Thesis

A Markov Decision Process approach to human-autonomous driving control logic

Daan Vermunt

Thesis

A Markov Decision Process approach to human-autonomous driving control logic

by

Daan Vermunt

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday August 26, 2020 at 10:00 AM.

Student number:	4319494	
Project duration:	September 1, 2019 – August 30, 2020	
Thesis committee:	Dr. M. T. J. Spaan,	TU Delft, supervisor
	Dr. N. Tintarev,	TU Delft
	Dr. ir. D. Cleij,	SWOV

TU Delft

”If you don’t know where you are going, any road will get you there.”

Lewis Carroll

Abstract

In the road toward autonomous vehicles, we will have to overcome the challenge that a car will not directly be able to drive fully autonomous in all situations a vehicle will come across. One way to tackle this problem is provided by the MEDIATOR project, which aims at creating a solution handling the continuous mediation between autonomous and human-controlled driving. This thesis focuses on one element of the MEDIATOR project referred to as the decision logic. It controls the level of automation the vehicle operates in, depending on the specific condition it encounters. One of the contributions of this thesis is the formalization of the decision logic as a mathematical problem; that is, it is modeled as a Markov decision process. This thesis also gives an implementation of such a model and tests it in a simulated environment for which a tailored simulator was constructed. The feasibility of the proposed Markov decision process approach was tested by comparing it to a simpler heuristic approach. Our results show that the new model is very promising both in terms of planning for the decision logic and meeting of the requirements of the original problem.

Preface

During the creation of the thesis a pandemic hit the world. This made it arguably more important to have people around me that actively helped me in the process of my research. In this preface I would like to thank those people who supported me throughout my entire journey.

First, I would like to thank the my supervisor Matthijs Spaan for guiding me in finding solutions for the problems I encountered in my research. Mainly I would like to thank him for being a partner in the process and giving direction when I did not know were to go.

Also, I would like to thank Canmanie Ponnambalam for helping me in writing down my research. I really appreciate the effort you have made in reading my work and giving me pointers, down to finest details.

Next to the people from the TU Delft there were also many people around me who helped me. I want to thank Syntactics Development BV for allowing me to use their office space to work on my thesis, not only from me but also from my flat mates Toon en Tristan who would arguably have gone mad if we all had to work in our small shared apartment. I also want to thank my family and friends for much needed support and encouragement, with a special thanks to my sister and father for using their academic experience to help me in finding the right words for this thesis.

Finally I want to point out that the icons used in this thesis were created by Freepik from www.flaticon.com.

Daan Vermunt
Delft, September 2020

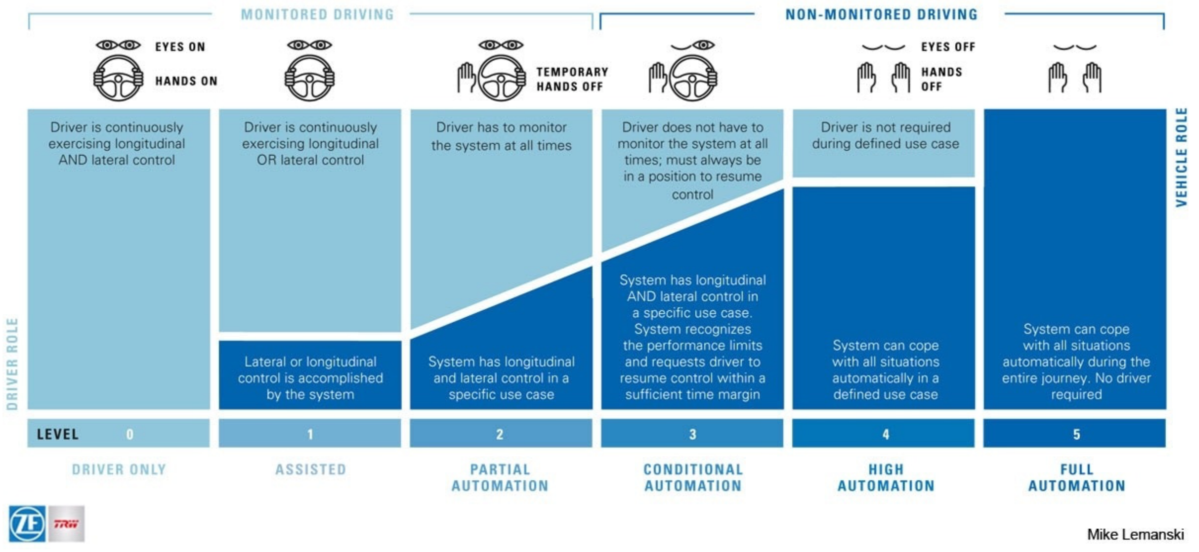
Contents

1 Introduction

Several alternative ways to transition towards autonomous vehicles within the sphere of personal driving have been proposed. This introduction starts with a discussion of the main approaches. Next, the MEDIATOR project is explained which implements one of the possible approaches, where it is also clearly indicated on which part of the MEDIATOR project this thesis focuses. Subsequently, a Markov decision process is introduced and motivated as a possible approach to the problem at hand. Finally, the layout of the rest of this thesis is described.

1.1. Going towards Autonomous driving

In January 2014, the Society of Automotive Engineering (SAE) International published a formal definition of different levels of driving automation [14]. Figure 1.1 provides a visualization of these levels with their explanation. Level 0 concerns a car without any automation features, while level 5 is a vehicle that can fully operate without a human driver and could be built without a steering wheel. The levels in between represent certain degrees of automation. Currently, there are systems up to level 2 available for consumers.



MONITORED DRIVING
NON-MONITORED DRIVING
EYES ON
HANDS ON
TEMPORARY HANDS OFF
EYES OFF
HANDS OFF
Driver is continuously exercising longitudinal AND lateral control
Driver is continuously exercising longitudinal OR lateral control
Driver has to monitor the system at all times
Driver does not have to monitor the system at all times; must always be in a position to resume control
Driver is not required during defined use case
Lateral or longitudinal control is accomplished by the system
System has longitudinal and lateral control in a specific use case
System has longitudinal AND lateral control in a specific use case. System recognizes the performance limits and requests driver to resume control within a sufficient time margin
System can cope with all situations automatically in a defined use case
System can cope with all situations automatically during the entire journey. No driver required
DRIVER ROLE
VEHICLE ROLE
LEVEL
0
1
2
3
4
5
DRIVER ONLY
ASSISTED
PARTIAL AUTOMATION
CONDITIONAL AUTOMATION
HIGH AUTOMATION
FULL AUTOMATION
Mike Lemanski

Figure 1.1: Levels of driving automation [16]

Many commercial and academic ventures are working on achieving level 5 automation [3], because then the applications have a significant increase of impact for society. However, there are two possible ways to get there, either through level 3 and 4 automation or directly, by skipping these two levels. The second approach is what we typically see in commercial ventures, as being the first one to produce level 5 autonomous vehicles would open up a vast area of potential markets.

