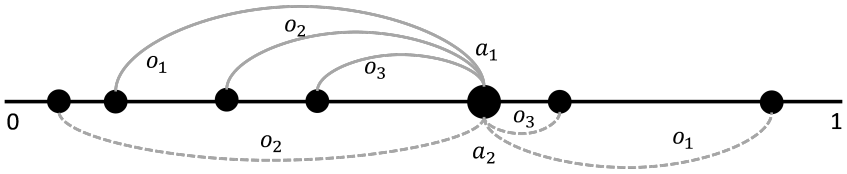


Figure 2.3: Reachable belief points when executing action a_1 or action a_2 in a two state POMDP. There are three observations available o_1, o_2 and o_3. The big black dot represent the initial belief point and the small black dots represent the belief points reachable from the initial belief from a single transition. The dotted arcs are associated with the transformations that result from executing action a_2.

2.3.2 Solving an Infinite-Horizon Discounted Reward POMDP

The value iteration step for an infinite-horizon discounted reward POMDP [11] is described as,

$$V_n(b) = \max_{a \in \mathcal{A}} \left[R(a,b) + \gamma \sum_{o \in \mathcal{O}} b_a(o) V_{n-1}(b_a^o) \right], \tag{2.5}$$

where $R(a,b)$ is the reward of executing an action a in belief point b, $b_a(o)$ is the probability of observing o after executing action a in belief point b and b_a^o is the belief point that is reached after executing action a and observing o, hence $b_a^o(s)$ is the probability of being in state s after executing action a and receiving o. The formulas for calculating these parameters are:

$$R(a,b) = \sum_{s \in \mathcal{S}} \mathcal{R}(s,a) b(s), \tag{2.6}$$

$$b_a(s) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s',a,s) b(s'), \tag{2.7}$$

$$b_a(o) = \sum_{s \in \mathcal{S}} \mathcal{Z}(s,a,o) b_a(s), \tag{2.8}$$

$$b_a^o(s) = \mathcal{Z}(s,a,o) b_a(s) / b_a(o), \tag{2.9}$$

where the same notation is used as in [10, pp. 98-100]. Note that $b_a(s)$ in (2.7) and $b_a(o)$ in (2.9) are different functions.

By iteratively using (2.5) a sequence of iterations $\{V_n(b)\}_{n=1,2...}$ is obtained. When this sequence has converged with a satisfactory accuracy the action to execute in belief point b is the action a that maximizes the value function,

$$a = \operatorname*{argmax}_{a \in \mathcal{A}} \left[R(a,b) + \gamma \sum_{o \in \mathcal{O}} b_a(o) V_n(b_a^o) \right]. \tag{2.10}$$

The value iteration algorithm used for an MDP (see (2.2)) cannot be used when solving a POMDP since a POMDP has an infinite number of belief points. However, the optimal value function V^* in a POMDP is a piecewise linear and convex function with respect to the belief space which means that the optimal value function can be expressed by a set of vectors Γ^* where each vector has $|\mathcal{S}|$ elements. These vectors are called α-vectors. With means of the optimal set Γ^*, the value of the optimal value function V^* at belief point b can be determined according to,

$$V^*(b) = \max_{\alpha\in\Gamma^*} \sum_{s\in\mathcal{S}} b(s)\alpha(s) = \max_{\alpha\in\Gamma^*} b^T\alpha, \tag{2.11}$$

which provides a way of implementing value iteration. If the initial value function V_0 is expressed with a set of α-vectors Γ_0 this set can be used to compute the set of α-vectors at the next iteration step Γ_1 which is used to express V_1 [10, pp. 99-101], [11]. By inductive reasoning on (Γ_0, V_0) and (Γ_1, V_1) the set of α-vectors Γ_{n-1} can be computed where Γ_{n-1} is used to express V_{n-1} at belief point b,

$$V_{n-1}(b) = \max_{\alpha\in\Gamma_{n-1}} \sum_{s\in\mathcal{S}} b(s)\alpha(s). \tag{2.12}$$

In utilizing (2.5) and (2.12) a POMDP iteration step can be expressed as,

$$\begin{aligned} V_n(b) &= \max_{a\in\mathcal{A}} \left[R(a,b) + \gamma \textstyle\sum_{o\in\mathcal{O}} b_a(o) V_{n-1}(b_a^o)\right] = \\ &= \max_{a\in\mathcal{A}} \left[R(a,b) + \gamma \textstyle\sum_{o\in\mathcal{O}} b_a(o) \left[\max_{\alpha\in\Gamma_{n-1}} \textstyle\sum_{s\in\mathcal{S}} b_a^o(s)\alpha(s)\right]\right] = \\ &= \max_{\alpha\in\Gamma_n} \textstyle\sum_{s\in\mathcal{S}} b(s)\alpha(s), \end{aligned} \tag{2.13}$$

[10, pp. 101-102] where the last equality can be used to obtain Γ_n from Γ_{n-1}. A more detailed explanation on how Γ_n is calculated from Γ_{n-1} is given next.

Given a set Γ_{n-1} with α-vectors, one can obtain the next set Γ_n by construction of three intermediate sets $\Gamma_n^{a,*}$, $\Gamma_n^{a,o}$ and Γ_n^a:

$$\Gamma_n^{a,*} \leftarrow \alpha^{a,*} \quad s.t. \quad \left(\alpha^{a,*}(s) = R(s,a), \forall s \in \mathcal{S}\right), \quad \forall a \in \mathcal{A}, \tag{2.14}$$

$$\Gamma_n^{a,o} \leftarrow \alpha^{a,o} \; s.t. \; \left(\alpha^{a,o}(s) = \gamma \sum_{s'\in\mathcal{S}} \mathcal{T}(s,a,s')\mathcal{Z}(s',a,o)\alpha(s'), \forall s \in \mathcal{S}\right), \forall \alpha \in \Gamma_{n-1}, \forall a \in \mathcal{A}, \forall o \in \mathcal{O}, \tag{2.15}$$

$$\Gamma_n^a = \Gamma_n^{a,*} + \bigoplus_{o\in\mathcal{O}} \Gamma_n^{a,o}, \quad \forall a \in \mathcal{A}, \tag{2.16}$$

where $\alpha^{a,*}$ and $\alpha^{a,o}$ are $|\mathcal{S}|$-dimensional hyperplanes[2], the symbol $\oplus$ is the cross-sum of sets and the arrow in the expression $\Gamma \leftarrow \alpha$ symbolizes the addition of α to the set Γ. The final set Γ_n is obtained from taking the union of the sets Γ_n^a,

[2]The $|\mathcal{S}|$-dimensional hyperplanes are given by $\alpha^{a,*} = \left(\alpha^{a,*}(s_1), \alpha^{a,*}(s_2), \ldots, \alpha^{a,*}(s_{|\mathcal{S}|})\right)^T$ and $\alpha^{a,o} = \left(\alpha^{a,o}(s_1), \alpha^{a,o}(s_2), \ldots, \alpha^{a,o}(s_{|\mathcal{S}|})\right)^T$.

$$\Gamma_n = \bigcup_{a \in \mathcal{A}} \Gamma_n^a. \tag{2.17}$$

The set Γ_n may contain as many as $|\mathcal{A}||\Gamma_{n-1}|^{|\mathcal{O}|}$ different α-vectors and has a time complexity[3] of $O(|\mathcal{S}|^2|\mathcal{A}||\Gamma_{n-1}|^{|\mathcal{O}|})$. The exponential increase in the number of α-vectors from a single iteration step is the major reason why computing Γ_n according to (2.14) - (2.17) is very time consuming, making approximation methods attractive. A popular approximating method used for solving POMDPs is called Point Based Value Iteration (PBVI) in which the value function is updated at a restricted but carefully chosen subset of belief points [10, pp. 101-102],[13],[5]. This method is described next.

2.3.3 Point Based Value Iteration

As stated in the previous section, a single iteration step for a POMDP can yield $|\mathcal{A}||\Gamma_{n-1}|^{|\mathcal{O}|}$ different α-vectors. However, some of these α-vectors are useless since they do not maximize the value function at any belief point. This is illustrated in Figure 2.4 where the dotted lines represent useless α-vectors. Every belief-point can be associated with one α-vector that maximizes the value function for that specific belief point. One α-vector may be associated with many different belief-points so if a set of belief points can be identified that automatically generates all the dominating α-vectors and at the same time excludes the useless ones, the iteration step can be less time expensive. This is the idea behind the PBVI algorithm. The main problem in PBVI is how to find the set of belief points that generates all dominating α-vectors. However, even if all belief points required to generate all of the dominating α-vector are not included in the algorithm, the main features of the optimal value function may be found which may serve as a satisfactory approximation.

In the PBVI algorithm there are many ways in which the belief points can be chosen and different conditions for when to terminate the iteration procedure can be used [10, pp. 101-103],[12]. Different strategies for selecting belief points and two common termination criteria are described in Section 2.3.3.1 and Section 2.3.3.2 respectively. Next follows an explanation of how Γ_n is computed in the PBVI algorithm.

Assuming there already exist a set $\mathcal{B}$ of strategically chosen belief points, the next set of α-vectors, Γ_n, is then obtained by generating three intermediate sets $\Gamma_n^{a,*}$, $\Gamma_n^{a,o}$ and Γ_n^a. The first two sets are identical to the definitions of $\Gamma_n^{a,*}$ and $\Gamma_n^{a,o}$ in (2.14) and (2.15) respectively. The third set however is now defined by,

$$\Gamma_n^a \leftarrow \alpha_b^a \equiv \left[\Gamma_n^{a,*} + \sum_{o \in \mathcal{O}} \arg\max_{\alpha \in \Gamma_n^{a,o}} \left(\sum_{s \in \mathcal{S}} \alpha(s) b(s) \right) \right] \quad \forall b \in \mathcal{B}, \forall a \in \mathcal{A}. \tag{2.18}$$

The final set Γ_n is then constructed from,

[3] See the Nomenclature section in the beginning of the thesis for an explanation of time complexity.

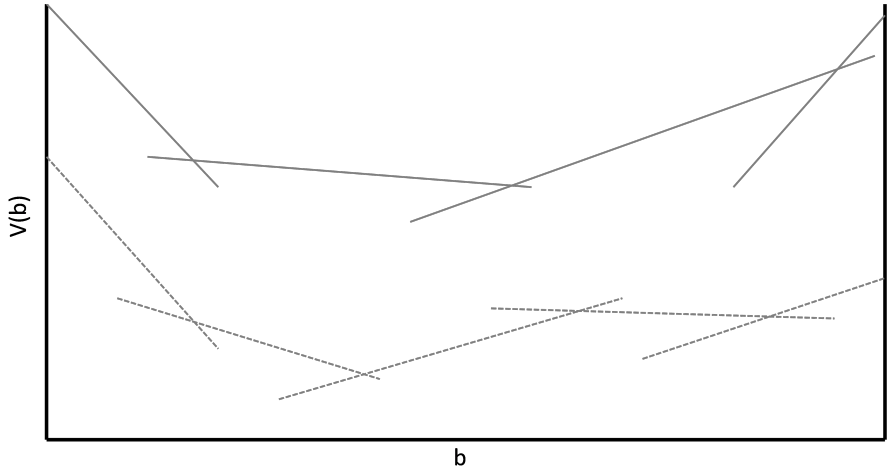


Figure 2.4: Some of the α-vectors generated from one iteration step may not maximize the value function at any belief point. The dotted lines represent such α-vectors.

$$\alpha_b = \arg\max_{\alpha_b^a \in \Gamma_n^a} \left(\sum_{s \in \mathcal{S}} \alpha_b^a(s) b(s) \right) \quad \forall b \in \mathcal{B}, \tag{2.19}$$

$$\Gamma_n = \bigcup_{b \in \mathcal{B}} \alpha_b. \tag{2.20}$$

These operations guarantee that only the best α-vector is kept for each belief point $b \in \mathcal{B}$. The two main advantages from making a point-based update instead of an exact update is that the update step is decreased to polynomial time complexity, $O(|\mathcal{S}||\mathcal{O}||\mathcal{A}||\Gamma_{n-1}||\mathcal{B}|)$. Moreover, the number of α-vectors are guaranteed not to exceed $|\mathcal{B}|$. The initial value function V_0 associated with the initial α-vector(s) $\alpha_0 \in \Gamma_0$ can be set arbitrarily [13] and one way of initializing the value function is to use a so called blind policy for which the same action is always executed. This way a lower bound on the value function is obtained [11]. Even though the PBVI algorithm guarantees that only the best α-vectors are kept for each selected belief point this does not mean that the policy (which is defined for *all* belief points) is guaranteed to improve monotonically with the number of iterations or the number of selected belief points [13].

2.3.3.1 Belief Point Selection

The task of choosing suitable belief points is crucial for the accuracy of the point-based value iteration algorithm. The more belief points selected, the higher is the computational complexity. On the other hand, more belief points included gives a higher chance of finding the optimal solution. As stated in Section 2.3.3, more than one belief point can be associated with the same α-vector so just including an arbitrary belief point does not guarantee a better approximation. There are several methods available for belief point selection and this section will further explain five heuristics all of which are described in [13].

Random Belief Selection (RA)
The RA strategy samples belief points randomly from the whole belief space Δ. It performs quite well for small problems (less than 20 states), but as the number of states grows, it cannot provide good coverage of the belief space with a reasonable number of points, resulting in poor performance.

The rest of the strategies focus on reachable belief points (previously mentioned in Section 2.3.1) rather than the whole belief space. Figure 2.5 shows the set of reachable belief points $\bar{\Delta}$ starting from an initial belief point b_0 where the available actions are $\{a_1,a_2,\ldots,a_{|\mathcal{A}|}\}$ and the available observations are $\{o_1,o_2,\ldots,o_{|\mathcal{O}|}\}$. If all reachable belief points are included, the solution at the initial belief point is guaranteed to be optimal. However, since the set of reachable belief points $\bar{\Delta}$ may grow rapidly for every step, such an approach is not very attractive. A subset of reachable belief points $\mathcal{B} \subset \bar{\Delta}$ therefore needs to be selected.

Stochastic Simulation with Random Action (SSRA)
This strategy randomly chooses a belief point that is reachable and adds it to the subset $\mathcal{B}$. First a belief point $b \in \mathcal{B}$ is randomly picked and then an action a and an observation o are randomly selected. The next belief point b_{new} is then calculated from (2.9) and added to to the set $\mathcal{B}$, that is $\mathcal{B} = \mathcal{B} \cup \{b_{new}\}$.

Stochastic Simulation with Greedy Action (SSGA)
The SSRA strategy does not consider whether the action picked is likely to be executed. This is considered in the SSGA strategy where the current best action at the randomly picked belief point b is selected with probability $1 - \epsilon$ and a random action is picked with probability ϵ.

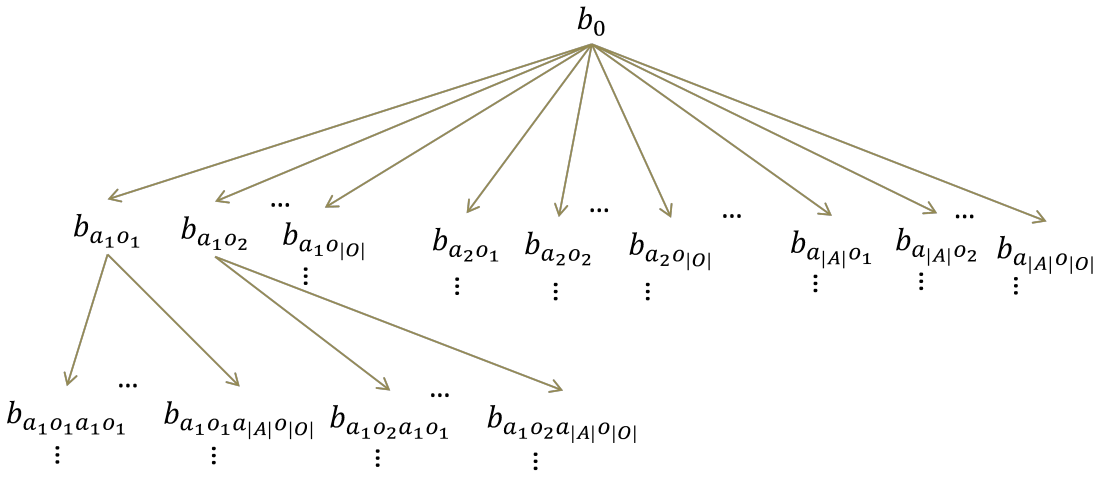


Figure 2.5: Illustration of the set of reachable belief points $\bar{\Delta}$ starting from belief point b_0. The available actions are $\{a_1,a_2,\ldots,a_{|\mathcal{A}|}\}$ and the available observations are $\{o_1,o_2,\ldots,o_{|\mathcal{O}|}\}$.

Stochastic Simulation with Exploratory Action (SSEA)
The SSEA strategy aims to sample belief points so that the distance d between belief points $b \in \mathcal{B}$ and the one step forward reachable belief points denoted $b' \in \mathcal{B}'$ is minimized. The distance is simply the L_1 norm,

$$d = \sum_{s \in \mathcal{S}} |b(s) - b'(s)| = ||b - b'||_1.$$

Greedy Error Reduction(GER)
The GER strategy focuses on minimizing the maximum expected error between the optimal value function and the approximation of ditto. Since the exact error is not known a bound on the error is used. The belief point b added to the set $\mathcal{B}$ is the belief point that maximizes the error bound reduction. For a more detailed explanation of this strategy the reader is referred to [13] which recommends this strategy even though it has higher computational complexity than the previously described strategies.

2.3.3.2 Termination Criteria

As described in Section 2.2.4 a termination criterion for the MDP value iteration algorithm is easily computed by comparing the value for each state from two subsequent iterations (previously described in Section 2.2.4). This is not possible for the POMDP model due to the unlimited number of belief points. A simple termination criterion that is commonly used within the POMDP community is to predefine the number of iterations and iterate this number of times. A drawback of this approach is that it is not possible to tell how close the final approximation is to the optimal value function without additional time consuming calculations [13].

Another approach is to compute an upper and lower bound on the optimal value function and then terminate when the maximum difference between the upper and lower bound is smaller than some predefined limit. Alternatively, the termination criterion could be a maximum number of iterations and when this limit has been reached, the accuracy of the final approximation can be determined using upper and lower bounds [11]. This approach may seem more attractive, but updating the upper bound requires solving linear programs which is computationally expensive. Hence, if time is the most limiting factor this approach may not be the best alternative.

3

The Merging Decision Algorithm

This chapter describes the merging decision algorithm. First Section 3.1 presents some terminology used when describing the algorithm and then Section 3.2, Section 3.3 and Section 3.4 describe in detail how Stage 1, Stage 2 and Stage 3 are implemented. Section 3.5 describes how all stages are put together to form the merging decision algorithm. An overview of the merging decision algorithm including a description of all stages was given in Section 1.7.

3.1 Terminology

Figure 3.1 shows an example of a merging situation for an automated car where five objects ($object_1 - object_5$) have been detected in the right lane and these are numbered in increasing order in the driving direction such that the object closest to the end-point has the highest object number. Out of the five objects, four are classified as cars and one is classified as a *suspected ghost-car*. The *status of the suspected ghost-car* in the example is *real car*. In total there are six *gaps* available, illustrated with circles in Figure 3.1. The exact definition of *suspected ghost-car*, *status of a suspected ghost-car* and *gap* are given next.

Definition 3.1.1 - Suspected ghost-car
A suspected ghost-car is a detected object that cannot be classified as a car with high probability and is either an object that can be neglected in the the context of merging or a car.

Definition 3.1.2 - Status of a suspected ghost-car
A suspected ghost-car has a status that is either *real car* or *ghost-car* where the *real car* status indicates that the true situation is that the suspected ghost-car is a real car and the *ghost-car* status means that the true situation is that the suspected ghost-car is a ghost-car and can be neglected.