One of the projects which focuses on bridging the gap towards autonomous driving via level 3 and level 4 automation is the MEDIATOR project. The goal of the MEDIATOR project is to develop a system which operates in semi-automated vehicles. The system should be able, in real-time, to manage the switching between Human Driver (HD) and Autonomous Driver (AD) depending on which is fittest to drive [19].

In Figure 1.2, a diagram of the MEDIATOR system is shown. The project consists of many components that are developed in parallel, ranging from interpreting the driving context to designing a human-machine interface to give feedback to a driver. This thesis focuses on the core of the system, the control component. Since all other parts of the system are still in development during this thesis project, a simple simulated representation of these factors will be used to develop the control system. This implies that the end product of this project should be seen as a proof of concept of the taken approach, rather than a finalized version.

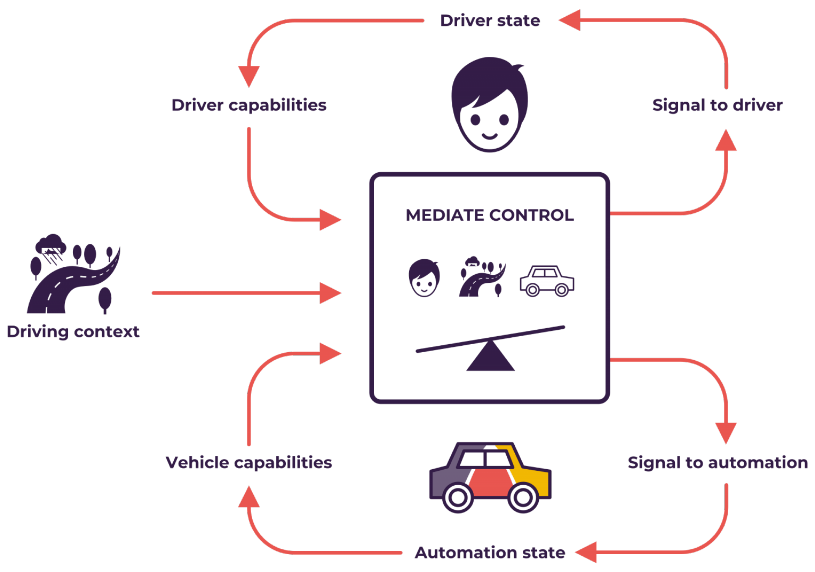


Figure 1.2: The mediator [19]

1.2. The decision logic

The control component of the MEDIATOR project can be conceptualized as a decision logic problem. The decision logic of interest for the MEDIATOR project has the form of a model that, based on the current and expected future context, can find an optimal plan for deciding whether the human or autonomous system has to be given control over the vehicle. To be able to solve this planning problem, the decision logic should have multiple possible actions. While the exact set of actions still needs to be decided on within the MEDIATOR project, it is clear that the decision logic will have control over the current mode of automation. It is also clear that it should have options to influence its context, mostly, the human drive. The latter allows the decision logic to make more sophisticated plans rather than only being subject to external influences.

This thesis aims at finding a feasible model for dealing with the decision logic problem. The problem of interest can be formulated as: find a model that maps the driving context to optimal action to take at the current time, considering a plan towards the future.

1.3. Prior work

Ruf and Stütz [25] described the design of a system for deciding which tasks need to be handled by the pilot of an army helicopter and what tasks can be dealt with by the automated system, given the current context. Based on an estimate of the automation trustworthiness and the current workload as input, it can change the automation level. The researchers successfully implemented a Markov decision process to model this task. The problem they had to solve is very similar to the decision logic problem of the MEDIATOR project, which is works a motivation to use a Markov decision process to model the decision logic of interest to us.

1.4. An MDP approach

A Markov decision process (MDP) [24] is a control process that works in a discrete-time stochastic domain. It is a framework used to model the decision making of an agent in an environment. Together with its generalization, the partially observable Markov decision process (POMDP) [22], MDPs have proven successful in many domains, including planning in healthcare [10], robot navigation [29], and trading [15].

The most basic intuition of an MDP is the one of a robot that needs to find the optimal path to some goal state. The robot can take several actions to get to the goal state, e.g. move east, each of which associated with some uncertainty. The mathematical framework of an MDP allows for multiple ways to find the most effective method, or policy, to reach the goal state.

Using an MDP approach seems to fit well with the task for the decision logic. The process can be described as a system that needs to select an action: either give more control to the autonomous driver (AD) or human driver (HD), based on the current state of the system. An MDP would be able to handle this task by describing the state space as a combination of the driver state, automation state, driving context, and the amount of control currently given to both AD and HD. The action space would then consist of changing the amount of control between vehicle and driver.

1.5. Contributions in this thesis

The research question to be answered is:

Is a Markov decision process a feasible model for the decision logic in a semi-autonomous vehicle, which has the task to mediate control between the human driver and the autonomous system?

The contribution of this thesis is to create a proof of concept of an MDP which implements the decision logic as described in section 1.2. This is achieved by redefining the problem as a mathematical problem, and subsequently, solving it as an MDP model, together with a set of adjustments from literature. The model is implemented and tested in a set of scenarios. For this purpose, a simulator is built in which the model can be tested for desired behavior, and the limits up to which the system keeps working.

Next to feasibility, it is also interesting to see whether other, less sophisticated approaches would work just as well. For this purpose, a heuristic approach is also implemented. This simpler implementation is used to check as to whether the more complex MDP approach adds extra value as a solution.

1.6. Layout of the rest of the thesis

First, the requirements that need to be met to call a model feasible are described in chapter 2. Before we define the model as an MDP, some background information on special cases of MDPs is given in chapter 3. In chapter 4 of this thesis, multiple adjustments to a standard MDP are introduced to deal with the specific challenges that are introduced by the problem definition described in chapter 2.

Next to the model definition, this thesis also contributes by giving an implementation of the model defined, together with a basic simulation to test the capacities of this model. A description of this implementation is given in chapter 5. Additionally, a set of scenarios is defined to showcase the capabilities and boundaries of the model and its implementation.

In chapter 6, the results of these scenarios are described and discussed.

This thesis concludes with suggestions on how the model can be extended or generalized to facilitate the requirements better.

2 Human-Autonomous Driving Decision Logic

To understand the requirements for the decision logic problem, first, the context in which it must operate is defined in section 2.1. Here, the time scale on which the system operates and the components that affect the decision making are described.

While defining the context, we cover the fact that the car in which the decision logic must operate does not yet exist. This has strong implications for the model requirements defined in section 2.3, as the design must take into account that many input and output elements are not final at the time of writing this thesis. The behavioral requirements of the system are described in section 2.2.

2.1. The context in which the system operates

The decision logic is one part of the MEDIATOR project, while in parallel the vehicle itself together with other supporting systems is developed. This means that the vehicle in which the system will operate does not exist yet. However, during the definition of the project, many elements have been decided on, such as that there should be multiple modes of automation. That the system should provide estimates of the environment towards the future. In this thesis, we build on these fixed elements and design for flexibility on the undecided elements. This makes sure the results of this thesis do not become irrelevant if some decisions in the MEDIATOR project come out different than assumed.

2.1.1. Time scale of operations

The first system requirement we have to decide on is the time scale on which we operate. Michon [20] defines three levels of driving tasks: 1) strategic or the planning of the fastest or most comfortable route, 2) maneuvering, which includes changing lane and turning on the blinker, and 3) control, which provides for steering, accelerating or an emergency brake to prevent a crash. The main task of the system is to determine which maneuvering and control tasks should be assigned to the human and which to the vehicle. This means the decision logic has to operate at a time scale bigger than the maneuvering level, for which the time scale needs to be a few seconds, implying the decision logic must operate at a time scale of roughly a couple of seconds.

2.1.2. The elements of the driving context

The second requirement is that the system should be able to deal with the environment. In Figure 1.2, three components are defined as input to the control component, namely driving context, driver capabilities, and vehicle capabilities. Figure 2.1 shows a different take on these three elements, where they are called environmental factors, human fitness factors, and autonomous vehicle fitness factors. The main difference is that the system only directly deals with the last two, as the environmental factors are only indirectly relevant as they can influence the fitness factors. An example is the situation where the vehicle is entirely in control, but in a little while, the weather will become slightly misty, causing the lanes on the road to become undetectable. The system should give control back to the human driver because of the lane visibility, the weather caused the degraded autonomous vehicle performance.

In this thesis, we assume that in the near future it becomes clear which factors should be included in the system and how these can be modeled. However, since at the time of writing this thesis this is not the case, the proposed model must be able to deal with a wide variety of factors. More specifically, what is clear is that the system will be able to measure these factors and represent them as some sort of numeric value. This value is associated with a certain level of uncertainty and with a prediction for future values.

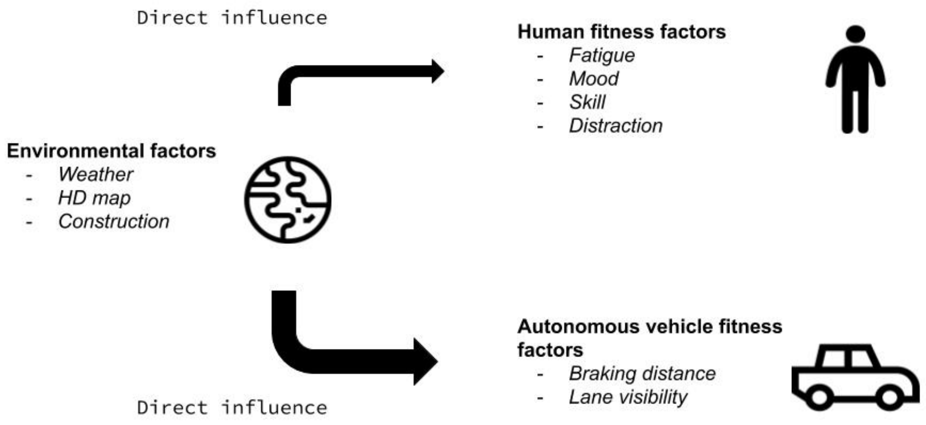


Figure 2.1: A diagram showing how the main input factors can have impact on each other.

2.1.3. The impact on the driving context

Lastly, there is going to be some model of what impact can be applied to these factors. This will be presented as a numeric impact on a factor with some uncertainty. The type of impact that can be applied will be directly related to the actions the system can take. As an example, we may have the option to turn on some sort of alarm which directly increases human attention. However, there is no likely action that impacts on the amount of mist in the near future.

2.2. The behavioral requirements of the system

When designing such a system, it is expected to show a specific type of behavior. At the same time, there are lots of types of behavior the system is not supposed to exhibit. For example, it is very unpleasant for an end-user if every few seconds the control is switched between the human and the autonomous system. But the most important requirement is that the user reaches its destination without any crashes. The goal is thus to optimize the system for comfort (2.2.2), while constraining the system for safety (2.2.1). Therefore, we classify the requirements regarding the behavior of the system in two categories: safety and comfort.

2.2.1. Safety requirement

The most important requirement of the system is that it always acts safely. This means that the system is designed to never take actions that can cause the vehicle to come in a situation where either the human gets an amount of control it is not fit to handle, or the autonomous system has more control than it can handle.

2.2.2. Comfort requirement

The second requirement is comfort. Comfort is hard to define in the context of the decision logic because it can mean something different for each individual. One person may find changes in control very uncomfortable and prefers giving away control only if the driver does not have to take it back for say another hour. But another person may prefer to drive herself as little as possible. To deal with such preference differences, the way the system optimizes for comfort needs to be manageable, either by the car manufacturer or the driver herself on a ride by ride basis.

2.3. Model requirements

When we take the context and behavioral requirements described in the previous section into account, the requirements of the system are now described more formally. In subsections 2.3.1 and 2.3.2, the input and output which the system must be able to handle are given. The remainder of this section describes various additional technical and functional requirements.