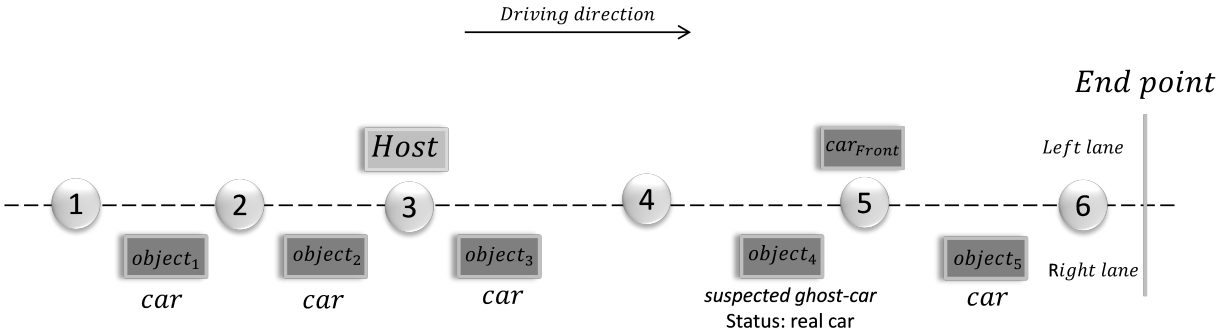


Figure 3.1: A merging situation with five objects detected in the right lane where four are classified as cars and one is a suspected ghost-car with status *real car*. There are six gaps available, illustrated with circles and numbered in increasing order in the driving direction.

Definition 3.1.3 - Gap
A gap refers to the space between two subsequent objects detected in the right lane. In the merging decision algorithm it is also said to be a gap behind the object with the lowest object number and in front of the object with the highest object number. Gaps are numbered in increasing order in the driving direction such that the gap with the highest gap number is closest to the end-point. The size of a gap may change depending on the status of a suspected ghost-car such that if object j is a suspected ghost-car with status *ghost-car* then the size of gap j equals the size of gap $j+1$. Hence, in this situation gap j and gap $j+1$ both refer to the same space. Note that the number of gaps only depends on the number of objects detected but that the size of a gap depends on the status of suspected ghost-cars.

A suspected ghost-car is denoted $car_{?m}$ where $m = \{1,2,\ldots,N_{unsure}\}$ and N_{unsure} is the total number of suspected ghost-cars. A higher value of m indicates that the suspected ghost-car is detected closer to the end-point.

In the description of the merging decision algorithm the term *master function* is also used which is defined below.

Definition 3.1.4 - Master function
In the merging decision algorithm the master function refers to the function that provides the merging decision algorithm with all inputs required as well as actualizes the decisions into driving commands.

3.2 Stage 1

Stage 1 in the merging decision algorithm deals with decision 1 and 2 stated in Section 1.6, hence the objective is to decide which gap to merge into and when to merge into the gap. Stage 1 is solved both with means of a POMDP model and an MDP model, both of which are frequently

computed and solved. The MDP model assumes that all suspected ghost-cars have status *real car*, whereas this is not assumed in the POMDP model. Section 3.2.1 and Section 3.2.2 present the POMDP model and the MDP model respectively and these sections utilize the theory from Section 2.3.1 and Section 2.2.2. Section 3.2.3 describes how an ϵ-consistent solution of the MDP model is found using the theory from Section 2.2.4 and it is also described how the POMDP model is solved approximately using the theory from Section 2.3.2. Finally Section 3.2.4 explains how all parts included in Stage 1 are put together.

3.2.1 Description of the POMDP model

This section defines the POMDP model used in Stage 1 in the merging decision algorithm. The type of POMDP used is an infinite-horizon discounted reward POMDP, described in Section 2.3.1 and therefore the state space, the action space, the reward function, the transition function, the observation space, the observation function, the initial belief point and the discount factor need to be described. Section 3.2.1.1 - Section 3.2.1.8 describe all of these components.

3.2.1.1 State space $\mathcal{S}$

The state space of the POMDP model is described with three state variables: X_{lane}, X_{gap} and X_{comb} which are defined as follows:

- X_{lane} indicates which lane the host car can be in, that is $X_{lane} \in \{Left, Right\}$.
- X_{gap} indicates which gap the host car can be in or adjacent to. If there are $N_{gaps} - 1$ objects detected in the right lane resulting in N_{gaps} gaps then $X_{gap} \in \{1, 2, \ldots, N_{gaps}\}$.
- X_{comb} indicates all possible combinations of the status of suspected ghost-cars. The set of all possible values that X_{comb} can assume, X_{comb}^{vec}, contains $N_{comb} = 2^{N_{unsure}}$ vectors[1] $x_{comb_k}^{vec} \in \mathbb{B}^{N_{unsure}}$ each of which can be represented as a binary number[2] $x_{comb_k} = (x_{comb_k}^{vec})_2$ where $x_{comb_k} < x_{comb_{k+1}}$ and $k \in \{1,2,\ldots,N_{comb}\}$. Also $x_{comb_k}(m) = 0$ indicates that $car_{?m}$ has status *ghost-car* and $x_{comb_k}(m) = 1$ indicates that $car_{?m}$ has status *real car*. From now on x_{comb_k} will be used both to denote the binary number and the vector representation of said number. The distinction will be clear from the context.

Example 3.2.1 If there are two suspected ghost-cars, $car_{?1}$ and $car_{?2}$ then $X_{comb}^{vec} = \{x_{comb_1}, x_{comb_2}, x_{comb_3}, x_{comb_4}\} = \{00_2, 01_2, 10_2, 11_2\}$ where for example 00_2 means that both $car_{?1}$ and $car_{?2}$ have status *ghost-car* and 10_2 means that $car_{?1}$ has status *real car* and $car_{?2}$ has status *ghost-car*.

A state in the state space consists of a state variable triple $(X_{lane}, X_{gap}, X_{comb})$ which means that the total number of states equals $|\mathcal{S}| = 2N_{gaps}N_{comb}$. The state space is ordered in the following way,

[1] $\mathbb{B}^N$ is the binary space of dimension N, $\mathbb{B}^N = \{X = (x_1, x_2, \ldots, x_N) | x_i \in \{0,1\}, i = 1,2,\ldots,N\}$.

[2] A number in binary form is represented with number two as subscript, e.g. $11_2 = 3$ where "3" is in base ten.

$$X_{lanes} = \begin{cases} Left \text{ in state } s_n & \text{, if } n \leq |\mathcal{S}|/2, \\ Right \text{ in state } s_n & \text{, if } n > |\mathcal{S}|/2, \end{cases} \tag{3.1}$$

$$X_{gap} = \quad j \text{ in state } s_n, \quad \text{, where } j = 1 + (n-1) \bmod N_{gaps}, \tag{3.2}$$

$$X_{comb} = \quad x_{comb_{(1+K \bmod N_{comb})}} \text{ in state } s_n \quad \text{, where } K = N_{gaps}^{-1}\Big((n-1) - (n-1) \bmod N_{gaps}\Big), \tag{3.3}$$

where $s_n \in \mathcal{S}$, $n \in \{1,2,\ldots,|\mathcal{S}|\}$.

Example 3.2.2 If there are six gaps available ($N_{gaps} = 6$) and two objects are suspected ghost-cars ($N_{unsure} = 2$) then the state space is defined as follows,

$$\mathcal{S} = \begin{cases} s_1 : (Left,1,00_2) & s_{13} : (Left,1,10_2) & s_{25} : (Right,1,00_2) & s_{37} : (Right,1,10_2) \\ s_2 : (Left,2,00_2) & s_{14} : (Left,2,10_2) & s_{26} : (Right,2,00_2) & s_{38} : (Right,2,10_2) \\ s_3 : (Left,3,00_2) & s_{15} : (Left,3,10_2) & s_{27} : (Right,3,00_2) & s_{39} : (Right,3,10_2) \\ s_4 : (Left,4,00_2) & s_{16} : (Left,4,10_2) & s_{28} : (Right,4,00_2) & s_{40} : (Right,4,10_2) \\ s_5 : (Left,5,00_2) & s_{17} : (Left,5,10_2) & s_{29} : (Right,5,00_2) & s_{41} : (Right,5,10_2) \\ s_6 : (Left,6,00_2) & s_{18} : (Left,6,10_2) & s_{30} : (Right,6,00_2) & s_{42} : (Right,6,10_2) \\ s_7 : (Left,1,01_2) & s_{19} : (Left,1,11_2) & s_{31} : (Right,1,01_2) & s_{43} : (Right,1,11_2) \\ s_8 : (Left,2,01_2) & s_{20} : (Left,2,11_2) & s_{32} : (Right,2,01_2) & s_{44} : (Right,2,11_2) \\ s_9 : (Left,3,01_2) & s_{21} : (Left,3,11_2) & s_{33} : (Right,3,01_2) & s_{45} : (Right,3,11_2) \\ s_{10} : (Left,4,01_2) & s_{22} : (Left,4,11_2) & s_{34} : (Right,4,01_2) & s_{46} : (Right,4,11_2) \\ s_{11} : (Left,5,01_2) & s_{23} : (Left,5,11_2) & s_{35} : (Right,5,01_2) & s_{47} : (Right,5,11_2) \\ s_{12} : (Left,6,01_2) & s_{24} : (Left,6,11_2) & s_{36} : (Right,6,01_2) & s_{48} : (Right,6,11_2) \ . \end{cases}$$

Thus, for this state space example there are 48 states in total where for example $s_9 = (Left, 3, 01_2)$ represents the host car being in the left lane adjacent to gap 3 and out of the two suspected ghost-cars, the first one, $car_{?1}$, is a ghost-car whereas the second one, $car_{?2}$, is a real car.

3.2.1.2 Action space $\mathcal{A}$

The action space $\mathcal{A}$ consists of four actions $a_i, i \in \{1,2,3,4\}$ defined as follows,

- a_1 : change lane,
- a_2 : stay at the current gap,
- a_3 : drive to the closest gap in front,

- a_4 : wait for the closest gap behind.

If the action to be executed in Stage 1 is a_1 then the system transitions from Stage 1 to Stage 2. Hence, even though a_1 is defined as "change lane" in the POMDP model, it is used as an action indicating that the host car should start showing its intention to merge. Action a_2 should be interpreted as staying by a gap, that is the host car should move as fast as the midpoint of the gap it is currently adjacent to. When executing action a_3 the host car may drive at its maximum allowed speed v_{max} and when executing action a_4 the host car may slow down to its minimum allowed speed v_{min}. Both v_{max} and v_{min} are given by the master function.

3.2.1.3 Reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

Equations (3.7)-(3.10) define the rewards given when starting in state s and ending up in state s' after executing any of the actions a_1, a_2, a_3 or a_4. The reward function is computed as an $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ matrix and then converted to an $|\mathcal{S}| \times |\mathcal{A}|$ matrix using (2.1).

Table 3.1 and Table 3.2 describe the variables and tuning parameters used in (3.7)-(3.10) and Figure 3.2 illustrates the distances $sizeGap(j,x_{comb_k})$ and $distMidEnd(j,x_{comb_k})$, for $j = 4$ and $x_{comb_k} = 1_2$. Figure 3.3 illustrates the same distances but for $j = 4$ and $x_{comb_k} = 0_2$. Next, the variables $sizeGap(j,x_{comb_k})$ and $distMidEnd(j,x_{comb_k})$ are explained further.

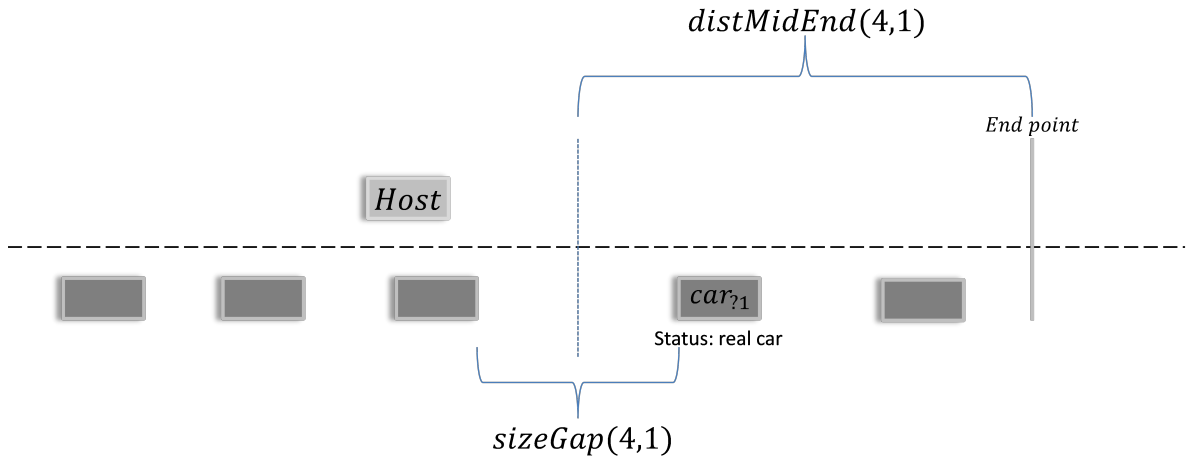


Figure 3.2: Illustration of distances $sizeGap(j,x_{comb_k})$ and $distMidEnd(j,x_{comb_k})$ for $j = 4$ and $x_{comb_k} = 1_2$. Hence gap 4 is considered and the suspected ghost-car has status *real car*.

In order to calculate $sizeGap(j,x_{comb_k})$ and $distMidEnd(j,x_{comb_k})$ Stage 1 is given a vector denoted the *certainity vector* defined as:

Definition 3.2.1 - Certainty vector
The certainty vector, denoted *certainty*, indicates whether an object is classified as a car or a suspected ghost-car. The certainty vector is defined as,

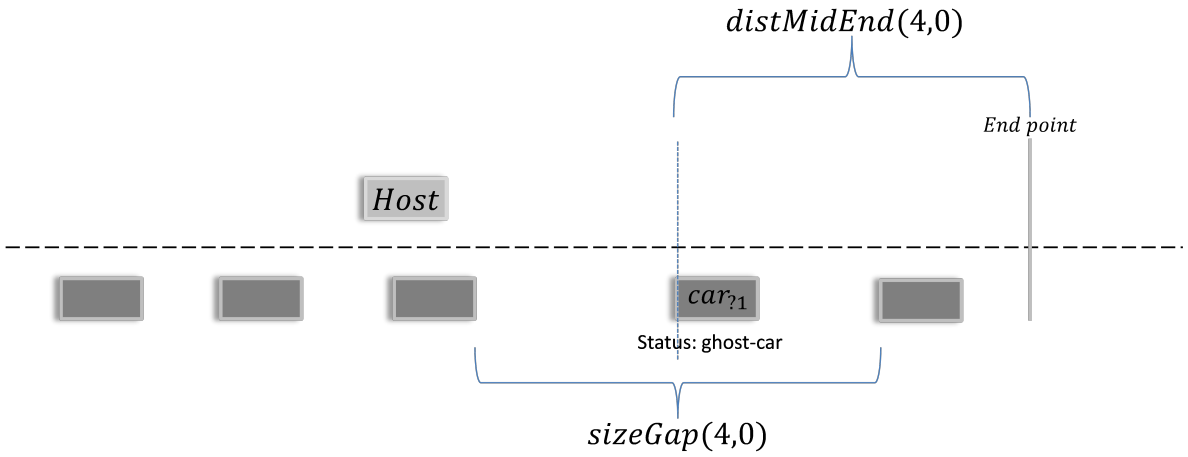


Figure 3.3: Illustration of distances $sizeGap(j,x_{comb_k})$ and $distMidEnd(j,x_{comb_k})$ for $j = 4$ and $x_{comb_k} = 0_2$. Hence gap 4 is considered and the suspected ghost-car has status *ghost-car*.

$$certainty(m) = \begin{cases} 1 & \text{, if } object_m \text{ is classified as a car,} \\ 0 & \text{, if } object_m \text{ is classified as a ghost-car,} \end{cases} \tag{3.4}$$

where $m \in \{1,2\ldots,N_{gaps}-1\}$.

The values in $sizeGap(j,x_{comb_k})$ are calculated according to,

$$sizeGap(j,x_{comb_k}) = \begin{cases} gapsize(j) + gapsize(j+1) + CarSize + \eta & \text{, if } \exists i \text{ s.t. } (j = I(i)) \wedge (x_{comb_k}(i) = 0) \\ gapsize(j) + gapsize(j-1) + CarSize & \text{, if } \exists i \text{ s.t. } (j-1 = I(i)) \wedge (x_{comb_k}(i) = 0) \\ gapsize(j) & \text{, otherwise} \end{cases} \tag{3.5}$$

where $gapsize(j)$ is the size of gap j if all suspected ghost-cars have status *real car*, $CarSize$ is a constant that equals the (average) length of a car, η is a small positive constant used in order to favor the gap with the lower gap number and I is an ordered list[3] s.t. $I = \text{orderedList}[\{m|certainty(m) = 0\}]$. The size of the first gap and the last gap in $gapsize$ are set such that $gapsize(1) < gapsize(2)$ and $gapsize(N_{gaps}) < gapsize(N_{gaps}-1)$. This is done in order to make the first gap and the last gaps unfavorable to merge into since it is assumed that the host car cannot detect objects behind the object with the lowest object number or in front of the object with the highest object number.

The values in $distMidEnd(j,x_{comb_k})$ are calculated according to,

[3] An ordered list is here defined as a list that can be indexed and where $I(i) \leq I(i+1)\ \forall i$

Table 3.1: Description of variables used in (3.7)-(3.10).

$sizeGap(j,x_{comb_k})$	The size of gap j if the status of the suspected ghost-cars are as indicated by x_{comb_k}.
$distMidEnd(j,x_{comb_k})$	The distance from the midpoint of gap j to the end-point if the status of the suspected ghost-cars are as indicated by x_{comb_k}.
$distMidHost(j)$	The distance from the midpoint of gap j to the host car if all suspected ghost-cars have status *real car*.
$distMidFront(j)$	The distance from the midpoint of gap j to $vehicle_{Front}$ if all suspected ghost-cars have status *real car*.
$t^{Mid,Front}(j)$	The estimated time for the midpoint of gap j to reach $vehicle_{Front}$ if all suspected ghost-cars have status *real car*.
$t^{Mid,End}(j)$	The estimated time for the midpoint of gap j to reach the end-point if all suspected ghost-cars have status *real car*.
$t^{Host,Front}$	The estimated time for the host car to reach $vehicle_{Front}$.
$t^{Host,End}_{min}$	The lowest estimated time for the host car to reach the end-point, given by driving at velocity v_{max}.
$t^{Host,End}_{max}$	The highest estimated time for the host car to reach the end-point, given by driving at velocity v_{min}.

Table 3.2: Tuning parameters used in (3.7)-(3.10).

$DistSafetyFront$	Minimum distance required between host car and $vehicle_{Front}$.
$GapSafetyLC$	Minimum size of a gap required in order to obtain a non-zero reward for execution of action a_1.
$T^{Host,Front}$	Minimum time required between host car and $vehicle_{Front}$.
$T^{Mid,Front}_S$	Minimum time required for the midpoint of a gap to reach $vehicle_{Front}$. Used with action a_2.
$T^{Mid,Front}_{LC}$	Minimum time required for the midpoint of a gap to reach $vehicle_{Front}$. Used with action a_1.
G_X	Weights, where $X \in \{LC, SG, SE, FG, FE, FM, WG, WE, WM\}$.

$$distMidEnd(j,x_{comb_k}) = \begin{cases} \Big(distmidend(j) + distmidend(j+1)\Big)/2 & \text{, if } \exists i \text{ s.t. } \Big(j = I(i)) \wedge (x_{comb_k}(i) = 0\Big) \\ \Big(distmidend(j) + distmidend(j-1)\Big)/2 & \text{, if } \exists i \text{ s.t. } \Big(j-1 = I(i)) \wedge (x_{comb_k}(i) = 0\Big) \\ distmidend(j) & \text{, otherwise} \end{cases} \tag{3.6}$$

where $distmidend(j)$ is the distance from the midpoint of gap j to the end-point if all suspected ghost-cars have status *real car* and I is as previously described.