2.3.1. Model input requirements

The input is list of factors described by $F = [F_1, F_2 ... F_n]$ which have realizations at each time step. The decision logic has to choose an action. For each factor, the system provides multiple values as input to the decision logic; that is, for factor k at time t it gives as the set of values $F_{k_t} = \{v, \epsilon, V_f, V_{f_\epsilon}, I\}$. These represent:

v	: the expected value for the factor at time t
ϵ	: the possible error of the value v
V_f	: A list of size m of expected future values for the factor
V_{f_ϵ}	: A list of size m of future uncertainty for future values
I	: A set of values that describe the expected impact of taking action a on value v

2.3.2. Model output requirements

The output of the decision logic is defined by an action to take. We have five actions that the decision logic must be able to take, a visual representation of which is given in Figure 2.2.

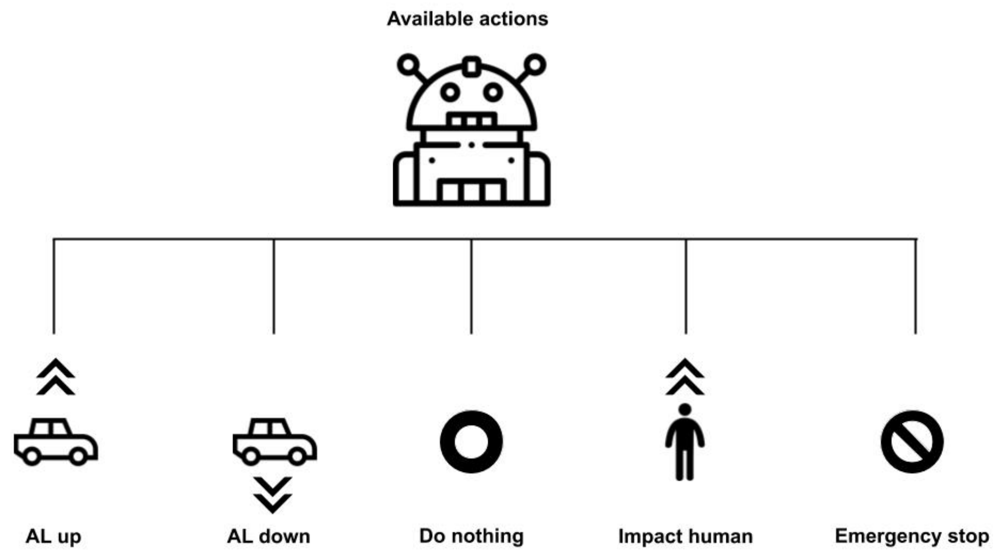


Figure 2.2: The actions that must be available to an agent that operates the decision logic.

The first two actions concern changing the automation level (AL), either by moving control from the driver to the autonomous system or the other way around. While the specific levels of automation implemented in the final version of the system are unknown at the time of writing the thesis, it is clear that the decision logic needs to be able to change between multiple modes of automation.

The second type of action is the option to do nothing. Because the system should decide what to do every time step, the system must also be able to do nothing.

Another type of action is an action that impacts one of the input factors. These actions can come in many different forms, e.g. activating an alarm to get the attention of the driver, suggesting a new route to have better road conditions, or even closing the roof of a convertible because a wet driver might be a worse driver than a dry one. The one thing these actions have in common is that there is always an expected impact on at least one of the factors.

The last type of action is the emergency stop. The most important constraint on the system is that it always has to operate safely. However, the system might come in a situation where both the autonomous system is unable to drive, and the human driver is not fit either. In this case, the system needs to be able to call for an emergency stop. How this looks like from a procedural point of view is not important for the decision logic, it only needs to have the option to call for it.

2.3.3. Predictability and explainability of the model

Since the system has to operate in an environment where mistakes can cause serious injury or even death [27], the entire system must be explainable and predictable. Predictability is required because there is a need for guarantees of safety. A system that is not predictable can impossibly guarantee anything. Even though during rigorous testing, the system always acts as expected, if it is not predictable, we cannot ensure that it will always behave safely.

In 2018 the European Union passed the General Data Protection Regulation, thereby also giving citizens of the European Union the ’right to explanation’ [12]. The ’right to explanation’ means that users have the right to get an explanation about the algorithmic decisions made about them. If the MEDIATOR system decides that a user is not fit to drive anymore and therefore takes over control, this is a decision about the user and must consequently be explainable.

3 Background on MDP and variants

A Markov decision process (MDP) is a general framework for modeling systems in which an agent needs to choose the optimal action to take. The theoretical background of MDPs is given in section 3.1. When solving real-life problems with this basic version of the Markov decision process, some aspects of those problems may be impossible to map. This is the reason why many adaptations to this basic model have been proposed in literature. In the next chapter, a few of these adaptations will be used to map the decision logic to an MDP. In section 3.2, the theoretical background of these adaptations is provided.

3.1. Background on Markov decision process

MDPs are discrete-time stochastic control processes, which try to optimize the actions an agent takes with respect to some reward function. An MDP is defined by a 4-tuple $< S, A, T, R >$, where:

- S is the set of states an agent can be in.
- A is the set of possible actions the agent can take.
- T are the transition probabilities defined by $T(s,a,s') = P(s_{t+1} = s'|s_t = s, a_t = a)$, the probability of being in state s' in $t+1$ given the agent takes action a in state s.
- R is the reward function given by $R(s,a,s') \mapsto r \in \mathbb{R}$, the immediate expected reward gained from moving from state s to s' as a result of action a.

An important aspect of MDPs is the Markov property of state transitions formulated in equation 3.1. This property implies that it is assumed that the probability of a transition from state s to state s' by action a is independent from past actions or visited states of the agent. This property is crucial in order to find an optimal policy. Later in this chapter, some issues regarding this property are discussed when designing the state space.

$$T(s,a,s') = P(s_{t+1} = s'|s_t = s, a_t = a) = P(s_{t+n+1} = s'|s_{t+n} = s, a_{t+n} = a) \tag{3.1}$$

A policy π is defined as $\pi(s) \mapsto a \in A$, an action a to take given a state s. Most often the goal of defining an MDP is to find a policy that maximizes the expected discounted reward, as given in equation 3.2, where we take the expectation with respect to the transition probabilities T. The discount factor γ is a value strictly less than 1 and in practice often close to 1. It discounts a reward which is obtained further in the future, i.e. if t is larger. Its aim is to make sure that the agent prefers immediate reward over reward further in the future, in this manner the model prevents agents from postponing actions indefinitely.