The reward function on form $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ is defined in (3.7)-(3.10) for all actions, where it can be seen that a non-zero reward is only given if a corresponding condition (denoted B_{LC}, B_S, B_F and B_W) is $True$. As an example, for action a_1 this condition is denoted B_{LC} (LC for lane change). Hence, a gap is only considered good enough to try to merge into if B_{LC} is $True$. In (3.7)-(3.10) it can also be seen that a non-zero reward cannot be obtained if the host car is in or adjacent to gap 1 or gap N_{gaps} since it is assumed that the host car cannot detect objects behind the object with the lowest object number or in front of the object with the highest object number.

$$\mathcal{R}(s,a_1,s') = \begin{cases} \max\Big(rew_{LC}(j,x_{comb_k}),0\Big) & \text{, if } \Big(B_{LC}(j,x_{comb_k}) = True\Big) \wedge \Big(s = (Left,j,x_{comb_k})\Big) \wedge \\ & \qquad \wedge \Big(s' = (Right,j,x_{comb_k})\Big), \\ 0 & \text{, otherwise,} \end{cases}$$

where,

$$\begin{aligned} &rew_{LC}(j,x_{comb_k}) = G_{LC} \cdot rew_S(j, x_{comb_k}) \quad (rew_S \text{ is defined in (3.8)}), \\ &B_{LC}(j,x_{comb_k}) = \Big(sizeGap(j,x_{comb_k}) > GapSafetyLC\Big) \wedge \Big(t^{Mid,Front}(j) > T_{LC}^{Mid,Front}\Big) \wedge \\ &\qquad \wedge \Big(distMidFront(j) > DistSafetyFront\Big) \wedge \Big(j \neq N_{gaps}\Big) \wedge \Big(j \neq 1\Big), \\ &j \in \{1,2,\ldots,N_{gaps}\}, k \in \{1,2,\ldots,N_{comb}\}. \end{aligned} \tag{3.7}$$

$$\mathcal{R}(s,a_2,s') = \begin{cases} \max\Big(rew_S(j,x_{comb_k}),0\Big) & \text{, if } \Big(B_S(j) = True\Big) \wedge \Big(s = (Left,j,x_{comb_k})\Big) \wedge \\ & \qquad \wedge \Big(s' = (Left,j,x_{comb_k})\Big), \\ 0 & \text{, otherwise,} \end{cases}$$

where,

$$\begin{aligned} &rew_S(j,x_{comb_k}) = G_{SG} \cdot sizeGap(j,x_{comb_k}) + G_{SE} \cdot distMidEnd(j,x_{comb_k}) \\ &B_S(j) = \Big(t^{Mid,Front}(j) > T_S^{Mid,Front}\Big) \wedge \Big(t^{Host,Front} > T^{Host,Front}\Big) \wedge \\ &\qquad \wedge \Big(distMidFront(j) > DistSafetyFront\Big) \wedge \Big(j \neq N_{gaps}\Big) \wedge \Big(j \neq 1\Big), \\ &j \in \{1,2,\ldots,N_{gaps}\}, k \in \{1,2,\ldots,N_{comb}\}. \end{aligned} \tag{3.8}$$

$$\mathcal{R}(s,a_3,s') = \begin{cases} \max\Big(rew_F(j,x_{comb_k}),0\Big) & \text{, if } \Big(B_F(j) = True\Big) \wedge \Big(s = (Left,j,x_{comb_k})\Big) \wedge \\ & \wedge \Big(s' = (Left,j+1,x_{comb_k})\Big), \\ 0 & \text{, otherwise,} \end{cases}$$

where,

$$\begin{aligned} rew_F(j,x_{comb_k}) &= G_{FG} \cdot sizeGap(j+1,x_{comb_k}) + G_{FE} \cdot distMidEnd(j+1,x_{comb_k}) - \\ &\quad - G_{FM} \cdot |distMidHost(j+1)|, \\ B_F(j) &= \Big(t_{min}^{Host,End} < t^{Mid,End}(j+1)\Big) \wedge \Big(j+1 \neq N_{gaps}\Big) \wedge \\ &\quad \wedge \Big(j \neq N_{gaps}\Big) \wedge \Big(B_S(j+1)\Big), \\ j &\in \{1,2,\ldots,N_{gaps}\}, k \in \{1,2,\ldots,N_{comb}\}. \end{aligned} \tag{3.9}$$

$$\mathcal{R}(s,a_4,s') = \begin{cases} \max\Big(rew_W(j,x_{comb_k}),0\Big) & \text{, if } \Big(B_W(j) = True\Big) \wedge \Big(s = (Left,j,x_{comb_k})\Big) \wedge \\ & \wedge \Big(s' = (Left,j-1,x_{comb_k})\Big), \\ 0 & \text{, otherwise,} \end{cases}$$

where,

$$\begin{aligned} rew_W(j,x_{comb_k}) &= G_{WG} \cdot sizeGap(j-1,x_{comb_k}) + G_{WE} \cdot distMidEnd(j-1,x_{comb_k}) - \\ &\quad - G_{WM} \cdot |distMidHost(j-1)|, \\ B_W(j) &= \Big(t_{max}^{Host,End} > t^{Mid,End}(j-1)\Big) \wedge \Big(j-1 \neq 1\Big) \wedge \\ &\quad \wedge \Big(j \neq 1\Big) \wedge \Big(B_S(j-1)\Big), \\ j &\in \{1,2,\ldots,N_{gaps}\}, k \in \{1,2,\ldots,N_{comb}\}. \end{aligned} \tag{3.10}$$

3.2.1.4 Transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$

In the transition function all actions except action a_1 are set to be deterministic, meaning that for action a_2, a_3 and a_4 the host car is guaranteed to be able to go to/stay at the gap it intends to but this is not true for action a_1. Action a_1 which should take the host car back and forth between the left lane and the right lane is stochastic since it is not certain that the merging will succeed e.g. due to lack of cooperation from surrounding vehicles. Equations (3.13)-(3.16) define the transition probabilities for action a_1, a_2, a_3 and a_4 respectively. First the transition probabilities for the first action are described in detail.

Transition probabilities for action a_1

The probability of being able to change lane, from left to right, is set to a low number denoted

$p_{low} \in$ (0,0.3) if the Boolean $B_{LC}(j, x_{comb_k})$ (defined in (3.7)) is $False$. If $B_{LC}(j, x_{comb_k}) = True$ the matrix $prob_{LC} \in \mathbb{R}^{N_{gaps} \times N_{comb}}$ is used. The probabilities in $prob_{LC}$ are set with means of a vector $prob$ and a matrix u_{LC}, where $prob = (0.95, (0.95 - d), (0.95 - 2d), \ldots, (0.95 - (N_{gaps} - 1)d))^T$ contains probabilities in descending order set with a fixed interval $0 < d < (0.95 - p_{low})/(N_{gaps} - 1)$ and u_{LC} is the utility of a lane change such that,

$$u_{LC}(j, x_{comb_k}) = \begin{cases} -\Big(G_U \cdot vMidRel(j) + 1/rew_S(j,x_{comb_k})\Big) & \text{, if } vMidRel(j) > 0, \\ -\Big(G_U \cdot vMidRel(j) - rew_S(j,x_{comb_k})\Big) & \text{, otherwise,} \end{cases} \tag{3.11}$$

where $j = \{1,2,\ldots,N_{gaps}\}$, $rew_S(j,x_{comb_k})$ is defined as in (3.8), G_U is a weight and $vMidRel(j)$ is the relative velocity between the midpoints of gap j and gap $j+1$. If $vMidRel(j) > 0$ this means that the midpoint of gap j is moving faster than the midpoint of gap $j+1$. A higher value of $u_{LC}(j,x_{comb_k})$ is used as an indicator of a higher probability of being able to change lane. A column in u_{LC} is denoted u_{LC_k} where $u_{LC_k} = (u_{LC}(1,x_{comb_k}), u_{LC}(2,x_{comb_k}), \ldots, u_{LC}(N_{gaps},x_{comb_k}))^T$ and in using $prob$ and u_{LC_k} each column $prob_{LC,k}$ in $prob_{LC}$ is calculated as,

$$prob_{LC,k} = M_{perm,k} prob, \tag{3.12}$$

where $M_{perm,k}$ is a permutation matrix, which is not necessarily unique s.t. $M_{perm,k} u_{LC,k} = u_{LC,k}^{sorted}$ where $u_{LC,k}^{sorted}$ is sorted in descending order such that $u_{LC,k}^{sorted}(j) \geq u_{LC,k}^{sorted}(j+1)$.

Note that a gap given the highest probability of a lane change is not necessarily the gap that will be chosen in Stage 1 since this gap may for example be far away from the host car. The transition probabilities for action a_1 are given by,

$$\mathcal{T}(s,a_1,s') = \begin{cases} p_{low} & \text{, if } B_{LC} = False \wedge s = (Left, j, x_{comb_k}) \wedge s' = (Right, j, x_{comb_k}), \\ 1 - p_{low} & \text{, if } B_{LC} = False \wedge s = (Left, j, x_{comb_k}) \wedge s' = (Left, j, x_{comb_k}), \\ prob_{LC}(j, x_{comb_k}) & \text{, if } B_{LC} = True \wedge s = (Left, j, x_{comb_k}) \wedge s' = (Right, j, x_{comb_k}), \\ 1 - prob_{LC}(j, x_{comb_k}) & \text{, if } B_{LC} = True \wedge s = (Left, j, x_{comb_k}) \wedge s' = (Left, j, x_{comb_k}), \\ 1 & \text{, if } s = (Right, j, x_{comb_k}) \wedge s' = (Left, j, x_{comb_k}), \\ 0 & \text{, otherwise,} \end{cases} \tag{3.13}$$

where $j \in \{1,2,\ldots,N_{gaps}\}$, $k \in \{1,2,\ldots,N_{comb}\}$.

Transition probabilities for action a_2, a_3 and a_4
The transition probabilities for action a_2, a_3 and a_4 are calculated as,

$$\mathcal{T}(s,a_2,s') = \begin{cases} 1 & \text{, if } s = (i,j,x_{comb_k}) \wedge s' = (i,j,x_{comb_k}), \\ 0 & \text{, otherwise,} \end{cases} \tag{3.14}$$

$$\mathcal{T}(s,a_3,s') = \begin{cases} 1 & \text{, if } s = (Left,j,x_{comb_k}) \wedge s' = (Left,j+1,x_{comb_k}) \wedge j \neq N_{pos}, \\ 1 & \text{, if } s = (Left,j,x_{comb_k}) \wedge s' = (Left,j,x_{comb_k}) \wedge j = N_{pos}, \\ 1 & \text{, if } s = (Right,j,x_{comb_k}) \wedge s' = (Right,j,x_{comb_k}), \\ 0 & \text{, otherwise,} \end{cases} \tag{3.15}$$

$$\mathcal{T}(s,a_4,s') = \begin{cases} 1 & \text{, if } s = (Left,j,x_{comb_k}) \wedge s' = (Left,j-1,x_{comb_k}) \wedge j \neq 1, \\ 1 & \text{, if } s = (Left,j,x_{comb_k}) \wedge s' = (Left,j,x_{comb_k}) \wedge j = 1, \\ 1 & \text{, if } s = (Right,j,x_{comb_k}) \wedge s' = (Right,j,x_{comb_k}), \\ 0 & \text{, otherwise,} \end{cases} \tag{3.16}$$

where $i \in \{Left, Right\}$, $j \in \{1,2,\ldots,N_{gaps}\}$ and $k \in \{1,2,\ldots,N_{comb}\}$. From inspecting (3.14) - (3.16) it can be seen that actions a_2, a_3 and a_4 are deterministic, as previously stated.

Example 3.2.3 Figure 3.4 illustrates a POMDP representing a merging situation with five gaps and one suspected ghost-car. States where $X_{lane} = Right$ are indicated with an 'R' and states where $X_{lane} = Left$ are indicated with an 'L'. The transitions are illustrated with arrows but the transition probabilities are not shown. In Figure 3.4 it can be seen that if the host car starts in state s_1 and executes action a_3 then it is taken from state s_1 to state s_2 where s_1 represents the host car being in the left lane by gap 1 and the suspected ghost car has status *ghost-car* and state s_2 represents the host car remaining in the left lane but by gap 2 and the suspected ghost car is still a ghost-car. The non deterministic behavior of action a_1 is also illustrated in Figure 3.4. It can be seen that if the host car starts in state s_1 and executes action a_1 then it is either taken from state s_1 to state s_{11} or it remains in state s_1.

3.2.1.5 Observation space $\mathcal{O}$

The observation space of the POMDP model consists of $2N_{gaps}$ number of observations and is described by means of the state variables X_{lane} and X_{gap} defined in Section 3.2.1.1. An observation in the observation space consists of a state variable pair (X_{lane}, X_{gap}) which are ordered in the following way,

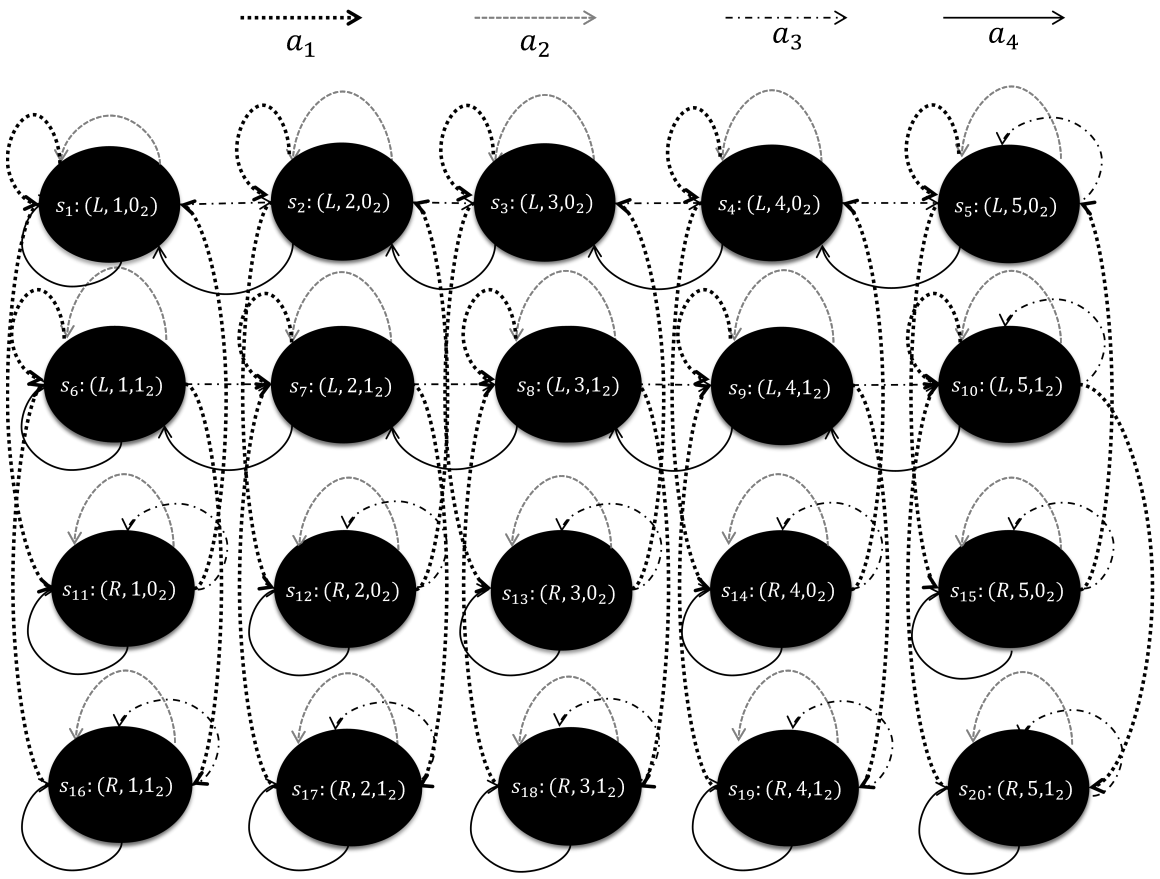


Figure 3.4: Illustration of the POMDP model in Stage 1 when there are five gaps and one suspected ghost-car. The transitions are illustrated with arrows.

$$
\begin{aligned}
X_{lanes} &= \begin{cases} Left \text{ in observation } o_n & \text{, if } n \leq |\mathcal{O}|/2, \\ Right \text{ in observation } o_n & \text{, if } n > |\mathcal{O}|/2, \end{cases} \\
X_{gap} &= j \text{ in observation } o_n \quad \text{, where } j = 1 + (n-1) \bmod N_{gaps},
\end{aligned}
$$

where $o_n \in \mathcal{O}$, $n \in \{1, \ldots, |\mathcal{O}|\}$.

Example 3.2.4 If there are six gaps available ($N_{gaps} = 6$) then the observation space is defined as,

$$\mathcal{O} = \begin{cases} o_1 : (Left,1) & o_7 : (Right,1) \\ o_2 : (Left,2) & o_8 : (Right,2) \\ o_3 : (Left,3) & o_9 : (Right,3) \\ o_4 : (Left,4) & o_{10} : (Right,4) \\ o_5 : (Left,5) & o_{11} : (Right,5) \\ o_6 : (Left,6) & o_{12} : (Right,6) \ . \end{cases}$$

3.2.1.6 Observation function $\mathcal{Z} : \mathcal{S} \times \mathcal{O} \rightarrow [0,1]$

It is assumed that the host car is able to sense in which lane it is and at which gap it is in or adjacent to but it cannot determine the status of a suspected ghost-car. Also, there is no action that the host car can execute in order to better determine the status. Therefore the observation function is independent of the action taken and it is also deterministic. The observation function is defined as,

$$\mathcal{Z}(s,o) = \begin{cases} 1 & \text{, if } s = (Left, j, x_{comb_k}) \text{ and } o = (Left,j), \\ 1 & \text{, if } s = (Right,j,x_{comb_k}) \text{ and } o = (Right,j), \\ 0 & \text{, otherwise,} \end{cases} \tag{3.17}$$

where $o \in \mathcal{O}$, $s \in \mathcal{S}$, $j \in \{1,2,\ldots,N_{gaps}\}$ and $k \in \{1,2,\ldots,N_{comb}\}$.

Example 3.2.5 The observation matrix from the example illustrated in Figure 3.4 is shown below where the POMDP models a merging situation with five gaps and one suspected ghost-car. In the observation matrix shown, it can be seen that state s_1 and s_6 both give rise to the same observation o_1. The reason is that from the definition of the state space, $s_1 = (Left,1,0_2)$ and $s_6 = (Left,1,1_2)$ indicating that the host car is in the left lane and adjacent to gap 1 which is what $o_1 = (Left,1)$ indicates.

$$\mathcal{Z} = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \\ s_9 \\ s_{10} \\ s_{11} \\ s_{12} \\ s_{13} \\ s_{14} \\ s_{15} \\ s_{16} \\ s_{17} \\ s_{18} \\ s_{19} \\ s_{20} \end{array} \begin{array}{c} \begin{array}{cccccccccc} o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 & o_9 & o_{10} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

3.2.1.7 Initial belief point b_0

The initial belief point is computed with means of the *probability vector of suspected ghost-cars*.

Definition 3.2.2 - Probability vector of suspected ghost-cars
The probability vector of suspected ghost-cars, denoted $prob_{real}$ contains N_{unsure} elements where $prob_{real}(m) \in (0,1)$ indicates the probability that $car_{?m}$ has status *real car*.

Example 3.2.6 Probability vector $prob_{real} = [0.25, 0.5]$ indicates that $car_{?1}$ has status *real car* with probability 0.25 and $car_{?2}$ has status *real car* with probability 0.5.

The initial belief point $b = b(s_1), b(s_2), \ldots, b(s_{|\mathcal{S}|})$ is computed as,

$$b_0(s) = \begin{cases} \prod_{m=1}^{N_{unsure}} probTot(m) & \text{, if } s = (Left,host_{pos},x_{comb_k}), \\ 0 & \text{, otherwise,} \end{cases} \tag{3.18}$$

s.t.

$$probTot(m) = \begin{cases} 1 - prob_{real}(m) & \text{, if } x_{comb_k}(m) = 0, \\ prob_{real}(m) & \text{, if } x_{comb_k}(m) = 1, \end{cases}$$

where $m \in \{1,\ldots,N_{unsure}\}$, $k \in \{1,2,\ldots,N_{comb}\}$, $s \in \mathcal{S}$ and $host_{pos} \in \{1,2,\ldots,N_{gaps}\}$ indicates which gap the host car is adjacent to.

3.2.1.8 Discount factor γ

The discount factor must be kept in the interval $0 < \gamma < 1$ and the influence of γ on the behavior of the host car is examined in Section 4.2.3.

3.2.2 Description of the MDP model

The MDP model used in Stage 1 is an infinite-horizon discounted reward MDP (see Section 2.2.2), and is described as follows.

State space, $\mathcal{S}_{\text{MDP}}$
The MDP state space $\mathcal{S}_{\text{MDP}}$ is a subset of the POMDP state space $\mathcal{S}$ and is given by,

$$\mathcal{S}_{\text{MDP}} = \left\{ s \in \mathcal{S} | s = (i,j,x_{comb_{N_{comb}}}),\ i \in \{Left, Right\},\ j \in \{1,2,\ldots,N_{gaps}\} \right\}. \tag{3.19}$$

Hence, the MDP state space only consists of those POMDP states representing all suspected ghost-cars having status *real car*. In total the MDP state space has $|\mathcal{S}_{\text{MDP}}| = 2N_{gaps}$ number of states and these are ordered as described in (3.1)-(3.3) but with $X_{comb} = x_{comb_{N_{comb}}}$.

Example 3.2.7 If the same POMDP example is considered as in Example 3.2.2, the resulting MDP state space is,

$$\mathcal{S}_{MDP} = \begin{cases} s_1 : (Left,1,11_2) & s_7 : (Right,1,11_2) \\ s_2 : (Left,2,11_2) & s_8 : (Right,2,11_2) \\ s_3 : (Left,3,11_2) & s_9 : (Right,3,11_2) \\ s_4 : (Left,4,11_2) & s_{10} : (Right,4,11_2) \\ s_5 : (Left,5,11_2) & s_{11} : (Right,5,11_2) \\ s_6 : (Left,6,11_2) & s_{12} : (Right,6,11_2). \end{cases}$$

Action space, $\mathcal{A}_{\text{MDP}}$
The MDP action space $\mathcal{A}_{\text{MDP}}$ equals the POMDP action space $\mathcal{A}$,

$$\mathcal{A}_{\text{MDP}} = \mathcal{A}. \tag{3.20}$$

Transition function, $\mathcal{T}_{\text{MDP}} : \mathcal{S}_{\text{MDP}} \times \mathcal{A}_{\text{MDP}} \times \mathcal{S}_{\text{MDP}} \rightarrow [0,1]$
The MDP transition function $\mathcal{T}_{\text{MDP}}$ equals the POMDP transition function $\mathcal{T}$ for states included in the MDP state space,

$$\mathcal{T}_{\text{MDP}}(s,a,s') = \mathcal{T}(s,a,s') \quad \text{, if } s,s' \in \mathcal{S}_{\text{MDP}}. \tag{3.21}$$

Reward function, $\mathcal{R}_{\text{MDP}} : \mathcal{S}_{\text{MDP}} \times \mathcal{A}_{\text{MDP}} \rightarrow \mathbb{R}$
The MDP reward function $\mathcal{R}_{\text{MDP}}$ equals the POMDP reward function $\mathcal{R}$ for states included in the MDP state space,

$$\mathcal{R}_{\text{MDP}}(s,a) = \mathcal{R}(s,a) \quad \text{, if } s \in \mathcal{S}_{\text{MDP}}. \tag{3.22}$$

Discount factor, γ_{MDP}
The discount factor in the MDP model equals the discount factor in the POMDP model,

$$\gamma_{\text{MDP}} = \gamma. \tag{3.23}$$

3.2.3 Solving the POMDP model and the MDP model

The solution of Stage 1 is computed in three steps illustrated in Figure 3.5 where in Step 1 the MDP model is derived, in Step 2 an ϵ-consistent solution of the MDP model is found and in Step 3 an approximate solution of the POMDP model is calculated. All steps are described next.

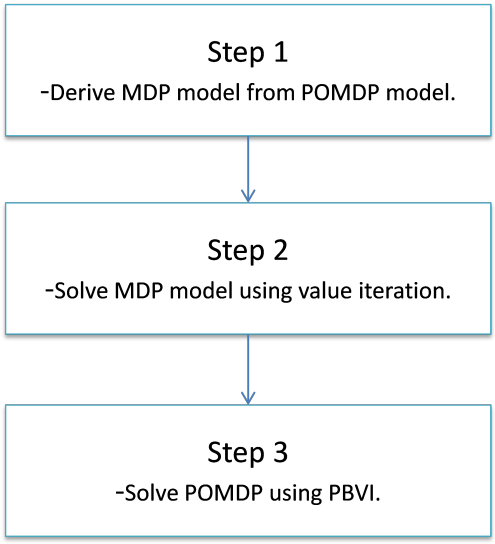


Figure 3.5: Illustration of the steps involved when solving the POMDP model and the MDP model.

Step 1: Derive MDP model from POMDP model
The first step derives the MDP model from the POMP model using (3.19)-(3.23). These equations are implemented in an algorithm called DERIVEMDP.

Step 2: Solve MDP using Value Iteration
In Step 2 the MDP model obtained from Step 1 is solved with means of value iteration using (2.2) and (2.3). A solution can be provided from this step if the maximum allowed execution time is reached before a solution has been found in the final step (Step 3). If all suspected ghost-cars have status *real car* or if the sensors are perfect so that there are no suspected ghost-cars, the solution from Stage 1 is ϵ-consistent with respect to the model. If there is still execution time left, the value function resulting from the value iteration V_{MDP} is passed to the next and final step.

Step 3: Solve POMDP using PBVI
In Step 3 the POMDP model is solved approximately with means of PBVI. The PBVI algorithm may run until the predetermined maximum execution time is reached and the host car needs to know which action to execute. The value function from Step 2, V_{MDP} is considered a good first approximation of the POMDP model and therefore the PBVI algorithm is initialized with this value function. The POMDP value function is initialized as $\Gamma_0 = \{\alpha_0\}$ where α_0 is the initial α-vector defined by,

$$\alpha_0(s) = \begin{cases} V_{\text{MDP}}(s_{\text{MDP}}) & \text{, if } s = s_{\text{MDP}} = (i,j,x_{comb_{N_{comb}}}), \\ 0 & \text{, otherwise,} \end{cases} \tag{3.24}$$

where $i \in \{Left, Right\}$, $j \in \{1,2,\ldots,N_{gaps}\}$, $s \in \mathcal{S}$ and $s_{\text{MDP}} \in \mathcal{S}_{\text{MDP}}$.

Algorithm 3.2.1 describes the PBVI algorithm used to solve the POMDP. The PBVI algorithm is based on the PBVI equations: (2.14), (2.15), (2.18), (2.19) and (2.20), but also utilizes the fact that the observation matrix is deterministic. The algorithm is given the set of α-vectors from the previous iteration step Γ_{n-1}, the transition matrix $\mathcal{T}$, the reward matrix $\mathcal{R}$, the selected belief points $\mathcal{B}$ to be used in the iteration, the discount factor γ, the number of actions $|\mathcal{A}|$, the number of observations $|\mathcal{O}|$ and the number of states $|\mathcal{S}|$. Also the algorithm is given a matrix $\mathcal{Z}_{indexes}$ where each row in $\mathcal{Z}_{indexes}$ indicates the non-zero elements in each column in the observation matrix $\mathcal{Z}$. The matrix $\mathcal{Z}_{indexes}$ is used in order to allow for faster computation of $\Gamma_n^{a,o}$. In Algorithm 3.2.1 $\mathbb{e}_i$ refers to the unit vector in the i^{th} dimension and S^a is an $|\mathcal{S}| \times |\mathcal{B}|$ matrix where each column equals the second term in (2.18), that is,

$$\text{Column } b \text{ in } S^a = \sum_{o \in \mathcal{O}} \arg\max_{\alpha \in \Gamma_n^{a,o}} \Big(\sum_{s \in \mathcal{S}} \alpha(s) b(s) \Big) \quad b \in \mathcal{B}, a \in \mathcal{A}, \tag{3.25}$$

The algorithms described in this thesis use some Matlab syntax so for example in Algorithm 3.2.1 the notation $\mathcal{T}(:,a,I_o)$ refers to all elements in the first dimension, element a in the second dimension and elements I_o in the third dimension. Also note that $(I_{b\alpha}^{max})_i$ refers to the i^{th} element in $I_{b\alpha}^{max}$ and that the sets $\Gamma_n^{a,o}$, $\Gamma_n^{a,*}$, Γ_n^a are treated as matrixes.

Algorithm 3.2.1: PointBasedValueIteration $(\Gamma_{n-1}, \mathcal{T}, \mathcal{R}, \mathcal{B}, \gamma, |\mathcal{A}|, |\mathcal{O}|, |\mathcal{S}|, \mathcal{Z}_{indexes})$

for $a = 1, \ldots, |\mathcal{A}|$ **do**
  Initialize sum over observations, $S^a = \mathbf{0}^{|\mathcal{S}| \times |\mathcal{B}|}$
  for $o = 1, \ldots, |\mathcal{O}|$ **do**
    Obtain row o from $\mathcal{Z}_{indexes}$, $I_o = \mathcal{Z}_{indexes}(o, :)$
    Calculate $\Gamma_n^{a,o} = \gamma \cdot \mathcal{T}(:, a, I_o) \cdot \Gamma_{n-1}(I_o, :)$
    Obtain sum over all $b \in \mathcal{B}$ and all $\alpha^{a,o} \in \Gamma_n^{a,o}$, $S_{b\alpha} = \mathcal{B} \cdot \Gamma_n^{a,o}$
    Calculate $I_{b\alpha}^{max}$, the indexes for the max values in each row in $S_{b\alpha}$, $(I_{b\alpha}^{max})_i = \text{argmax}(S_{b\alpha} e_i)$
    Calculate $S^a = S^a + \Gamma_n^{a,o}(:, I_{b\alpha}^{max})$
  end for
  Set $\Gamma_n^{a,*} = \mathcal{R}(:, a)$
  Set $\Gamma_n^a = \Gamma_n^{a,*} + S^a$
end for
for $a = 1, \ldots, |\mathcal{A}|$ **do**
  Obtain sum over all $b \in \mathcal{B}$ and all $\alpha_b^a \in \Gamma_n^a$, $S_{b\Gamma} = \mathcal{B} \cdot \Gamma_n^a$
end for
Calculate $I_{b\Gamma}^{max}$, the indexes for the max value in each row in $S_{b\Gamma}$, $(I_{b\Gamma}^{max})_i = \text{argmax}(S_{b\Gamma} e_i)$
Remove duplicates in $I_{b\Gamma}^{max}$
for each j **in** $I_{b\Gamma}^{max}$ **do**
  Add new α-vector, $\Gamma_n = \Gamma_n \cup \{\Gamma_n^a(j)\}$
end for
return Γ_n

The belief point selection strategy used with the PBVI algorithm does not implement any of the strategies described in Section 2.3.3.1. Instead the strategy used is specifically designed for the Stage 1 POMDP model. The strategy is based on the fact that Stage 1 will transition to Stage 2 when the host car wishes to execute action a_1. Therefore this action is only considered interesting when it is used for taking the host car from the left lane to the right lane, not the other way around. The belief points resulting from the host car returning to the left lane are therefore excluded.

If the host car cannot receive any non-zero reward for changing lane, that is $\sum_s \mathcal{R}(s,a_1) = 0$, then states corresponding to the same observation have exactly the same transition probabilities (see (3.13)-(3.16)). Since the observation matrix is deterministic this means that if $\sum_s \mathcal{R}(s,a_1) = 0$ the number of belief points that can ever be reached is no more than the number of observations. This can be seen from inspecting (2.9). When the host car cannot receive any non-zero reward for changing lane, all belief points that can ever be reached are selected.

If the host car can receive non-zero rewards for changing lane, Algorithm 3.2.2 is used. This algorithm is initially given those belief points that are reachable when executing the determin-

istic actions (a_2, a_3 and a_4) from the initial belief point. All of these actions cause the host car to stay in the left lane. Within Algorithm 3.2.2, Algorithm 3.2.3 is called which finds the one step forward reachable belief points that correspond to the host car still being in the left lane (denoted $\mathcal{B}'_{Left}$) and the one step forward reachable belief points corresponding to the host car being in the right lane (denoted $\mathcal{B}'_{Right}$). Algorithm 3.2.3 is only given the set $\mathcal{B}_{Left}$ as input since these are the belief points still of interest. In Algorithm 3.2.3 expressions (2.7) - (2.9) are used, where, the algorithm calculates $b_a(s)$, $b_a(o)$ and $b_a^o(s)$ $\forall s \in \mathcal{S}$, $\forall b \in \mathcal{B}_{Left}$ and these sets are denoted B_{a_s}, B_{a_o} and B_a^o respectively. Algorithm 3.2.3 returns the sets $\mathcal{B}'_{Right}$ and $\mathcal{B}'_{Left}$ which are used in Algorithm 3.2.2 to update the old sets $\mathcal{B}_{Right}$ and $\mathcal{B}_{Left}$. In Algorithm 3.2.2 a parameter called *resolution* is also used to remove belief points in $\mathcal{B}'_{Left}$ and $\mathcal{B}'_{Right}$ that are closely spaced in the belief space. Using $\mathcal{B}'_{Right}$ as an example this operation is mathematically expressed as follows:

Let $\hat{\mathcal{B}} = \emptyset$ and $f(b) = \text{nint}(resolution \cdot b)$ where $\text{nint}(x)$ is the nearest integer function applied to each element in x, and define the set operation $f(\hat{\mathcal{B}}) = \{f(b)|b \in \hat{\mathcal{B}}\}$. The set $\hat{\mathcal{B}}$ is then calculated from the following statement,

$$\forall b \in \mathcal{B}'_{Right} \quad \Big(f(b) \notin f(\hat{\mathcal{B}}) \rightarrow \hat{\mathcal{B}} = \hat{\mathcal{B}} \cup \{b\}\Big), \tag{3.26}$$

hence $\hat{\mathcal{B}} \subseteq \mathcal{B}'_{Right}$ where $\hat{\mathcal{B}}$ is used to update $\mathcal{B}'_{Right}$. The default value of *resolution* is $resolution = 1000$.

Algorithm 3.2.2 is called as long as the number of belief points in $\mathcal{B} = \mathcal{B}'_{Right} \cup \mathcal{B}'_{Left}$ does not exceed Max_b. The final set of selected belief points is the set $\mathcal{B}$.

Algorithm 3.2.2: SELECTBELIEFPOINTS ($\mathcal{B}_{Right}, \mathcal{B}_{Left}, |\mathcal{A}|, |\mathcal{O}|, |\mathcal{S}|, \mathcal{T}, \mathcal{Z}, resolution$)

$[\mathcal{B}'_{Right}, \mathcal{B}'_{Left}] =$ GETREACHABLEBELIEFPOINTS($\mathcal{B}_{Left}, |\mathcal{A}|, |\mathcal{O}|, |\mathcal{S}|, \mathcal{T}, \mathcal{Z}$)
$\mathcal{B}'_{Right} = \mathcal{B}_{Right} \bigcup B'_{Right}$
Remove zero rows and duplicate rows in $\mathcal{B}'_{Right}$
Use *resolution* to remove belief points in $\mathcal{B}'_{Right}$ that are very close in the belief space
$\mathcal{B}'_{Left} = \mathcal{B}_{Left} \bigcup B'_{Left}$
Remove zero rows and duplicate rows in $\mathcal{B}'_{Left}$
Use *resolution* to remove belief points in $\mathcal{B}'_{Left}$ that are very close in the belief space
return $\mathcal{B}'_{Right}$,$\mathcal{B}'_{Left}$

Algorithm 3.2.3: GetReachableBeliefPoints ($\mathcal{B}_{Left}$,$|\mathcal{A}|$, $|\mathcal{O}|$, $|\mathcal{S}|$, $\mathcal{T}$, $\mathcal{Z}$)

for $a = 1, \ldots, |\mathcal{A}|$ **do**
  Calculate $B_{a_s} = \mathcal{B}_{Left} \cdot \mathcal{T}(:, a, :)$
  Calculate $B_{a_o} = B_{a_s} \cdot \mathcal{O}$
  Only loop over left lane observations
  for $o = 1, \ldots, |\mathcal{O}|/2$ **do**
    Find indexes of non-zero elements in B_{a_o}, denote $I_{B_{a_o}}$
    Find indexes of zero elements in B_{a_o}, denote $I^0_{B_{a_o}}$
    if $I^0_{B_{a_o}} \neq \emptyset$
      Action a and observation o is not a possible combination for belief points with index $I_{B_{a_o}}$
      Set $B^o_a = 0$ for these belief points
    end if
    if $I_{B_{a_o}} \neq \emptyset$
      Action a and observation o is a possible combination for belief points with index $I_{B_{a_o}}$
      Calculate B^o_a for these belief points using (2.9)
    end if
  end for
end for
Set $\mathcal{B}'_{Left} = B^o_a$
Set $a = 1$
Calculate $B_{a_s} = \mathcal{B}_{Left} \cdot \mathcal{T}(:, a, :)$
Calculate $B_{a_o} = B_{a_s} \cdot \mathcal{O}$
Only loop over right lane observations
for $o = |\mathcal{O}|/2 + 1, \ldots, |\mathcal{O}|$ **do**
  Find indexes of non-zero elements in B_{a_o}, denote $I_{B_{a_o}}$
  Find indexes of zero elements in B_{a_o}, denote $I^0_{B_{a_o}}$
  if $I^0_{B_{a_o}} \neq \emptyset$
    Action a and observation o is not a possible combination for belief points with index $I_{B_{a_o}}$
    Set $B^o_a = 0$ for these belief points
  end if
  if $I_{B_{a_o}} \neq \emptyset$
    Action a and observation o is a possible combination for belief points with index $I_{B_{a_o}}$
    Calculate B^o_a for these belief points using (2.9)
  end if
end for
Set $\mathcal{B}'_{right} = B^o_a$
return $\mathcal{B}'_{Right}$,$\mathcal{B}'_{Left}$

3.2.4 Top Level Implementation of Stage 1

Stage 1 is described in Algorithm 3.2.4. The tuning parameters (given as inputs) to the algorithm are listed in Table 3.2 and Table 3.4 and the input variables are listed in Table 3.3. Algorithm 3.2.4 returns the action to be executed in Stage 1, denoted a_{Stage_1}.

Table 3.3: Input variables to Stage 1.

$vObjRightRelHost$	The velocities of the detected objects in the right lane relative to the host car.
$lObjRightRelHost$	The longitudinal positions of the detected objects in the right lane relative to the host car.
$vVehicleLeftRelHost$	The velocity of the vehicle in front of the host car in the left lane relative to the host car.
$lVehicleLeftRelHost$	The longitudinal position of the vehicle in front of the host car in the left lane relative to the host car.
$lEndRelHost$	The distance to the end-point from the host car.
v_{host}	The velocity of the host car.
v_{max}	Maximum allowed velocity of the host car.
v_{min}	Minimum allowed velocity of the host car.
$certainty$	The certainty vector, see Definition 3.2.1.
$prob_{real}$	The probability vector of suspected ghost-cars, see Definition 3.2.2.

Within Algorithm 3.2.4 another algorithm, Algorithm 3.2.5, is called and this algorithm is given all input variables listed in Table 3.5 and Table 3.1 except $sizeGap(j,x_{comb_k})$ and $distMidEnd(j, x_{comb_k})$. Algorithm 3.2.5 is also given the input tuning parameters listed in Table 3.2 as well as the tuning parameters G_U and $CarSize$ which are used in (3.11) and (3.5) respectively.

Algorithm 3.2.5 computes all parts required in the POMDP model and returns the transition function, the reward function, the observation function and the initial belief point. Also a Boolean called *solve* is returned which if $False$ indicates that the POMDP model should not be solved and then the action to be executed is set to $a_{Stage_1} = -1$. This can for example happen if $vehicle_{Front}$ drives very slowly so that the host car cannot stay adjacent to any gap between two subsequent objects detected in the right lane. Some extra caution needs to be taken if there are no suspected ghost-cars, that is $N_{unsure} = 0$. In that case a POMDP model is still created and an MDP model is derived, however, only the MDP model is solved. If there is at least one suspected ghost-car ($N_{unsure} > 0$) the PBVI algorithm is used to find an approximate solution of the POMDP. The PBVI algorithm runs Max_{iter} times or until the number of α-vectors exceeds Max_α. If Max_α is reached it is assumed that a PBVI update takes to long to compute resulting in a non-real time decision. The action to be executed is calculated by means of calling MostPromisingAction which is not described further in this report since it

Table 3.4: Remaining tuning parameters (given as inputs) to Stage 1. In Table 3.2 the other tuning parameters used in Stage 1 are shown.

Max_b	The selection of belief points terminates when the number of selected belief points exceeds Max_b. Used in Algorithm 3.2.2.
$resolution$	Tolerance used to remove reachable belief points that are closely spaced, see Algorithm 3.2.2.
Max_α	Maximum number of α-vectors allowed in the PBVI algorithm.
Max_{iter}	Maximum number of PBVI updates allowed.
G_U	Weight used when computing the POMDP transition function, used in (3.11).
$CarSize$	The average length of a car.
γ	Discount factor in the POMDP model and the MDP model.

Table 3.5: Remaining input variables to CreatePOMDP (Algorithm 3.2.5). In Table 3.1 the other input variables to CreatePOMDP are shown.