One way to find a policy with a maximum expected discounted reward is via a value function $V_\pi(s)$, which represents the expected discounted reward of a given state s given a policy $\pi(s)$. The equation is given in 3.3. We can also compute a new policy from a value function using equation 3.5, giving a policy maximizing the expected reward. To find a good or even optimal policy, one can start with a random value function and subsequently compute a new value function using the Bellman equation 3.4 [5]. This can be iterated until the $V_{i+1} = V_i$.

$$E[\sum_{t=0}^{\inf} \gamma^t R(s,a,s')] \tag{3.2}$$

$$V_\pi(s) = \sum_{s'} T(s,\pi(s),s')(R(s,\pi(s),s') + \gamma V(s')) \tag{3.3}$$

$$V_{i+1}(s) = \underset{a}{\operatorname{argmax}}\{\sum_{s'} T(s'|a,s)(R(s'|a,s) + \gamma V_i(s'))\} \tag{3.4}$$

$$\pi(s) = \underset{a}{\operatorname{argmax}}\{\sum_{s'} T(s'|a,s)(R(s'|a,s) + \gamma V_i(s'))\} \tag{3.5}$$

3.2. Adaptations to the standard model

There are many reasons to make adaptations to the basic MDP model. In this section, three of those adaptations will be covered. For each of them, after introducing the problem it solves, the adaptation to the model is described.

3.2.1. Constrained MDP

Imagine a situation in which an agent tries to find as much candy in a world as possible. This would be a task that could be well modeled by an MDP. The current state is the location of the agent, the actions are moves in the world, the transition probabilities are the success rates of moving to a new location in the world, and the reward function is dictated by the amount of candy acquired. However, what happens if the agent has a finite amount of fuel? It is not possible to incorporate it in the reward function because getting more candy and running out of fuel are separated processes, which do not cancel each other out in any way, meaning that one cannot increase the amount of fuel by getting more candy.

To solve this issue, the Constrained MDP (CMDP) was created [1]. It is defined as an MDP where each state transition $< s, a, s' >$ has both a reward and a cost related to it as displayed in equation 5.9 and 5.10 respectively. The agent will try to optimize for reward while being on a budget B of cost, implying that if an action might result in a budget overrun, the action becomes impossible. This is behaviour is formulated mathematically as finding a policy which has the maximum expected reward as given in equation 3.2, while keeping to the constraint of having a lower expected cost (equation 3.8) than budget as stated in equation 3.9.

$$R(s,a,s') \mapsto r \in \mathbb{R} \tag{3.6}$$

$$C(s,a,s') \mapsto c \in \mathbb{R} \tag{3.7}$$

$$C_\pi = E[\sum_{t=0}^{\inf} \gamma^t C(s,a,s')] \tag{3.8}$$

$$C_\pi \leq B \tag{3.9}$$

This method has been in many applications, for example in the planning of the building of new housing under a construction budget [30] and a pavement management system implemented in Arizona[11], which tries to minimize the cost of upkeep while having constraints on the minimal performance of the roads was implemented in Arizona.

3.2.2. Non-stationary MDP

Another problem that can be encountered is that of a system which is not stationary. This means that the transition probabilities are not equal at each epoch the system runs. As an example, let us take the world of the previous candy-hungry agent. When we add wind to it as a factor, every few epochs there is a big blow of wind from one direction, making the agent move in the direction it is blowing, no matter what action is taken. This implies the process is not stationary anymore if for the factor is not accounted.

The above problem can be solved by turning the model into a non-Stationary MDP or NS-MDP [13]. The basic idea of an NS-MDP that time is added to the state space. By stating that transitions from a state with time k transition to some state with time equal to $k+1$, one can allow for time dependence in the transition probabilities in the model. However, the state space becomes infinitely large if the time is not bound. This can be solved by creating a forecast horizon [7], t_h, at which we cut off the process. One hard part when using NS-MDPs is to decide about the value of the forecast horizon. The goal is to have it as low as possible to increase solving speed, while still making the optimal decision at $t = 0$ [21].

Tang et al. used an NS-MDP to control the network selection system in new 5F dense network scenarios [28]. They aimed at optimizing the throughput within a diverse network setting, while new users kept entering and leaving the system, making the system non-stationary. They were able to achieve better throughput than other systems have on the same problem.

3.2.3. Options over MDPs

The last adaptation of the basic MDP discussed here is that of options. The need for options arises when actions cannot be executed within a single epoch. An example of such an action, in the context of our candy eating robot scenario, occurs when eating a candy might take longer than taking a step and consists of a series of actions which itself can be optimized. Such actions might be unwrapping the candy, chewing the candy and then swallowing it. Such an action is also called a temporally extended action. We now want our agent, especially with a limit of fuel, to weigh the time it takes to eat a candy and possibly walk past a candy to find more in a later stage, as well as find an optimal policy for the process of eating a candy.

Sutton et al. introduced a framework for such temporally extended actions [26], and this was further developed by McGovern et al. [18]. These authors build upon the concept of Semi MDPs [6], which is an MDP where actions do not operate in discrete time, hence allowing to deal with continuous-time concepts within the MDP framework. However, in [18], the idea of options is introduced. An option is a temporally extended action that consists of a set of primitive actions, from now on just called primitives and denoted by Z. More formally, an option consists of three components, an Input set $I \subseteq S$, describing the states in which it is possible to invoke the option, an option policy $\pi_O(s)$ similar to the policy of an MDP, mapping states to primitives to perform in that state, and a Termination Function $\mathrm{B}(s)$ defining whether an option terminates in state s. Figure 3.1 shows a presentation of the relationship between MDPs, SMDPs, and options over MDPs. As can be seen, the main difference of options over MDPs compared to SMDPs is that when options are used, it is still possible to fully reason about the entire model as if it were an MDP.

To further clarify on the relation between MDPs, SMDPs and options over MDPs we will look back at the candy eating example given at the start of this section. When eating the candy would just be some action that takes longer than other actions the system would best be modeled as a SMDP. However, if eating a candy is defined as a sub-policy consisting of a number of steps, we can reason about it as options over MDPs.

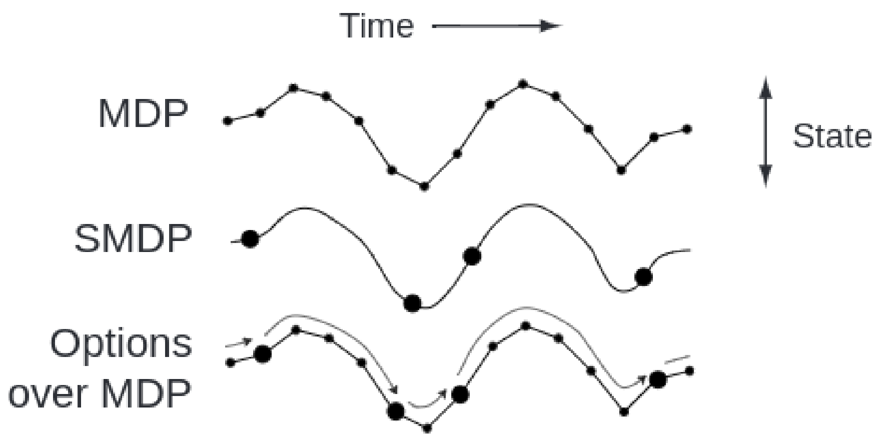


Figure 3.1: A graphical representation of the difference between MDPs, SMDPs and Options over MDPs, [26] [18]

4 The Model Definition

Now a theoretical model of the Markov decision process has been given in the previous chapter, it is possible to map our problem, that of the decision logic onto it. First a naive implementation is proposed, this implementation will show not to be able to meet all requirements. In section 4.2 the challenges that need to be faced in order to meet the requirements are listed. For each challenge, an adaption to the basic MDP from literature is proposed. Finally, a formal mathematical definition of the model with the previously proposed adaptions is given in section 4.3.

After describing the theory of the Markov decision process in the previous chapter, in this chapter we provide details on how it can be used to deal with our decision logic problem. First, a naive implementation is proposed using a basic MDP, which will be shown not to be able to meet all system requirements. Next, in section 4.2, the challenges that need to be faced in order to meet the requirements are listed, and for each challenge an adaption of the basic MDP is proposed. Finally, a formal mathematical definition of the model with the previously proposed adaptations is given in section 4.3.

4.1. A naive MDP approach

We start with the description of a naive implementation of the decision logic as an MPD. In order to formulate a problem as an MPD, the four-tuple $< S, A, T, R >$ needs to be defined. In this section, each of these four elements will be specified, where we start with the state space S, followed by the action space A, the transition probabilities T, and the reward function R.

State space

A naive definition of S is obtained by combining all factors described in section 2.3.1 together with the current state of automation. This implies in a simple version of the model, a state could be described by a tuple of:

< *current-automation-state, fatigue, amount-of-mist, lane-visibility, etc...* >

Since the state space of an MDP must be finite to guarantee convergence [4], all factors should be either discrete or discretized. Each combination of discrete factor values defines a unique state, implying the size of the state space becomes $|S| = \Pi_{f \in F} |f|$, where $|f|$ represents the number of discrete values for the particular factor concerned. Since the number of possible combinations increases exponentially with the number of factors included in the model, it is clear that this approach will become problematic if one wishes to account for more than a few factors. This issue will be addressed in more detail in section 4.2.2.

Action space

The action space can be partially copied from the requirements listed in section 2.3.2, which includes doing nothing, changing the level of automation, and using an emergency brake. Others actions may be added depending on the system requirements the model needs to take into account. An example such an extra action could be attracting the attention of the driver.

Transition probabilities

The transition probabilities, T, can be defined by a set of models that predict the impact of actions on the factors, e.g., the probability of increasing the drivers attention when an alarm goes off. Included in the list of probabilities defining the impact of actions is also the impact of doing nothing, which, for example, includes defining how human fatigue changes over time as a driver drives longer.

Reward function

Lastly, the reward function R can be defined by giving a high score for being or going to states that are safe and comfortable. Simultaneously, negative rewards can be used for changes in the automation state, as a way to lower the number of changes in control between human driver (HD) and autonomous driver (AD). But more importantly, the negative reward of states which are not safe must be large enough so that it is never optimal to go to an unsafe state, even if this results in a very large reward in the future. It should be noted that a basic MDP as described in 3.1 is in fact not suited for our purpose because it optimizes just a single reward function, while we have two functions; i.e., one for comfort and one for safety. Even though the basic MDP may yield an agent able to trade off cost and reward, it cannot be used to deal with the situation where cost and reward cannot be crossed out. This problem will be covered in more detail in subsection 4.2.1, where an implementation of Constrained MDPs is proposed. The resulting adaptation allows an agent to separate cost from the reward function allowing for accumulation of cost while gaining reward.

4.2. Challenges

Multiple parts of the decision logic do not directly translate well to a classic MPD as described earlier in this chapter. In this section, five problems associated with the previously described approach are mentioned. For each of these problems, an adaptation to the basic model from literature is made to solve the issue concerned.

4.2.1. Constraint on safety

The first problem of mapping the decision logic to an MDP originates from the fact that an MDP solely optimizes for reward. Each state transition has an associated reward, positive or negative, related to it via R for which the model will optimize. However, the first requirement of the system is to guarantee safety rather than to optimize for something. One possible way of dealing with this issue is to make the reward associated with not safe states extremely low so that it is never optimal to transition via one of these states. Nevertheless, it remains hard to guarantee an MDP will show the desired behaviour with this trick; that is, it makes the system more complex, meaning the agent may find undesired ways to optimize its reward. An example of this occurs if the system has a negative buffer overflow in its reward and extreme negative reward becomes extremely high [2]. The solution of Constraint MDPs described in section 3.2.1 involves forcing an agent not to go to those states. This is not only simpler, but also better, because it gives guarantees about safety.

By adapting our model in the manner described above for dealing with the required constraints, it becomes straightforward to constrain for safety. One has to make sure that the cost of every transition towards an unsafe state is higher than the total budget of the agent, which is more formally described as follows:

$$\forall s' \in S_{unsafe}, C(< s, a, s' >) > B. \tag{4.1}$$

An additional advantage of this approach is that we can indicate the agent will call for an emergency stop only if all alternative actions become illegal.

4.2.2. Exponentially growing state space

When taking the state space S proposed in section 4.1, the problem arises that factors may change over time and thus cannot be fixed at the current time. However, the largest problem is the growing size of the state space. Because of the fact that the time which is necessary to find an optimal policy grows with the size of S [17] and that the state space grows exponentially with the number of factors, it is obvious that one should try to limit the number of variables defining the state space. On the other hand, to guarantee safety, every aspect that might influence the performance of either the human driver (HD) or autonomous driver (AD) should be taken into account, which causes the number of factors to grow rapidly. Even in the situation in which the number of factors is just 10 and each of these is discretized into 3 categories, there will already be $3^{10} = 59049$ possible states, which is a huge task to solve.