$host_{pos}$	The number of the gap that the host car is adjacent to.
$certainty$	The certainty vector, see Definition 3.2.1.
$prob_{real}$	The probability vector of suspected ghost-cars, see Definition 3.2.2.
N_{comb}	The number of possible values of X_{comb}^{vec}, see Section 3.2.1.1.
N_{unsure}	The number of suspected ghost-cars.
N_{gaps}	Number of gaps.
$\lvert\mathcal{S}\rvert$	Number of states in the POMDP model
$gapsize(j)$	The size of gap j if all suspected ghost-cars have status *real car*, see (3.5).
$distmidend(j)$	The distance from the midpoint of gap j to the end-point if all suspected ghost-cars have status *real car*, see (3.6).
$vMidRel$	The relative velocity between the midpoints of gap j and gap $j+1$, see (3.11).

is merely an implementation of (2.10).

Algorithm 3.2.4: STAGE1 (args)

Calculate the input variables to CREATEPOMDP, (ALGORITHM 3.2.5).
if $N_{unsure} = 0$
    Temporarily change the number of suspected ghost-cars: $certainty(end-1) = 0, N_{unsure} = 1, |\mathcal{S}| = 2 \cdot N_{gaps} 2^1$
    $[\mathcal{T}, \mathcal{R}, \mathcal{Z}, b_0, solve] =$ CREATEPOMDP$(args)$
    Restore the number of suspected ghost-cars: $certainty(end-1) = 1, N_{unsure} = 0, |\mathcal{S}| = 2 \cdot N_{gaps} 2^0$
else
    $[\mathcal{T}, \mathcal{R}, \mathcal{Z}, b_0, solve] =$ CREATEPOMDP$(args)$
end if
if $solve = True$
    if $N_{unsure} = 0$
        Temporarily change N_{comb}, $N_{comb} = 2$
        $[\mathcal{T}_{\text{MDP}}, \mathcal{R}_{\text{MDP}}] =$ DERIVEMDP$(\mathcal{T}, \mathcal{R}, N_{gaps}, N_{comb}, |\mathcal{A}|)$
        Restore N_{comb}, $N_{comb} = 1$
    else
        $[\mathcal{T}_{\text{MDP}}, \mathcal{R}_{\text{MDP}}] =$ DERIVEMDP$(\mathcal{T}, \mathcal{R}, N_{gaps}, N_{comb}, |\mathcal{A}|)$
    end if
    Set current MDP state to the gap that the host car is adjacent to, $state_{IN} = host_{pos}$
    Solve MDP with value iteration. Obtain value function V_{MDP} and action a_{MDP} to execute in $state_{IN}$
    Set POMDP action to MDP action, $a_{Stage_1} = a_{\text{MDP}}$
    if $N_{unsure} > 0$
        Initialize PBVI algorithm using V_{MDP}
        Find non-zero indexes in each column in $\mathcal{Z}$, store in $\mathcal{Z}_{indexes}$
        Select belief points $\mathcal{B}$ using SELECTBELIEFPOINTS$(\mathcal{B}_{Right}, \mathcal{B}_{Left}, |\mathcal{A}|, |\mathcal{O}|, |\mathcal{S}|, \mathcal{T}, \mathcal{Z}, resolution)$
        for $n = 1,2,\ldots,Max_{iter}$ **do**
            $\Gamma_n =$ POINTBASEDVALUEITERATION$(\Gamma_{n-1}, \mathcal{T}, \mathcal{R}, \mathcal{B}, \gamma, |\mathcal{A}|, |\mathcal{O}|, |\mathcal{S}|, \mathcal{Z}_{indexes})$
            if number of α-vectors in $\Gamma_n > Max_\alpha$
                break
            end if
        end for
        $a_{Stage_1} =$ MOSTPROMISINGACTION$(b_0, \mathcal{R}, \mathcal{T}, \mathcal{Z}, |\mathcal{S}|, |\mathcal{A}|, |\mathcal{O}|, \Gamma_n)$
    end if
else
    No gap is suitable to stay by, set $a_{Stage_1} = -1$
end if
return a_{Stage_1}

Algorithm 3.2.5: CREATEPOMDP($args$)

Calculate the initial belief point b_0 using (3.18).
Create transition matrix $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ using (3.13)-(3.16).
Create observation matrix $\mathcal{Z} : \mathcal{S} \times \mathcal{O}$ using (3.17).
Create reward matrix $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ using (3.7)-(3.10).
if Reward matrix $\mathcal{R}$ gives no rewards for action a_2
    $solve = False$
end if
Convert $\mathcal{R}$ from $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ to $\mathcal{S} \times \mathcal{A}$ using (2.1).
return $\mathcal{T}, \mathcal{R}, \mathcal{Z}, b_0, solve$

3.3 Stage 2

In Stage 2 the objective for the host car is to show its intention to merge by signaling and moving laterally towards the line that separates the left lane from the right lane. The host car should give signal for some time, allowing surrounding vehicles to be aware of its intention to merge and then decide whether to continue to Stage 3 or return to Stage 1.

In Stage 2 Algorithm 3.3.1 is called where several conditions are checked in order to decide which action to execute. The action to be executed in Stage 2 is denoted a_{Stage_2} where,

- $a_{Stage_2} = 1$ indicates that the host car shall return to Stage 1,
- $a_{Stage_2} = 2$ indicates that the host car shall continue to show its intention,
- $a_{Stage_2} = 3$ indicates that the host car shall continue to the next stage, Stage 3.

Table 3.6 describes the input variables used in Algorithm 3.3.1 and Table 3.7 describes the tuning parameters used.

Table 3.6: Input variables to Stage 2.

$host_{pos}$	The number of the gap that the host car is adjacent to.
$gapsize(j)$	The size of gap j if all suspected ghost-cars have status *real car*.
t_{int}	Indicates for how many time samples the host car has shown its intention.
bad_{gap}	The number of the gap that the host car failed to merge into. This variable is further explained in Section 3.5.

Table 3.7: Tuning parameters (given as inputs) to Stage 2.

$GapSafety$	Minimum gap size required for allowing host car to start merging into a gap.
T_{int}	Indicates for how many time samples the host car has to show its intention.

Algorithm 3.3.1: STAGE2 $(host_{pos}, gapsize, GapSafety, t_{int}, T_{int}, bad_{gap})$

Set goal gap, $goal_{gap} = host_{pos}$
Find gap size of goal gap, $sizeGap_{goal} = gapsize(goal_{gap})$
if $t_{int} = 0$
    Store size of gap at start, $gap_{mem}(1) = sizeGap_{goal}$
else if $t_{int} = T_{int}$
    Store size of gap at end time, $gap_{mem}(end) = sizeGap_{goal}$
    Find out if gap has become larger, $prog_{gap} = gap_{mem}(1) - gap_{mem}(end)$
end if
if $t_{int} < T_{int}^{lim}$
    Keep showing intention, $a_{Stage_2} = 2$
    Increase intention timer $t_{int} = t_{int} + 1$
else if $t_{int} \geq T_{int}$ **and** $((sizeGap_{goal} \geq GapSafety)$ **or** $(prog_{gap} > 0$ **and** $sizeGap_{goal} \geq GapSafety \cdot 0.8))$
    Go to Stage 3, $a_{Stage_2} = 3$
    Reset intention timer $t_{int} = 0$
else
    Go back to Stage 1, $a_{Stage_2} = 1$
    Reset intention timer $t_{int} = 0$
    Update bad gap $bad_{gap} = goal_{gap}$
end if
return $a_{Stage_2}, t_{int}, bad_{gap}$

3.4 Stage 3

In Stage 3 the objective for the host car is to perform the actual lane changing maneuver. In this stage Algorithm 3.4.1 is called which checks a condition in order to decide whether to continue to merge or cancel the merging operation. The action to be executed in Stage 3 is denoted a_{Stage_3} where,

- $a_{Stage_3} = 1$ indicates that the host car should continue to merge,

- $a_{Stage_3} = 2$ indicates that the host car should cancel the merging operation and return to Stage 1.

In Table 3.6 and Table 3.7 all input variables and tuning parameters used in Algorithm 3.4.1 are included.

Algorithm 3.4.1: STAGE3 $(host_{pos}, gapsize, GapSafety)$

Set goal gap, $goal_{gap} = host_{pos}$
Find gap size of goal gap, $sizeGap_{goal} = sizeGaps(goal_{gap})$
if $sizeGap_{goal} \geq GapSafety$
    Keep performing merging, $a_{Stage_3} = 1$
else
    Cancel merging, $a_{Stage_3} = 2$
    Update bad gap, $bad_{gap} = goal_{gap}$
end if
return a_{Stage_3}, bad_{gap}

3.5 Top Level Implementation of the Merging Decision Algorithm

Algorithm 3.5.1 describes the top level implementation of the merging decision algorithm. The tuning parameters (given as inputs) to the algorithm are shown in Table 3.2, Table 3.4, Table 3.7 and Table 3.8 and the input variables are described in Table 3.3. The merging decision algorithm is also given a variable called *Stage* which indicates which stage the system is in. The tuning parameters shown in Table 3.8, that is $T_{Bad_{gap}}$ and $T^{Host,End}$, will next be explained in more detail.

Table 3.8: Remaining tuning parameters (given as inputs) to the merging decision algorithm. Table 3.2, Table 3.4, Table 3.7 and Table 3.8 show the other tuning parameters used.

$T_{Bad_{gap}}$	Number of time samples before bad_{gap} is reset.
$T^{Host,End}$	Minimum time required in order to set $end_{close} = True$.

The tuning parameter denoted $T_{Bad_{gap}}$ is associated with the variable bad_{gap}. If the host car tries to merge into a gap but the gap turns out to be unfavorable to merge into then bad_{gap} takes on the number of the unfavorable gap. The host car is prevented from trying to merge into the same unfavorable gap while $bad_{gap} \neq 0$ and bad_{gap} is set to zero after some time $T_{Bad_{gap}}$. If $bad_{gap} \neq 0$ then $sizeGap(bad_{gap},x_{comb_k})$, $distMidEnd(bad_{gap},x_{comb_k})$ and $distMidHost(bad_{gap})$

are adjusted so that the host car will not find that gap favorable.

The tuning parameter denoted $T^{Host,End}$ is associated with a Boolean called end_{close} which indicates whether the host car is close to the end-point or not. This Boolean is set to $True$ if the estimated time for the host car to reach the end-point is less than $T^{Host,End}$. The Boolean end_{close} is returned by Algorithm 3.5.1 and can be used by the master function to increase aggressiveness of the host car.

Algorithm 3.5.1: MERGINGDECISIONALGORITHM (args)

Set stage actions to zero, $a_{\text{Stage}_i} = 0$, $i \in \{1,2,3\}$
If there are no or one object detected in the right lane, set $Stage = -1, a_{\text{Stage}_1} = -1$
Estimate average time for host to end-point, denote $t^{Host,End}$
if $t^{Host,End} < T^{Host,End}$ **then** $end_{close} = True$ **end if**
if $t_{Bad_{gap}} > T_{Bad_{gap}}$ **then** $bad_{gap} = 0$ **end if**
Increase bad_{gap} timer, $t_{Bad_{gap}} = t_{Bad_{gap}} + 1$
if $Stage = 1$
    Calculate all input variables to be used in STAGE1
    $a_{Stage_1} = \text{STAGE1(args)}$
else if $Stage = 2$
    Clear bad gap timer, $t_{Bad_{gap}} = 0$
    Calculate all input variables to be used in STAGE2
    $[a_{Stage_2}, t_{int}, bad_{gap}] = \text{STAGE2}(host_{pos}, sizegap, t_{int}, T_{int}, GapSafety, t_{Bad_{gap}})$
else if $Stage = 3$
    Calculate all input variables to be used in STAGE3
    $[a_{Stage_3}, bad_{gap}] = \text{STAGE3}(host_{pos}, sizegap, GapSafety)$
end if
return $a_{Stage_1}, a_{Stage_2}, a_{Stage_3}, end_{close}$

The master function calls Algorithm 3.5.1 frequently and determines when the merging operation has been completed. If it has been completed the system transitions to Stage 4 where the host car is controlled (by the master function) so that it follows the car in front of it in the right lane.

4

Results

This chapter presents results obtained when testing the merging decision algorithm (Algorithm 3.5.1). Section 4.1 describes how the algorithm is implemented in a simulation environment and Section 4.2 presents four test cases where the main focus is on showing different properties of the algorithm and evaluating the quality of the decisions made by the algorithm. Section 4.3 evaluates the performance of policies obtained when using the implemented POMDP solver and lastly Section 4.4 evaluates the execution time of the algorithm. The results from this chapter are discussed in more detail in Section 5.1.

4.1 Implementation of the Merging Decision Algorithm

All algorithms described in Chapter 3 are implemented in embedded Matlab in order to make it easier to use the merging decision algorithm in a real test car. Also a simulation environment is developed so that the merging decision algorithm can be tested for different traffic scenarios also without using a real test car. Figure 4.1 shows a snap shot of the simulation window. The host car is in the left lane and there are six objects detected in the right lane where the third object and the fifth object are suspected ghost-cars. Also there is a car in front of the host car in the left lane. The end-point cannot be seen.

4.2 Test Cases

This section presents results obtained from using the merging decision algorithm in the simulation environment. Test case 1 and Test case 2 show how the behavior of the host car differs between decisions based on the POMDP model and decisions based on the MDP model. Test case 3 shows how the discount factor γ can be used in order to adjust the behavior of the host car. In Test case 4 the algorithm is tested with data from a real traffic scenario. In all test cases the desirable behavior of the host car is discussed through rational reasoning.

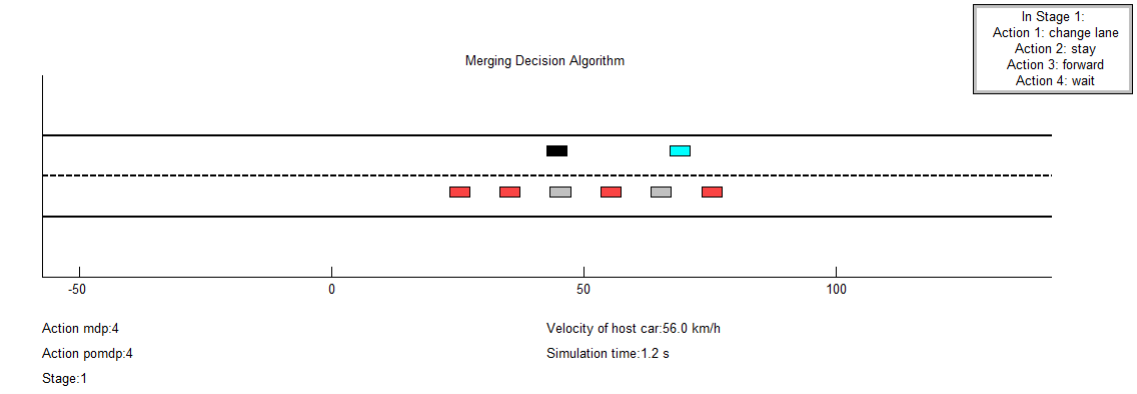


Figure 4.1: Snap shot of the simulation window. There are six objects detected in the right lane and the ones colored in gray are suspected ghost-cars.

In the first three test cases six cars are simulated in the right lane and in all test cases there is either zero, one or two suspected ghost-cars where the status for the suspected ghost-cars are *real car*. Test case 1, 2 and 3 include at least two sub-tests and for a given test case the behavior of the cars in the right lane is the same for all sub-tests. Also in all test cases the distance to the end-point is set to a high value. The values of the tuning parameters used when performing the tests can be found in Appendix A. Note that the values of Max_b, Max_α, Max_{iter} and $resolution$ are only important for sub tests where there is at least one suspected ghost-car, that is $N_{unsure} > 0$ (see Algorithm 3.2.4). In Appendix A it is also described how the maximum allowed speed for the host car v_{max} and the minimum allowed speed for the host car v_{min} are set in the simulation environment.

In the first three test cases the simulation time in Simulink is set to 100 s and a decision is made every 0.2 s. The real execution times are presented in Table 4.1 where it can be seen that the execution times vary significantly between sub-tests. This is explained by the fact that the number of suspected ghost-cars vary between tests and the amount of time that the host car is in Stage 1, which is the most computationally complex stage, also varies between the tests.

4.2.1 Test Case 1

In all sub-tests in Test case 1 the host car starts by gap 4 (in Definition 3.1.3 it is described how the gaps are numbered) and must not drive above 75 km/h. In this test case the cars in the right lane are equally spaced and are driving at 70 km/h. No gap is wide enough to try to merge into, hence B_{LC} will always be $False$.

Test case 1.1

The certainty vector in Test case 1.1 equals $certainty = [1\ 1\ 1\ 1\ 1\ 1]$, hence there are no suspected ghost-cars. This means that the action given from Stage 1 will always equal the action provided from solving the MDP model, hence $a_{Stage_1} = a_{\text{MDP}}$ (see Algorithm 3.2.4).

Table 4.1: Execution times required for Test case 1, 2 and 3.

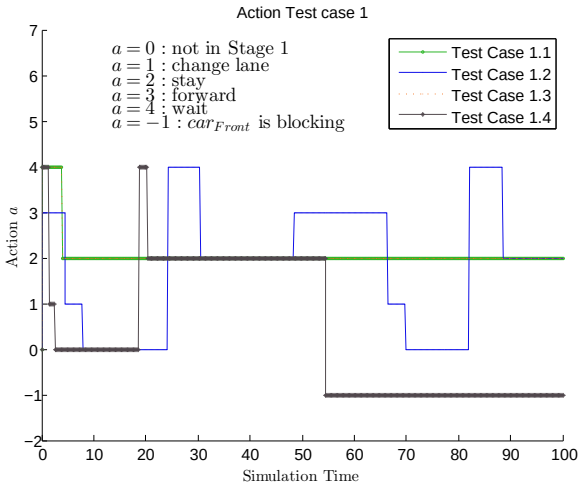
Test Case	**Execution Time [s]**
Test Case 1.1	13.4
Test Case 1.2	189.6
Test Case 1.3	233.3
Test Case 1.4	55.0
Test Case 2.1	10.6
Test Case 2.2	43.3
Test Case 3.1	12.7
Test Case 3.2	9.5

Desirable behavior of host car
Since all cars are equally spaced and are driving at the same speed it is reasonable to assume that the likelihood that one of the gaps becomes wider the next time instance is the same for all gaps. In this situation it would therefore be reasonable to move so that the sensor system may detect other gaps that are larger. If the host car increases its speed it will get even closer to the end-point and therefore a better alternative is to slow down. Since gap 1 should always be considered an unfavorable alternative, a good decision would be to stay by gap 2.

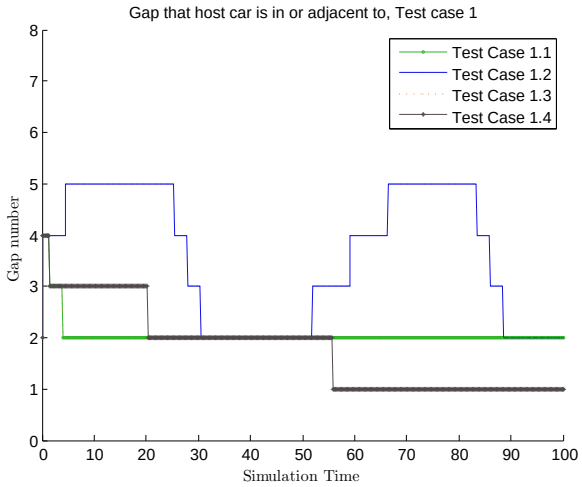
Behavior of host car
The host car waits for gap 2 and stays there throughout the rest of the simulation. The results from Test case 1.1 are shown with green lines with connected circles in Figure 4.2a and Figure 4.2b where Figure 4.2a shows the actions executed at each simulation time step and Figure 4.2b shows which gap the host car is adjacent to at every time step.

Test case 1.2
In test case 1.2 the certainty vector is set to $certainty = [1\ 1\ 1\ 1\ 0\ 1]$ and the probability vector for suspected ghost-cars equals $prob_{real} = [0.5]$, thus $object_5$ is a suspected ghost-car and has status *real car* with probability 0.5 and the other objects are classified as cars.

Desirable behavior of host car
Since there is a 50% chance that $object_5$ has status *ghost-car* this means that there is a 50% chance that gap 5 and gap 6 correspond to the same space which is then large enough to try to merge into. A good decision would therefore be to try to merge into that space. However, a 50% chance that gap 5 and gap 6 correspond to the same space may also be considered too small. Then a good decision could be to try to find larger gaps that are further away and therefore waiting for gap 2 could be a good alternative (using the reasoning made in Test case 1.1). If the host car tries to merge into gap 5 or gap 6 but fails to complete the maneuver, a good decision would be to wait for gap 2. In this case the same reasoning as in Test case 1.1 is used.



(a) Actions executed versus time. A decision, followed by an action, is made every 0.2 s.