To deal with this problem, here we propose using a simplified version, which involves creating a simultaneous discretization over all factors into a set of critical factors. Since the system should only consider factors that are relevant for the safety of a mode of operation, the state space should only include factors directly associated with a safe decision.

To recap, the system has to plan which level of automation is safe and ideal for the future. It has to consider that a low level of automation at least requires a human that is fit to drive, and that a fit autonomous vehicle is required in higher levels of automation. So simply put, it only needs three state variables: The Automation Level (AL), Human Fitness (HF), and Automation Fitness (AF).

The factors (F) affecting HF, AF, or both HF and AF can be used to compute the relevant metrics, using a function V shown in equations 4.2 and 4.3. That is,

$$V_{hf}(F) = hf \in HF, \tag{4.2}$$

and

$$V_{af}(F) = af \in AF. \tag{4.3}$$

For this purpose, one needs a function V mapping all relevant factors (F) into a single value for HF and AF, respectively. While there are many different ways to solve this problem, here we will use a simple mapping since it does make much sense at this stage to focus too much on the exact impact of a specific factor given that it is still unclear which are the factors of main interest.

The AL is already discrete, as it is bounded by the number of modes a vehicle has. Because for every AL the requirement is to estimate whether the current HF and AF are sufficient to make it a safe option, we can discretize HF and AF in such a way that they directly relate to the Automation Levels. Table 4.1 shows an example of the resulting required fitnesses for the case of 4 ALs. Note that this is only possible with the proposed smart discretization of the AF and HF variables.

AL	0	1	2	3
required HF	≥ 3	≥ 2	≥ 1	≥ 0
required AF	≥ 0	≥ 1	≥ 2	≥ 3

Table 4.1: Required Human Fitness (HF) and Automation Fitness (AF) to safely be in the set Automation Level (AL) in the case of 4 ALs

More formally, the state space is given by

$$S = \{\forall al \in AL, \forall hf \in HF, \forall af \in AF(< al, hf, af >)\}, \tag{4.4}$$

and the number of options for each state equals to the number of automation modes of the system:

$$|HF| = |AF| = |AL| = \textit{number of automation modes}. \tag{4.5}$$

This shows the total size of the state space is equal to the number of automation modes cubed.

4.2.3. Non-stationary

Subsection 4.2.2 introduces a method to drastically reduce the state space to deal with an uncertain, changing, and expanding number of factors. However, while doing so the model will not be stationary anymore, while stationarity is required to be able to solve the MDP. The reason why this occurs is best illustrated with an example. Before the discretization of human fitness (HF), we could assume the probability of a transition between state $A =<$ *human fatigue: .8*$, \ldots >$ and state $B =<$ *human fatigue: .85*$, \ldots >$ to be constant as long as the system decides to do nothing. In contrast, the transition between the discretized states $C =< HF : 2, \ldots >$ and $D =< HF : 3, \ldots >$ is not constant because it depends on the underlying factors, which through V_{hf} determine its fitness.

A possible way out is to turn the model into a Non-stationary MDP as described in section 3.2.2. The Non-stationarity can be dealt with by adding time to the state space and, moreover, only allowing transitions in which the next state is in the next time step. This can be described more formally as follows:

$$S^* = \{\forall al \in AF, \forall hf \in HF, \forall af \in AF, \forall t < t_h \in \mathbb{N}_0 (< al, hf, af, t >)\} \tag{4.6}$$

$$T(s, a, s') = 0 | s'_t - s_t \neq 1. \tag{4.7}$$

Such models can be solved by creating a forecast horizon [7], t_h, for which we solve the MPD and retrieve a short term plan allowing the system to take an action. At the next time step, the model is updated and needs to be solved again. Such an approach is feasible due to the use of the reduced state space introduced in the previous subsection.

A time horizon can be created by adding a sink state s_0 to which all states with $s_t = t_h$ transition to no matter what action, i.e.,

$$\forall a \in A, T(s, a, s' | s_t = t_h, s' = s_0) = 1, \tag{4.8}$$

with

$$S = S^* \cup \{s_0\}. \tag{4.9}$$

The sink state has a reward of 0 and will, moreover, only transition to itself. This guarantees the sink state does not influence on the policy of the system.

The model developed so far is still not stationary since it does not yet account for all actions of the system in the state space. That is, the impact an action has on factors and thus indirectly on the state space is not part of the state space. This issue can be illustrated by a concrete example. Assume the autonomous system is in control and a decrease of human fitness is expected at $t = 5$. Moreover, at $t = 10$ the autonomous system will no longer be fit to drive anymore. Lucky for us there is an action a_w which can be used to increase the value of the underlying factor which in turn will cause an increase in HF. The expectation is now to call this action somewhere in $5 < t < 10$ and subsequently give control to the human. However, due to the way T is defined as a function of the underlying factors, this

$$T(< t : 8, \ldots >, \forall a \in A, < hf : 3, t : 9 >) = \begin{cases} 1 & \textit{iff } a = a_w \textit{ at } 5 < t < 8 \\ 0 & \textit{otherwise} \end{cases} \tag{4.10}$$

will hold, showing the model is not stationary. Equation 4.10 states that at say $t = 0$, a transition to a high $hf : 3$ is certain if action a_w has been taken before that.

However, the resulting non-stationarity can be solved by adding a helper state variable for 'inflicted impact' for human fitness impact (HFI) and autonomous fitness impact (AFI). States with an impact factor can only be reached by actions that influence HF or AF via underlying factors and serve as an equal part of the fitness they complement. When computing rewards or estimating safety, the impact is added to fitness. This way the impact does not change those systems. However, because the impact can only be increased, it essentially serves as a state containing the information on which impactful actions have been taken in the past.

4.2.4. Temporarily extended actions

In a traditional MDP, an action is defined as being taken at a single time point and only influencing a single state transition. However, this does not fit the decision logic. This applies to, for example, the action of giving back control to the HD, which consist of 1) alerting the driver that control is going to be given back, 2) asking the HD if the driver is ready to take over control, and 3) after the HD has answered, giving back control or a failure of the action. This requires clearly more than one time step.

Not only do actions in practice take longer than a single time step, they may also contain complex policies. An example of a situation where this may be the case is in getting the attention of the driver; that is, while the action of activating the HD may start with a simple light turning on, depending on the reaction of the HD, the system should have the option to take a more aggressive action.