(b) Number of the gap that the host car is adjacent to versus time.

Figure 4.2: Results from the sub-tests in Test case 1. In Test case 1.1 no cars are classified as suspected ghost-cars and in this test case the host car waits (action a_4) for gap 2 and then stays (action a_2) adjacent to this gap throughout the rest of the simulation (green line with connected circles). In Test case 1.2 $object_5$ is classified as a suspected ghost-car and has status *real car* with probability 0.5. In this test case the host car tries to merge into gap 5 (blue solid line). In Test case 1.3 $object_5$ has status *real car* with probability 0.9 and the host car behaves as in Test case 1.1. There is only a small chance that $object_5$ is a ghost-car and therefore gap 5 is not as attractive as in Test case 1.2. In Test case 1.4 there are two suspected ghost-cars as well as a vehicle in front of the host car in the left lane. At time step $t \gtrapprox 54$ this vehicle prevents the host car from staying adjacent to gap 2 which is why the host car is forced to stay by gap 1 (dark gray line with connected stars).

Behavior of host car

The host car first drives forward to gap 5 where it shows its intention to merge and thus the system transitions to Stage 2, which can be seen from the blue solid line in Figure 4.2a at time steps $8 \lessapprox t \lessapprox 24$ where the action equals zero. In the graph, $a = 0$ only indicates that the host car is not in Stage 1, therefore when $a = 0$ the host car can either be in Stage 2, Stage 3 or Stage 4 (see Algorithm 3.5.1). It turns out that gap 5 is not a favorable gap to merge into (since the status of the suspected ghost-car is *real car*) and the host car then waits for gap 2. The host car stays by gap 2 for some time, see the blue solid line in Figure 4.2b for time steps $32 \lessapprox t \lessapprox 52$, but then it once again returns to gap 5. This is explained by the fact that the bad_{gap} variable has been reset. Before the simulation stops, the host car tries to merge into gap 5 once again, seen in Figure 4.2a for $74 \lessapprox t < 100$.

Test case 1.3

In test case 1.3 the certainty vector is, in similarity with Test case 1.2, set to $certainty = [1\ 1\ 1\ 1\ 0\ 1]$, however the probability vector for suspected ghost-cars equals $prob_{real} = [0.9]$, hence the probability that $object_5$ has status *real car* is now 0.9 instead of 0.5.

Desirable behavior of host car
In this situation the same reasoning can be made as in Test case 1.2. However, in this test case the chance that gap 5 and gap 6 refer to the same space is decreased to just 10% so it is more reasonable to stay by gap 2 compared to Test case 1.2.

Behavior of host car
The host car behaves exactly as in Test case 1.1. The resulting graphs from Test case 1.3 can be seen from the orange dashed line in Figure 4.2a and Figure 4.2b. This line coincides with the line representing the results from Test case 1.1.

Test case 1.4
In Test case 1.4 $object_3$ and $object_5$ are classified as suspected ghost-cars, thus the certainty vector is set to $certainty = [1\ 1\ 0\ 1\ 0\ 1]$. The probability vector for suspected ghost-cars is set to $prob_{real} = [0.5, 0.8]$, hence $object_3$ has status *real car* with 50% confidence and $object_5$ has status *real car* with 80% confidence. A vehicle is also driving in front of the host car in the left lane. This vehicle is initially driving faster than the cars in the right lane but eventually slows down.

Desirable behavior of host car
If there would not be a vehicle in front of the host car in the left lane a good decision could be to try to merge into gap 3 or gap 4 which are the gaps closest to $object_3$. There is a 50% chance that these gaps correspond to the same space. If the host car tries to merge into one of these gaps but fails to complete the maneuver a good decision could be to try to merge into gap 5 or gap 6 which are the closest gaps to $object_5$. However, these gaps only refer to the same space with a probability of 20%. Therefore a good decision could also be to stay by gap 2, using the same reasoning as in Test case 1.1. If $vehicle_{Front}$ is getting too close, the host car should not try to merge into any gap but slow down to keep a safe distance to the car in front.

Behavior of host car
The host car first tries to merge into gap 3, but this gap turns out to be unfavorable (since the status of all suspected ghost-cars are *real car*). The car in front of the host car in the same lane, $vehicle_{Front}$, is blocking gap 5 so the host car waits for gap 2 and it stays adjacent to this gap for some time. However, at $t \approx 54$ the host car cannot stay by gap 2 any more but has to adjust its behavior so that it does not drive into $vehicle_{Front}$. The Boolean *solve* equals $False$ and so the action is set to $a_{Stage_1} = -1$ (see Algorithm 3.2.4). The dark gray line with connected stars in Figure 4.2a and Figure 4.2b show the results from Test case 1.4.

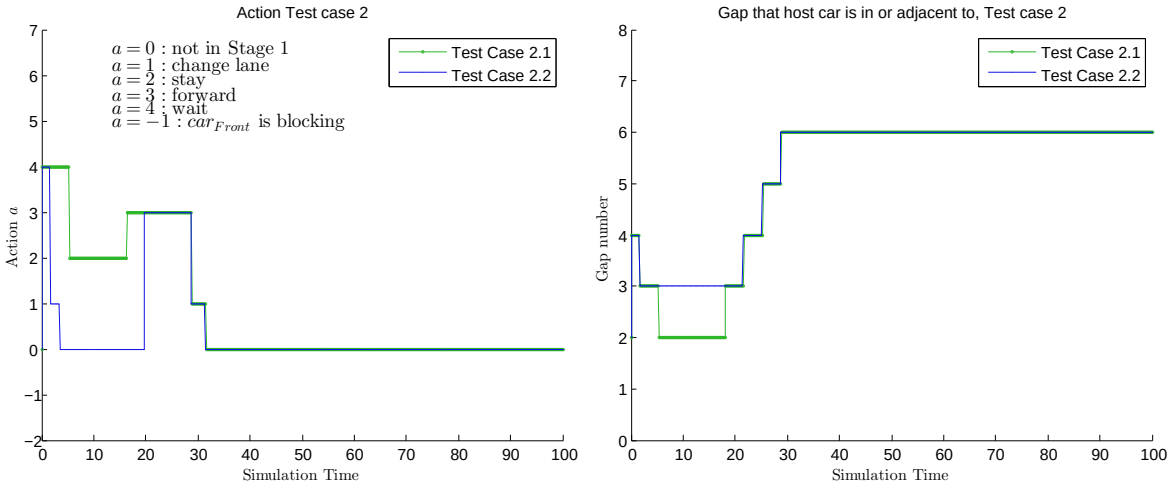
4.2.2 Test Case 2

In all sub-tests in Test case 2 the host car starts by gap 4 and the speed limit is set to 65 km/h. All cars in the right lane are first equally spaced and are driving at 50 km/h. However, after a while $object_6$ increases its speed to 53 km/h which eventually makes gap 6 large enough to merge into.

Test case 2.1
In Test case 2.1 the certainty vector is set to $certainty = [1\ 1\ 1\ 1\ 1\ 1]$, thus there are no suspected ghost-cars.

Desirable behavior of host car
By using the same reasoning as in Test case 1.1, a good decision would be to stay by gap 2 during the time when all cars are equally spaced. After a while when gap 6 has become larger a good decision would be to try to merge into this gap. The host car will have to accelerate to reach this gap but since the goal of merging into the right lane may be fulfilled the increased impact level associated with such an action can be justified.

Behavior of host car
The host car waits for gap 2 and stays adjacent to this gap and when the size of gap 6 increases the host car drives to this gap where it shows its intention and successfully merges. The actions executed and the gap that the host car is in or adjacent to for all time steps can be seen in Figure 4.3a and Figure 4.3b where the green line with connected circles represents Test case 2.1.



(a) Actions executed versus time. A decision, followed by an action, is made every 0.2 s.

(b) Number of the gap that the host car is adjacent to versus time.

Figure 4.3: Results from the sub-tests in Test case 2. In Test case 2.1 the host car waits (action a_4) for gap 2 and stays (action a_2) there until gap 6 becomes larger. Then the host car drives to this gap (action a_3) where it merges successfully. In this test case there are no suspected ghost-cars. In Test case 2.2 there is one suspected ghost-car which has status *real car* with probability 0.7. The host car tries to merge into gap 3 but without success. Next the host car tries to merge into gap 6 even though the host car can obtain a higher reward for trying to merge into gap 3 again. However, at gap 6 the host car is guaranteed to receive a quite high reward for a lane change for all values of X_{comb} but this is not the case for gap 3.

Test case 2.2
In Test case 2.2 the certainty vector is set to $certainty = [1\ 1\ 0\ 1\ 1\ 1]$ and the probability vector for suspected ghost-cars equals $prob_{real} = [0.7]$, thus $object_3$ is a suspected ghost car and has status *real car* with probability 0.7. The variable $T_{Bad_{gap}}$ is set to zero (see Appendix A) so that the host car will forget any unfavorable gap right away.

Desirable behavior of host car
Before gap 6 has become larger a good decision could be to try to merge into one of the gaps closest to $object_3$, that is gap 3 or gap 4. There is an 30% chance that $object_3$ has status *ghost car* and therefore there is a 30% chance that gap 3 and gap 4 refer to the same space which would then be large enough to try to merge into. After gap 6 has become larger a good decision would be to try to merge into this gap. Even though gap 6 is closer to the end-point $object_6$ is a car with high probability and therefore the size of gap 6 is known whereas this is not the case for gap 3 and gap 4.

Behavior of host car
The host car first waits for gap 3 and shows its intention to merge into this gap. After failing to merge into gap 3 the host drives forward to gap 6 which has now become larger. In inspecting the reward matrix at this time instance, it can be seen that the host car can receive the highest rewards if it executes action a_1 ="change lane" and merges into gap 3 or gap 4 ($T_{Bad_{gap}} = 0$ which is why a lane change into gap 3 can give a non-zero reward, even though the host car just tried to merge into this gap). However, these rewards can only be obtained if the status of the suspected ghost-car is *ghost-car*. If the suspected ghost-car has status *real car*, no rewards can be obtained for a lane change neither when the host car is by gap 3 nor by gap 4. On the other hand, the host car can always receive a quite high reward for changing lane and merging into gap 6, regardless of the status of the suspected ghost-car and therefore this gap turns out to be the most favorable. The host car successfully merges into gap 6. The actions executed and the gap that the host car is in or adjacent to in Test case 2.2 can be seen from the blue solid line in Figure 4.3a and Figure 4.3b.

4.2.3 Test Case 3

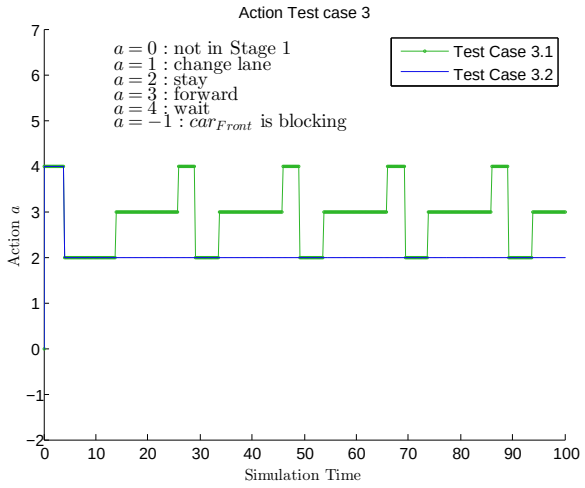
In all sub-tests in Test case 3 the host car starts by gap 4 and the speed limit is set to 75 km/h. All cars in the right lane except car 6 are driving at 70 km/h and they are equally spaced with none of the gap sizes larger than $GapSafety$. Car 6 is frequently changing speed so that it drives above 70 km/h for some time and then below 70 km/h for some time. This means that gap 6 is also frequently changing its size. The maximum size of gap 6 is larger than $GapSafety$.

Test case 3.1
In test case 3.1 the certainty vector is set to $certainty = [1\ 1\ 1\ 1\ 1\ 1]$, thus there are no suspected ghost-cars. The discount factor is set to $\gamma = 0.95$ (see Appendix A).

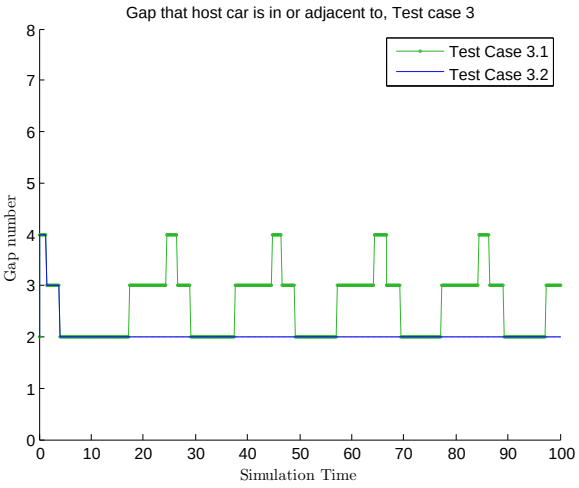
Desirable behavior of host car
A good decision in this test case could be to try to merge into gap 6 since when the size of this gap starts to increase a reasonable assumption is that gap 6 may eventually become large

enough to try to merge into. The host car does not start by gap 6 and therefore it will take some time before it can reach this gap and by that time the traffic situation might have changed so that gap 6 is not a preferable gap to try to merge into. Taking this into account a good decision could also be to try to detect other gaps. This means that a good decision could also be to stay by gap 2, using the reasoning in Test case 1.1.

Behavior of host car
The actions executed and the gaps that the host car is adjacent to in Test case 3.1 are shown with the green line with connected circles in Figure 4.4a and Figure 4.4b. It can be seen that the host car frequently changes its position. When gap 6 becomes larger the host car accelerates and drives towards this gap. However, when the gap becomes smaller the host car waits by gap 2. The behavior is repeated.



(a) Actions executed versus time. A decision, followed by an action, is made every 0.2 s.



(b) Number of the gap that the host car is adjacent to versus time.

Figure 4.4: Results from the sub-tests in Test case 3 in which there are no suspected ghost-cars. In Test case 3.1 the discount factor is set to $\gamma = 0.95$ and the host car changes its position relative to a gap frequently. In Test case 3.2 the discount factor is set to $\gamma = 0.34$. In comparing the results from this test with Test case 3.1 it can be seen that a lower value on the discount factor results in smoother driving.

Test case 3.2
In test case 3.2 the certainty vector is still set to $certainty = [1\ 1\ 1\ 1\ 1\ 1]$ but the discount factor is decreased to $\gamma = 0.34$.

Desirable behavior of host car
The reasoning made in Test case 3.1 can be used also in this test case.

Behavior of host car
The actions executed and the gaps that the host car is adjacent to are presented with the blue solid line in Figure 4.4a and Figure 4.4b. It can be seen that the host car chooses to stay at gap 2 throughout the whole simulation.

4.2.4 Test Case 4

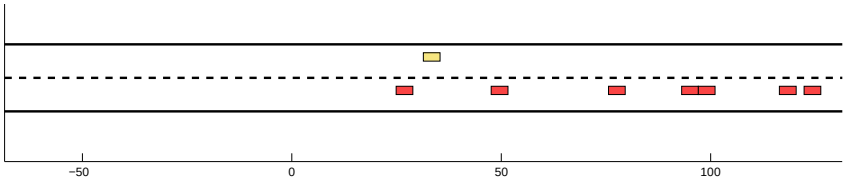
The merging decision algorithm is in this test case analyzed with data collected from a real traffic scenario. All objects that were detected on the right side of the test car and that were traveling with a speed exceeding 10 m/s are included. The lateral positions of the objects are adjusted so that all objects appear on a straight line in the simulation but the longitudinal positions are not altered. In this test case all objects are set to be real cars. Many objects come and go throughout the simulation because an object detected at one time instance may not be detected the next time instance. However, there are many gaps available that are large enough to try to merge into.

Desirable behavior of host car
In this test case the host car should drive to the closest gap that is large enough to try to merge into and whose size is not quickly decreasing.

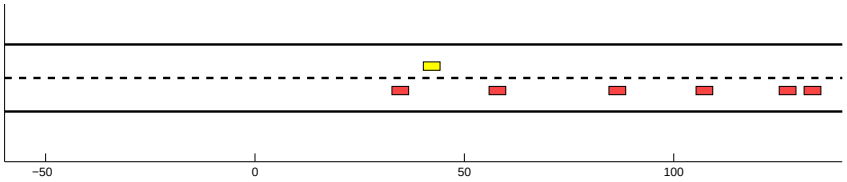
Behavior of host car
The host car first drives to the middle of gap 2 which is the closest gap to the host car that fulfills the conditions required in order to try to merge into a gap ($B_{LC} = True$, see (3.7)). The host car makes an attempt to merge into this gap but before it has reached the middle of the right lane all objects detected behind the host car have disappeared and then the host car is by gap 1 which is considered an unfavorable gap to merge into, causing the host car to return to the left lane. Figure 4.5a - Figure 4.5d show this phenomena. In the end of the simulation the host car manages to merge into a gap between two cars that are continuously detected by the sensor system.

4.3 Policy Evaluation

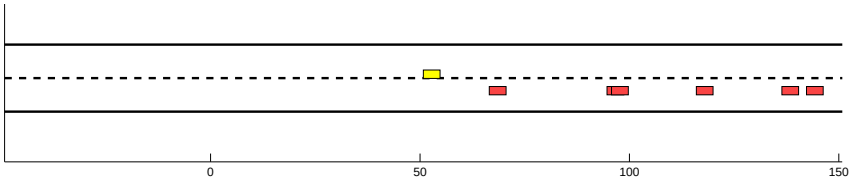
A policy π can be evaluated by applying it in a simulation where at each time step t' an action provided from the policy is executed. The action to be executed a is given by the policy, the current state s is sampled in accordance with the probabilities given in the current belief state b, and the observation outcome o given the state s is determined stochastically in accordance with the probabilities given in the observation function. For every action executed the agent receives a reward $R_{t'}$ which depends on the current state s. The rewards obtained are discounted with means of the discount factor γ and the sum of discounted rewards gained at time step $t' = t$ is calculated, $\sum_{t'=1}^{t} R_{t'}\gamma^{t'-1}$. The sequence of rewards creates a trajectory of rewards $\{R_{t'}\}_{t'=1,2,\ldots,n}$ which can be associated with the sequence of actions - the *action trajectory* - $\{a_{t'}\}_{t'=1,2,\ldots,n}$. Since the dynamics in a POMDP model (or an MDP model) typically are not deterministic the amount of reward obtained from using the policy can vary from time to time. Therefore, in order to test the performance of a policy it has to be simulated several times. The action trajectory should ideally, in the case of an infinite-horizon discounted reward POMDP, be infinitely long



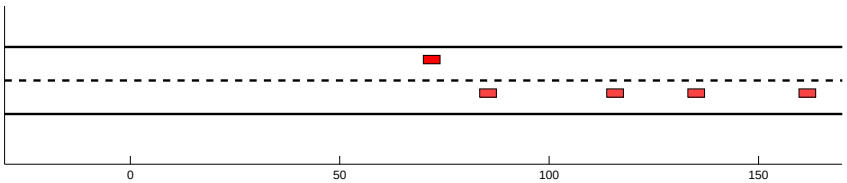
(a) The host car is in Stage 1 in the left lane adjacent to gap 2.



(b) The host car is in Stage 2 and shows its intention to merge into gap 2.



(c) The car that was behind the host car in Figure 4.5b has now disappeared so the host car is in gap 1 which is considered an unfavorable gap.



(d) The host car has canceled the merging operation and returned to Stage 1.

Figure 4.5: Results from test case 4. Before the host car has reached the middle of the right lane it is in gap 1 and therefor the merging operation is canceled.

$(n \to \infty)$. Of course it is not possible to simulate an infinitely long trajectory but the performance of a policy may be compared to other policies also with shorter trajectories $(n = N_{steps})$. The higher reward that (on average) can be obtained from using a policy, the better is the policy.

In this section the performance of policies obtained from Stage 1 are evaluated. The driving scenarios described in Test case 1.2 and Test case 2.2 are used and for each of these scenarios two POMDP models are extracted at two different time instances. These POMDP models are used for creating a total of ten different policies $\Pi = \{\pi_i | i = 1,2,\ldots,10\}$ described in Table 4.2. The two last policies in Table 4.2 are obtained from using a POMDP solver called the gapMin algorithm which is available for download and described in [11]. This algorithm finds a lower and an upper bound on the value function and the policies corresponding to the lower bound value function and the upper bound value function are evaluated. Every policy is simulated with its corresponding POMDP model $N_{sim} = 300$ times and in each simulation an action trajectory

Table 4.2: Names, notations and descriptions of the policies evaluated.

Name of policy	**Notation**	**Description of policy**
Greedy policy	π_1	The policy chooses the action that can give the highest reward when just looking one step forward. The likelihood of receiving the reward is not considered.
MDP-policy	π_2	The policy corresponding to α_0 which is computed with means of the value function V_{MDP} provided from solving the MDP model (see (3.2.3)).
($\|\mathcal{B}\| = x$, $Max_{iter} = y$)-policy	$\pi_3, \ldots, \pi_8$	The policy corresponding to the value function resulting from calling PointBasedValueIteration (Algorithm 3.2.1) $Max_{iter} = y$ times and using $\|\mathcal{B}\| = x$ number of belief points selected with means of SelectBeliefPoints (Algorithm 3.2.2).
gapMin UB policy	π_9	The policy corresponding to the upper bound value function provided from the gapMin algorithm.
gapMin LB policy	π_{10}	The policy corresponding to the lower bound value function provided from the gapMin algorithm.

of length $N_{steps} = 1000$ is used. The *average discounted reward* at time step $t = 1,2,\ldots,N_{steps}$ for policy π, denoted $ADR_\pi(t)$, is then calculated as,

$$ADR_\pi(t) = \frac{1}{N_{sim}} \sum_{sim=1}^{N_{sim}} \sum_{t'=1}^{t} R^\pi_{sim,t'} \gamma^{t'-1}, \tag{4.1}$$

where $R^\pi_{sim,t'}$ is the reward obtained at time step t' in simulation sim using policy π. The *normalized average discounted reward* at time step t' for policy π, denoted $ADR^{norm}_\pi(t)$, is computed as,

$$ADR^{norm}_\pi(t) = \frac{ADR_\pi(t)}{\max_{\bar{\pi} \in \Pi}(ADR_{\bar{\pi}}(N_{steps}))}, \quad \pi \in \Pi. \tag{4.2}$$

The normalized average discounted reward (normalized ADR) is used in the policy evaluation tests since it can be used to compare the performance of policies from different tests.

Table 4.3: Value of normalized ADR at time step $t = N_{steps}$ for each policy evaluated in Policy Evaluation Test 1.

Notation	Name of policy	$ADR_{\pi}^{norm}(N_{steps})$
π_1	Greedy-policy	0.8794
π_2	MDP-policy	0.8825
π_3	($\|\mathcal{B}\| = 24$, $Max_{iter} = 100$)-policy	0.9730
π_4	($\|\mathcal{B}\| = 24$, $Max_{iter} = 200$)-policy	0.9837
π_5	($\|\mathcal{B}\| = 72$, $Max_{iter} = 100$)-policy	0.9645
π_6	($\|\mathcal{B}\| = 72$, $Max_{iter} = 200$)-policy	1.0000
π_7	($\|\mathcal{B}\| = 120$, $Max_{iter} = 100$)-policy	0.9828
π_8	($\|\mathcal{B}\| = 120$, $Max_{iter} = 200$)-policy	0.9921
π_9	gapMin UB-policy	0.8760
π_{10}	gapMin LB-policy	0.9896

4.3.1 Policy Evaluation Test 1

In Policy evaluation test 1 the POMDP model computed at simulation time $t = 2.4$ in Test case 1.2 in Section 4.2.1 is used and at this time instance the host car is driving forward to gap 5, see solid blue lines in Figure 4.2a and Figure 4.2b. Figure 4.6a shows, for each time step, the normalized ADR for the policies described in Table 4.2 with $|\mathcal{B}| \in \{24, 72, 120\}$ and $Max_{iter} \in \{100, 200\}$. Figure 4.6b is a zoomed in version of Figure 4.6a. In Table 4.3 the normalized ADR values for time step $t = N_{steps}$ are shown for each policy. In inspecting Table 4.3 it can be seen that the policy computed from the highest number of belief points and the highest number of iterations, that is the ($|\mathcal{B}| = 120, Max_{iter} = 200$)-policy, does not perform the best but is outperformed by one other policy, the ($|\mathcal{B}| = 72, Max_{iter} = 200$)-policy. This phenomenon may be explained by the fact that the PBVI does not guarantee that the policy improves monotonically with the number of belief points or the number of iterations (see Section 2.3.3). From Table 4.3 it can also be seen that the policy that performs the worst is the gapMin UB -policy, its final normalized ADR value is 0.8760.

4.3.2 Policy Evaluation Test 2

In Policy evaluation test 2 the POMDP model computed at simulation time $t = 27$ in Test case 1.2 in Section 4.2.2 is used and at this time instance the host car is waiting for gap 2, see solid blue lines in Figure 4.2a and Figure 4.2b. Figure 4.7a shows how the policies described in Table 4.2 perform with $|\mathcal{B}| \in \{26, 66, 115\}$ and $Max_{iter} \in \{100, 200\}$. Figure 4.7b is a zoomed in version of Figure 4.7a. In Table 4.4 the normalized ADR values for time step $t = N_{steps}$ are shown for each policy. From this table it can be seen that the greedy-policy and the gapMin LB-policy perform the worst and that the remaining policies perform equally well. However, since the final normalized ADR values for the policies that perform the worst is greater than 0.99 the difference in performance between all policies in this policy test may be considered small.

Table 4.4: Value of normalized ADR at time step $t = N_{steps}$ for each policy evaluated in Policy Evaluation Test 2.

Notation	Name of policy	$ADR_{\pi}^{norm}(N_{steps})$
π_1	Greedy-policy	0.9956
π_2	MDP-policy	1.0000
π_3	($\|\mathcal{B}\| = 26$, $Max_{iter} = 100$)-policy	1.0000
π_4	($\|\mathcal{B}\| = 26$, $Max_{iter} = 200$)-policy	1.0000
π_5	($\|\mathcal{B}\| = 66$, $Max_{iter} = 100$)-policy	1.0000
π_6	($\|\mathcal{B}\| = 66$, $Max_{iter} = 200$)-policy	1.0000
π_7	($\|\mathcal{B}\| = 115$, $Max_{iter} = 100$)-policy	1.0000
π_8	($\|\mathcal{B}\| = 115$, $Max_{iter} = 200$)-policy	1.0000
π_9	gapMin UB-policy	1.0000
π_{10}	gapMin LB-policy	0.9956

4.3.3 Policy Evaluation Test 3

In Policy evaluation test 3 the POMDP model computed at simulation time $t = 2.4$ in Test case 2.2 in Section 4.2.2 is used and at this time instance the host car is by gap 3, see Figure 4.3a and Figure 4.3b. Figure 4.8a shows how the policies described in Table 4.2 perform with $|\mathcal{B}| \in \{24, 67, 98\}$ and $Max_{iter} \in \{100, 200\}$. Figure 4.8b is a zoomed in version of Figure 4.8a. In Table 4.5 the normalized ADR values for time step $t = N_{steps}$ are shown for each policy. From this table it can be seen that the greedy-policy, which performs the worst, performs significantly worse than the greedy-policy in Policy Evaluation Test 1 and the greedy-policy in Policy Evaluation Test 2. Also in this test the non-monotonic policy improvement feature of the PBVI algorithm is apparent since the policy with the highest number of belief points and the highest number of iterations does not perform the best. However, all policies but the greedy-policy have a final normalized ADR value that is greater than 0.9, thus the best policy may not be considered significantly better than the other policies.

4.3.4 Policy Evaluation Test 4

In Policy evaluation test 4 the POMDP model computed at simulation time $t = 21.2$ in Test case 2.2 in Section 4.2.2 is used and at this time instance the host car is driving forward to gap 5, see Figure 4.3a and Figure 4.3b. Figure 4.9a shows how the policies described in Table 4.2 perform where $|\mathcal{B}| \in \{24, 67, 97\}$ and $Max_{iter} \in \{100, 200\}$. Figure 4.9b is a zoomed in version of Figure 4.9a. It can be seen that the greedy policy once again performs significantly worse than the other policies. The remaining policies perform almost equally well since they all have a final normalized ADR value above 0.99 which can be seen in Table 4.6.

Table 4.5: Value of normalized ADR at time step $t = N_{steps}$ for each policy evaluated in Policy Evaluation Test 3.

Notation	**Name of policy**	$ADR_{\pi}^{norm}(N_{steps})$
π_1	Greedy-policy	0.3885
π_2	MDP-policy	0.9070
π_3	($\|\mathcal{B}\| = 24$, $Max_{iter} = 100$)-policy	0.9725
π_4	($\|\mathcal{B}\| = 24$, $Max_{iter} = 200$)-policy	0.9635
π_5	($\|\mathcal{B}\| = 67$, $Max_{iter} = 100$)-policy	0.9718
π_6	($\|\mathcal{B}\| = 67$, $Max_{iter} = 200$)-policy	0.9784
π_7	($\|\mathcal{B}\| = 98$, $Max_{iter} = 100$)-policy	1.0000
π_8	($\|\mathcal{B}\| = 98$, $Max_{iter} = 200$)-policy	0.9876
π_9	gapMin UB-policy	0.9052
π_{10}	gapMin LB-policy	0.9753

Table 4.6: Value of normalized ADR at time step $t = N_{steps}$ for each policy evaluated in Policy Evaluation Test 4.

Notation	**Name of policy**	$ADR_{\pi}^{norm}(N_{steps})$
π_1	Greedy-policy	0.3360
π_2	MDP-policy	0.9983
π_3	($\|\mathcal{B}\| = 24$, $Max_{iter} = 100$)-policy	0.9938
π_4	($\|\mathcal{B}\| = 24$, $Max_{iter} = 200$)-policy	0.9966
π_5	($\|\mathcal{B}\| = 67$, $Max_{iter} = 100$)-policy	0.9956
π_6	($\|\mathcal{B}\| = 67$, $Max_{iter} = 200$)-policy	0.9985
π_7	($\|\mathcal{B}\| = 97$, $Max_{iter} = 100$)-policy	0.9968
π_8	($\|\mathcal{B}\| = 97$, $Max_{iter} = 200$)-policy	1.0000
π_9	gapMin UB-policy	0.9987
π_{10}	gapMin LB-policy	0.9980

4.4 Speed Performance of Solving the POMDP Model

In this section the execution time of the merging decision algorithm is evaluated. The execution time of the algorithm depends almost entirely on the time required for selecting belief points (Algorithm 3.2.2) and then running the PBVI algorithm (Algorithm 3.2.1). Therefore only these two steps are evaluated for various limits on the maximum number of iterations and a varying number of selected belief points. The speed performance tests ran on a 2.5 GHz processor. Table 4.7 shows the execution times for solving a POMDP model computed from a problem with

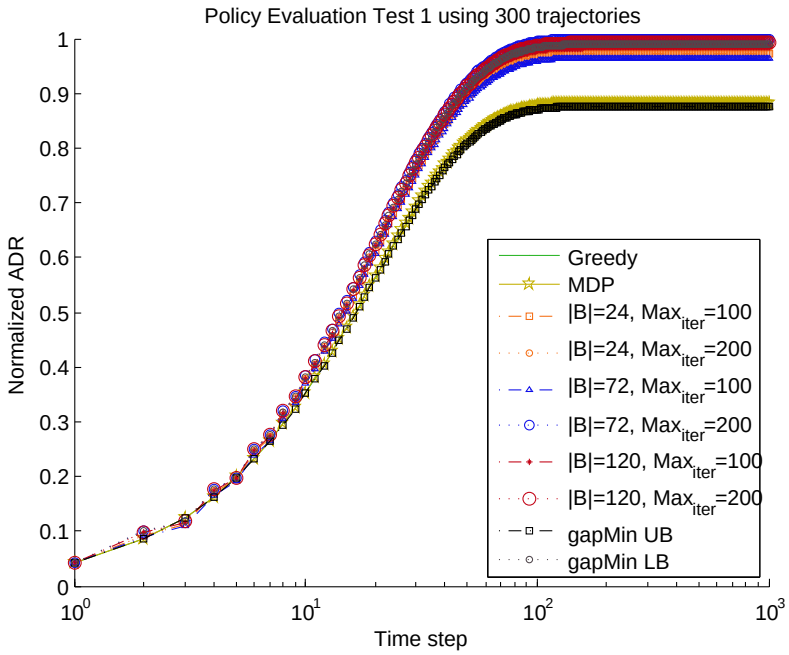
six cars in the right lane where one of the cars is classified as a suspected ghost-car. In total the POMDP model has 28 states. It is clear that the execution time increases with the maximum number of iterations and the number of selected belief points. Table 4.8 shows the corresponding values for a 56 state POMDP. The execution time required for the gapMin algorithm described in Section 4.3 is not included in the table since it varies significantly for different POMDP models with the same number of states. In the policy evaluation tests described in Section 4.3 the execution time of the gapMin algorithm varied between a couple of hundreds of milliseconds to more than seven seconds. In all but one of the policy evaluation tests the gapMin algorithm required longer execution time than the POMDP solver implemented in Stage 1. The time required to compute the MDP-policy in the policy evaluation tests was about 0.006 seconds. Hence the MDP policy requires significantly shorter execution time than the policies provided from using the PBVI algorithm.

Table 4.7: Execution times required for selection of belief points and running the PBVI algorithm Max_{iter} times for a 28 state POMDP. Times are given in seconds.

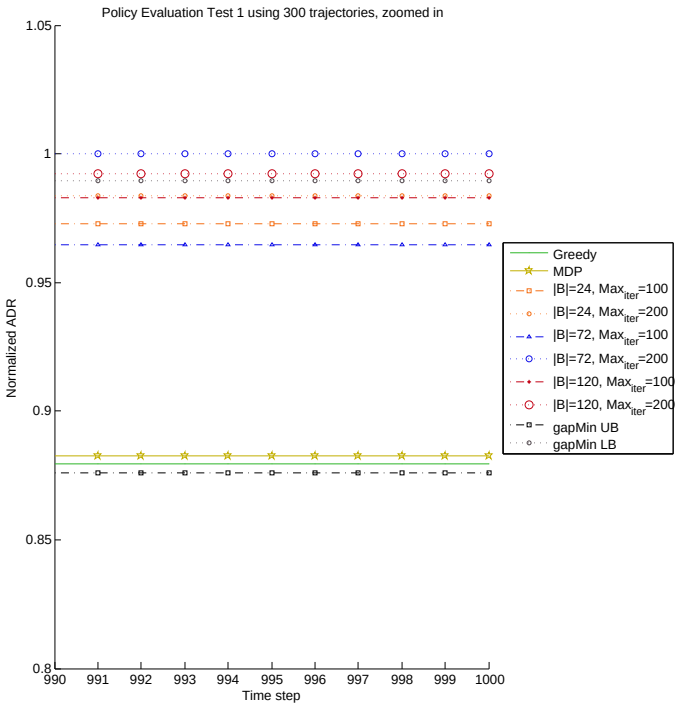
	$Max_{iter} = 100$	$Max_{iter} = 200$	$Max_{iter} = 300$	$Max_{iter} = 400$
$\|\mathcal{B}\| = 24$	0.1328	0.2649	0.3920	0.5142
$\|\mathcal{B}\| = 69$	0.2228	0.4462	0.6439	0.8780
$\|\mathcal{B}\| = 107$	0.3371	0.6916	1.0750	1.3491

Table 4.8: Execution times required for selection of belief points and running the PBVI algorithm Max_{iter} times for a 56 state POMDP. Times are given in seconds.

	$Max_{iter} = 100$	$Max_{iter} = 200$	$Max_{iter} = 300$	$Max_{iter} = 400$
$\|\mathcal{B}\| = 29$	0.1718	0.3434	0.5025	0.6858
$\|\mathcal{B}\| = 56$	0.2671	0.6109	0.7356	0.9676
$\|\mathcal{B}\| = 100$	0.3952	0.7905	1.1563	1.5080

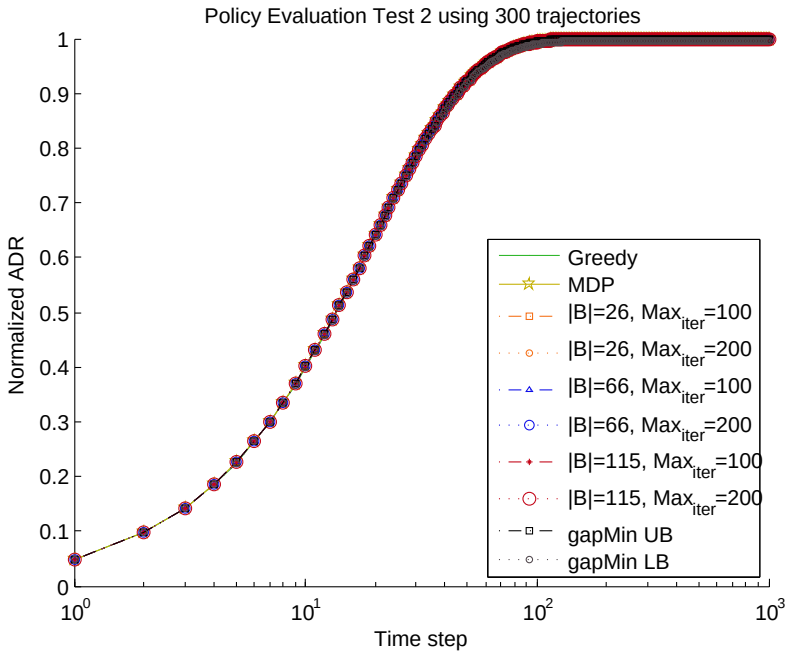


(a) Normalized averaged discounted reward versus time step.

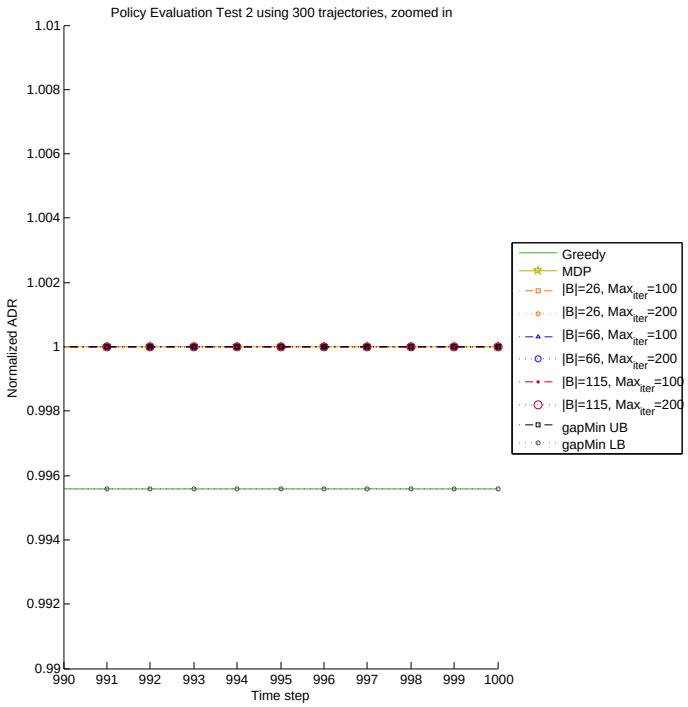


(b) Figure 4.6a zoomed in.

Figure 4.6: Results from Policy evaluation test 1. The policy computed from the highest number of belief points and the highest number of iterations is outperformed by one other policy computed from the POMDP solver implemented in Stage 1. The gapMin UB-policy performs the worst.

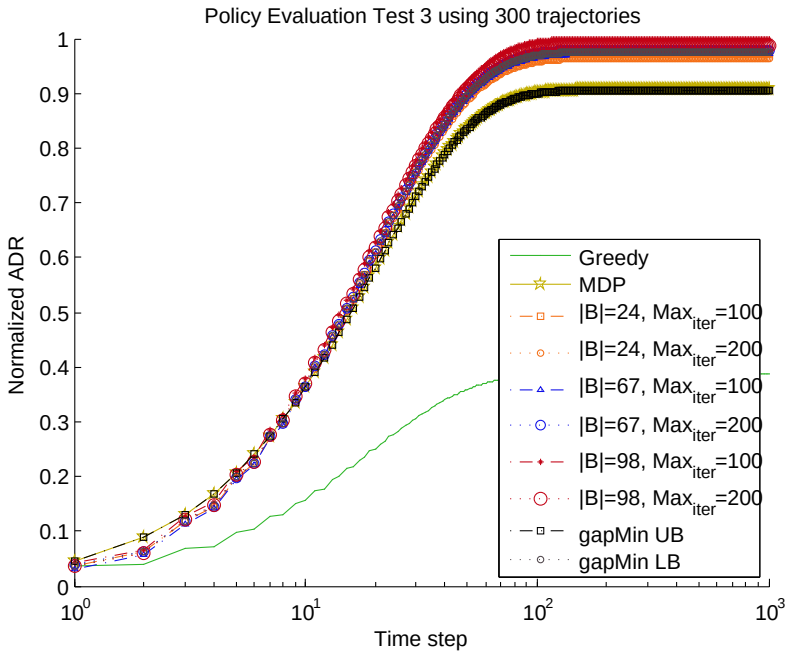


(a) Normalized averaged discounted reward versus time step.

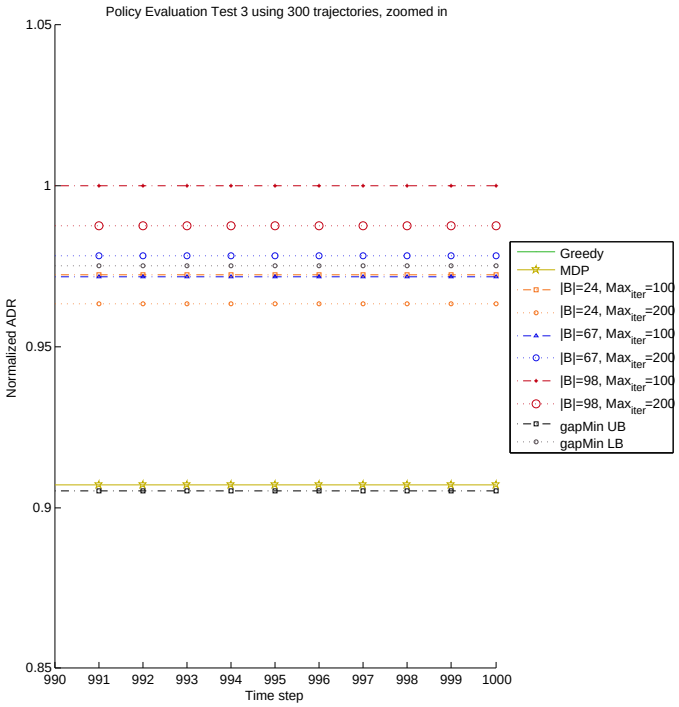


(b) Figure 4.7a zoomed in.

Figure 4.7: Results from Policy evaluation test 2. The greedy-policy and the gapMin LB-policy perform the worst. The policies computed from the POMDP solver implemented in Stage 1 perform equally well.

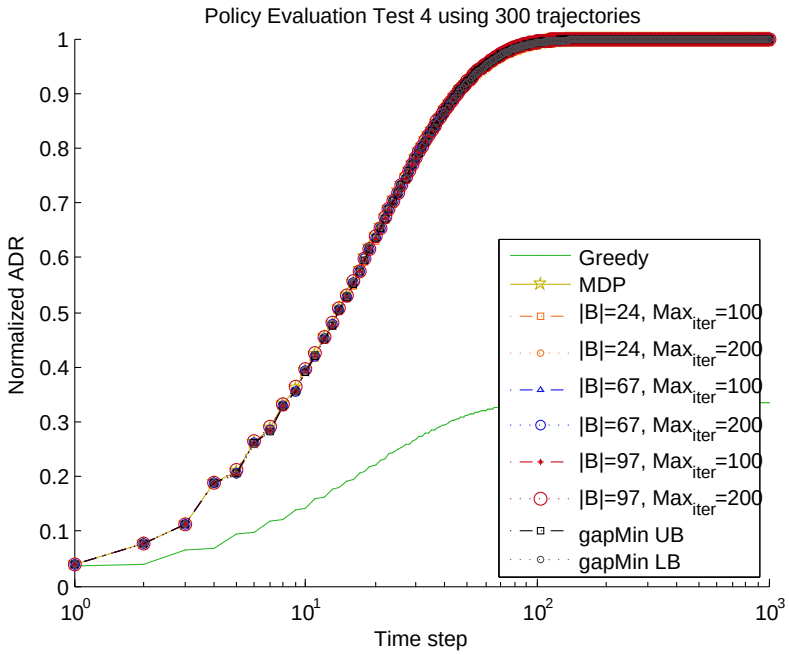


(a) Normalized averaged discounted reward versus time step.

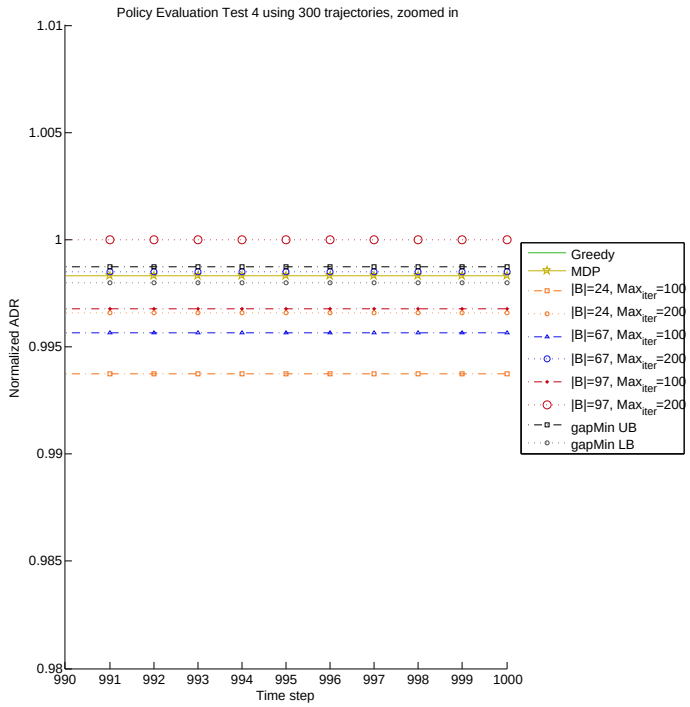


(b) Figure 4.7a zoomed in.

Figure 4.8: Results from Policy evaluation test 3. In this test the non-monotonic policy improvement feature of the PBVI algorithm can be seen. However, the difference in performance is not significant for all but the greedy-policy.



(a) Normalized averaged discounted reward versus time step.



(b) Figure 4.7a zoomed in.

Figure 4.9: Results from Policy evaluation test 4. All of the policies except the greedy-policy perform almost equally well.

5

Discussion and Conclusions

This chapter discusses the developed merging decision algorithm and states the conclusions. In Section 5.1 the results presented in Chapter 4 are discussed and conclusions are drawn. Section 5.2 discusses the applicability of the proposed merging decision algorithm in other traffic situations and Section 5.3 proposes some algorithm improvements. Lastly, in Section 5.4 some concluding remarks are given.

5.1 Comments on the Results

In Chapter 4 the proposed merging decision algorithm was tested in various simulated traffic scenarios, the performance of policies were evaluated and the execution time of the algorithm was examined. This section discusses the results.

5.1.1 Quality of the Decisions Made

In Section 4.2 the desirable behavior of the host car was discussed for all test cases and from the results obtained it can be concluded that the host car behaved as desired. From the discussion on the desirable behavior it is also evident that many times there are more than just one decision that may be considered appropriate. In Test case 3.1 the discount factor γ was set to a higher value than in Test case 3.2 and it was shown that by changing the value of γ the host car also made different decisions both of which were considered appropriate. However, a passenger that prefer smoother driving would choose the lower value on γ in order to favor a behavior where the host car changes its position relative to a gap less frequently.

5.1.2 Attractiveness of a Gap

In Section 4.2 it was shown how the host car acts in correspondence to the probability of the status of the suspected ghost-car(s), given by $prob_{real}$. A lower confidence on the *real car* status means a higher chance that the object is a ghost-car and this makes the gaps next to the suspected ghost-car more attractive than if the suspected ghost-car has status *real car* with high

confidence. This can be seen when comparing the results from Test case 1.1, Test case 1.2 and Test case 1.3. In Test case 1.1 there are no suspected ghost-cars, but in Test case 1.2 the fifth object is classified as a suspected ghost-car which is assumed to have status *real car* with 50% confidence. This is why gap 5 is more attractive in Test case 1.2 than in Test case 1.1. In Test case 1.3 the fifth car, still classified as a suspected ghost-car, is assumed to have status *real car* with 90% confidence making gap 5 less attractive compared to Test case 1.2. In Test case 1.3 the host car behaves as in Test case 1.1, hence the confidence of the *real car* status influences the attractiveness of a gap.

5.1.3 Data with High versus Data with Low Confidence

In Test case 2 it is shown how data with high confidence is favored over data with low confidence. The host car chooses a gap with a lower reward since this gap guarantees quite high rewards for all possible values of X_{comb} (0_2 and 1_2) in contrast to the highest rewarded gap which only has the highest reward for one out of the two possible values of X_{comb} (only 0_2). The gap corresponding to the lower reward is generated by objects that are classified as cars with high confidence and it is therefore independent of the status of the suspected ghost-car.

5.1.4 Using the Algorithm with Data from a Real Traffic Scenario

In Section 4.2.4 the merging decision algorithm was tested with data collected with means of a test car driving in real traffic. It was described how the host car tried to merge into a gap without succeeding due to the fact that an object in the right lane behind the host car could not be detected throughout the merging maneuver. Since the objects detected in the right lane are the objects that the test car detected without applying the merging decision algorithm it is reasonable to assume that other objects would have been detected if the test car would have acted in accordance with the merging decision algorithm. This test is still valuable since the merging decision algorithm is used on data that better imitates driving behavior of vehicles in the right lane.

5.1.5 Policy Performance

In Section 4.3 the performance of policies provided by the POMDP solver implemented in Stage 1 were evaluated. It can be concluded that in the policy tests the greedy-policy is outperformed by all policies computed from the POMDP solver implemented in Stage 1 (Algorithm 3.2.2 and Algorithm 3.2.1). In inspecting Table 4.3 - Table 4.6 it can be concluded that the difference in performance between the policies provided from the POMDP solver implemented in Stage 1 is relatively small; the lowest normalized ADR value at time step $t = N_{steps}$ for these policies is 0.9635 (see Table 4.5). From Table 4.3 - Table 4.6 it can also be concluded that the policies provided from the POMDP solver implemented in Stage 1 perform very well in comparison to the gapMin algorithm. Also the MDP-policy, which is computed using value iteration, outperformed at least one of the gapMin-policies in all policy performance tests.

In the policy evaluation tests the non-monotonic policy improvement feature of the PBVI algorithm is evident. This feature is disadvantageous since it means that that even though more time

is spent on finding a better approximate solution of a POMDP model the resulting policy is not guaranteed to perform better than a policy found after a shorter period of time. The resulting policy can even perform worse. Another approximate solution method for POMDPs is called Point Based Policy Iteration which, according to [14], does not suffer from this disadvantage, hence this method would be interesting to investigate further.

5.1.6 Execution Time Required

Section 4.4 also reveals a less advantageous feature of the implemented POMDP solver. The POMDP solver used in Stage 1 is in general faster than the gapMin algorithm, but it is still not as fast as desired. In this real time application, 'not so fast' refers to more than a couple of hundreds of milliseconds. Much effort has been put in trying to optimize the code, for example by using the fact that the observation matrix is deterministic, but the inherent computational time complexity associated with solving a POMDP still remains.

The speed of the algorithm may be improved if sparse matrices are used instead, however the sparse matrix feature is not available in embedded Matlab so the sparse functions would first have to be implemented in C-code. The PBVI algorithm implemented in Stage 1 is considered to be a promising POMDP solver choice, however one may test other available POMDP solvers to see if that speeds up the merging decision algorithm.

Some POMDP models that need to deliver decisions in real time can first be solved roughly offline after which a better solution based on the rough solution and the current belief point can be computed online. This is not an applicable approach for the POMDP model described in this thesis since the POMDP model is not only solved online but also created in real time. When the traffic situation changes, so does the POMDP model.

Even though it would be advantageous if the POMDP solver used in Stage 1 executes faster, the policy evaluation tests show that the MDP-policy perform quite well since in all policy tests the final normalized ADR value for the MDP-policy was greater than 0.88, which may be considered high. If also taking into account that the MDP-policy can be provided in significantly shorter time than policies provided from the belief selection heuristic and the PBVI algorithm, the MDP-policy appears even more attractive.

5.2 Scalability of the Proposed Model

Probably the most attractive property of the merging decision algorithm is its scalability to other decision making situations for automated cars. There are many situations in which the structure of the proposed POMDP model may be applicable. As an example, in a situation where the host car needs to turn left in an intersection and where the right-of-way rule is applicable the sensors might detect 'suspected ghost vehicles' or 'suspected ghost pedestrians' and this situation can be modeled using a similar structure as the one used in the POMDP model in Stage 1. The actions might be set to 'wait', 'drive forward' and 'turn left' and the state variables might be set to $X_1 \in \{\textit{start lane, middle of intersection, end lane}\}$ and $X_2 = X_{comb}$.

5.3 Additional Improvements

The PBVI algorithm is only allowed to run Max_{iter} times or until the number of α-vectors exceeds Max_{α}. Also the number of reachable belief points $|\mathcal{B}|$ is restricted with means of Max_b. This way the PBVI algorithm is guaranteed to not run for to long. Another approach would be to use a predefined time limit and then let the PBVI algorithm run until the time limit is reached. This feature was not implemented due to limitations in Simulink.

An additional feature that can further improve the POMDP model is to also account for cases when there are two subsequent suspected ghost-cars. Two suspected ghost-cars in a row would result in a very wide gap if both suspected-ghost cars have status *ghost-car*. The POMDP model proposed in this thesis is a bit more restrictive and only combines at most two gaps (see (3.5)).

The calculation of the vector $prob_{LC}$ which gives the probabilities of the host car being able to change lane from left lane to right lane may be further improved. Factors other than distances between the detected objects in the right lane and the differences in velocity between any two subsequent objects may be included. Preferably the probability values should not be set with a fixed interval since the true probabilities of being able to change lane are most often not ordered with a fixed difference. Section 3.2.1.4 describes how $prob_{LC}$ is calculated.

The merging decision algorithm assumes that all vehicles in the right lane are cars and thus the size of the the vehicles are known to a relatively high accuracy. At present time the sensors in the automated car cannot be used in order to obtain information of the sizes of the vehicles in the right lane. However, if this information is provided the merging decision algorithm can easily be extended so that it also handles different vehicle sizes.

It can be hard to find a set of parameters that creates a reward function that guides the algorithm in a desirable way in all situation, especially without using a real test car for verification. However, if it is of interest to add more features this can easily be done by extending expressions (3.7) - (3.10).

5.4 Concluding Remarks

This thesis has proposed an algorithm for decision making for automated cars in merging situations. The performance has been evaluated in a simulation environment with simulated data as well as data collected from a real traffic scenario. The algorithm has not yet been tested in a real test car.

Solving the POMDP model exactly is very time consuming even for moderate sized problems but as shown in this thesis the POMDP model can be solved approximately in real time with good results. Also, in case of perfect sensors, an ϵ-consistent solution of the underlying model can be found in real time. Incorporating a POMDP solver that is faster and that guarantees that the policy is improved monotonically, if such a solver exists, would further improve the solving of the POMDP model computed in Stage 1. Hopefully the algorithm proposed in this thesis

will serve as a foundation for further development and subsequent implementation in vehicles at Volvo Cars.

Bibliography

[1] Volvo car group initiates world unique swedish pilot project with self-driving cars on public roads, https://www.media.volvocars.com/global/en-gb/media/pressreleases/136182/volvo-car-group-initiates-world-unique-swedish-pilot-project-with-self-driving-cars-on-public-roads, accessed: 2014-01-21 (2013).

[2] J. Nilsson, J. Sjöberg, Strategic decision making for automated driving on two-lane, one-way roads using model predictive control, IEEE Intelligent Vehicles Symposium (IV), Gold Coast, Australia, 2013.

[3] W. Cao, M. Mukai, T. Kawabe, Two-dimensional merging path generation using model predictive control, Artificial Life and Robotics 17 (3-4) (2013) 350–356.

[4] S. Brechtel, T. Gindele, R. Dillmann, Probabilistic mdp-behavior planning for cars, in: Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on, 2011, pp. 1537–1542.

[5] S. Ulbrich, M. Maurer, Probabilistic online pomdp decision making for lane changes in fully automated driving, in: Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference, 2013, pp. 2063–2067.

[6] J. Wei, J. M. Dolan, J. M. Snider, B. Litkouhi, A point-based mdp for robust single-lane autonomous driving behavior under uncertainties, in: Robotics and Automation (ICRA), 2011 IEEE International Conference on, 2011, pp. 2586–2592.

[7] M. Kolobov, A. Kolobov, Planning with Markov Decision Processes: An AI Perspective, Morgan and Claypool, 2012, [e-book].

[8] M. de Guadalupe Garcia-Hernandez et al., New prioritized value iteration for markov decision processes, The Artificial Intelligence Review 37 (2) (2012) 157–167.

[9] G. Shani, J. Pineau, R. Kaplow, A survey of point-based pomdp solvers, Autonomous Agents and Multi-Agent Systems 27 (1) (2013) 1–51.

[10] H. Geffner, B. Bonet, A Concise Introduction to Models and Methods for Automated Planning, Morgan and Claypool, 2013, [e-book].

[11] P. Poupart, K.-E. Kim, D. Kim, Closing the gap: Improved bounds on optimal pomdp solutions, in: ICAPS, Association for the Advancement of Artificial Intelligence, 2011.

[12] A. R. Cassandra, Pomdps for dummies, http://www.pomdp.org/tutorial/index.shtml, accessed: 2014-04-11 (2013).

[13] J. Pineau, G. J. Gordon, S. Thrun, Anytime point-based approximations for large pomdps, Journal of Artificial Intelligence Research 27 (2006) 335–380.

[14] S. Ji, R. Parr, H. Li, X. Liao, L. Carin, Point-based policy iteration, in: In Proceedings of the Twenty-Second National Conference on Artificial Intelligence, 2007.

Appendix A

Table 1: Parameter values used in Test Case 1 - Test Case 4 in Section 4.2.

Parameter Name	**Test Case 1**	**Test Case 2**	**Test Case 3**	**Test Case 4**
$DistSafetyFront$ [m]	20	20	20	20
$GapSafety$ [m]	10	10	10	10
$GapSafetyLC$ [m]	8	8	8	8
G_{FE}	0.9	0.9	0.9	0.9
G_{FG}	1.8	1.8	1.8	1.8
G_{FM}	0.9	0.9	0.9	0.9
G_{LC}	3	3	3	3
G_{SE}	0.9	0.9	0.9	0.9
G_{SG}	1.8	1.8	1.8	1.8
G_U	50	50	50	50
G_{WE}	0.9	0.9	0.9	0.9
G_{WG}	1.8	1.8	1.8	1.8
G_{WM}	0.9	0.9	0.9	0.9
γ	0.95	0.95	0.95 or 0.34	0.95
Max_α	40	40	40	40
Max_b	20	20	20	20
Max_{iter}	100	100	100	100
$resolution$	1000	1000	1000	1000
$T_{Bad_{gap}}$ [samples]	120	0	120	120
$T^{Host,End}[s]$	25	25	25	25
$T^{Host,Front}$ [s]	3	3	3	3
T_{int} [samples]	60	60	60	60
$T_{LC}^{Mid,Front}$ [s]	20	20	20	20
$T_S^{Mid,Front}$ [s]	10	10	10	10

The maximum allowed speed v_{max} and the minimum allowed speed v_{min} for the host car in Test case 1, 2, 3 and 4 are given by,

$$v_{max} = \begin{cases} \min\Big(v_{limit}, v_{Front}\Big) & \text{, if } \Big(\exists\ vehicle_{Front}\Big) \wedge \dots \\ & \qquad \wedge \Big(lVehicleLeftRelHost < DistSafetyFront\Big), \\ \min\Big(v_{limit}, \text{median}(vObjRight) \cdot G_{v_{max}}\Big) & \text{, otherwise,} \end{cases} \tag{1}$$

$$v_{min} = \begin{cases} \min\Big(\text{median}(vObjRight) \cdot G_{v_{min}}, v_{Front} \cdot G_{v_{min}}\Big) & \text{, if } \Big(\exists\ vehicle_{Front}\Big) \wedge \dots \\ & \qquad \wedge \Big(lVehicleLeftRelHost < DistSafetyFront\Big), \\ \text{median}(vObjRight) \cdot G_{v_{min}} & \text{, otherwise,} \end{cases} \tag{2}$$

where v_{limit} is the speed limit, v_{Front} is the velocity of $vehicle_{Front}$, the vector $vObjRight$ are the velocities of the objects detected in the right lane and the gains $G_{v_{max}}$ and $G_{v_{min}}$ have values 1.2 and 0.8 respectively.