Such temporally extended actions can be implemented via options as described in section 3.2.3. For the model, this means that all actions the system can invoke become options; i.e the actions defined in subsection 2.3.2 and used as a naive action set in section 4.1 now become a set of options O. In addition, a set of primitives must be defined which these options policies use. For the current study, the policies and primitives will be kept simple, since it is unclear which factors will have a play in each of the options. In section 5.1, more details about the implementation decisions will be given.

4.2.5. Dealing with uncertainty

The final challenge that needs to be addressed is the uncertainty in the state space. In many real-world settings, such as the problem at hand, the measurement of the current state value is not perfect. For example, human fatigue at any point in time can be estimated based on camera footage from within the vehicle, but this estimate will contain some error.

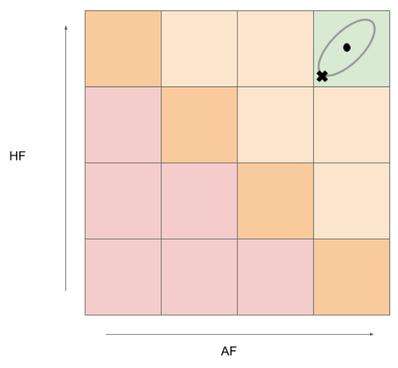
For the sake of explaining how this error will be dealt with, let us focus on the aggregated but non-discretized human fitness (hf) and autonomous fitness (af). This can be done because when aggregating hf and af, the error can be aggregated as well. Figure 4.1 displays the uncertainty as an oval around a specific expected point in the state space. When comparing Figure 4.1.a and Figure 4.1.b, it becomes clear that this error is very important, because the same expected point in the state space can either be completely safe or might be unsafe.

Because the system must constrain for safety, i.e. may never possibly be in an unsafe state, we say that the scenario in Figure 4.1.b is considered an unsafe state. This suggests that we should not create a system working with the expected point in state space, but instead with the worst-case scenario, marked by the ’X’ in the figures. In the following, we will refer to this point as the worst-case point.

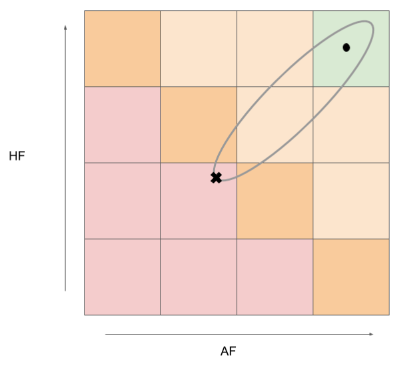
One possible reason to consider the entire region is that the action the system must take is heavily dependent on the underlying cause for the unsafe state. When comparing Figure 4.2.a and 4.3.a, it can be seen that the same worst-case point can have a completely different underlying reason. Figure 4.2.b and 4.3.b show that this different underlying reason is crucial for deciding which action will be effective.

The uncertainty in the state space is dealt with by allowing the transition function to be dependent on both the underlying factors and their uncertainties. This is feasible by defining the transition function as a transition between worst-case points. Because the underlying models work with the expected values and their uncertainties, these values are propagated to the transition function.

It should be noted that our approach differs from the typical way to deal with uncertainty in Markov decision processes (MDP). A more common approach is to turn the MDP into a Partially observable MDP (POMDP) [22]. A POMDP has the advantage that it can reason about the entire current uncertain state, allowing the system to compare a small risk of great loss with a large probability of great success when solving the model. However, what is different in the problem at hand is that the main goal is to be certain of safety. This constraint does not allow the system to trade of a small risk of unsafety with a high probability of great success. There might be situations where the system has the possibility to optimize for comfort within an uncertain fully safe state space. The worst case

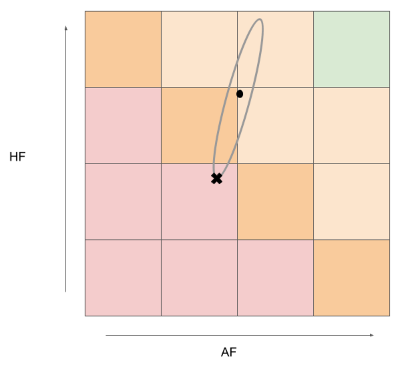


a) A very good scenario, even the uncertainty is fully in the most positive parts of the state space.

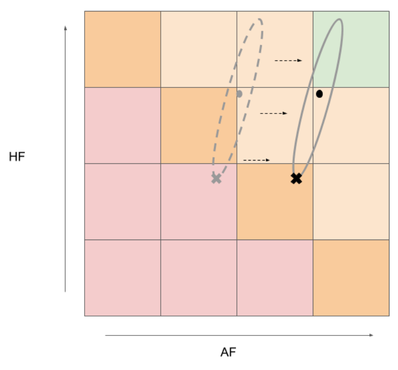


b) Not such a good scenario, even though in reality the state is positive, due to the uncertainty in both autonomous fitness and human fitness the worst-case is unsafe.

Figure 4.1: A possible state in 2d state space, with on one axis human fitness (HF) and on the other axis autonomous fitness (AF). The point represents the expected state, the circle the area of uncertainty, and the cross the worst-case state. Green states are safe, red states are unsafe.

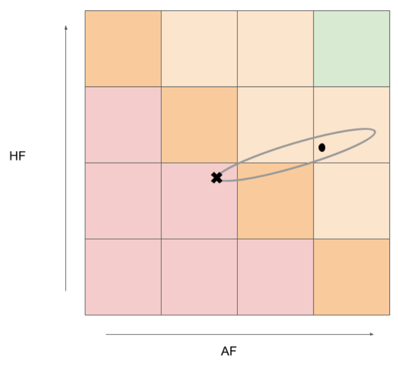


a) An unsafe state, due to low AF and a high uncertainty in HF.

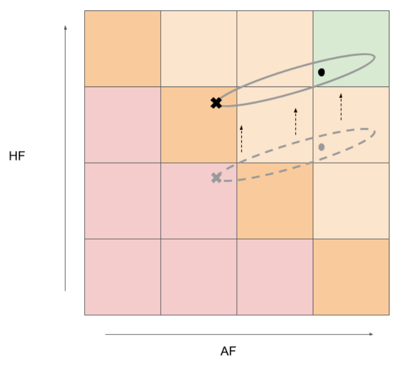


b) showing that an action that increases AF would result in a safe state.

Figure 4.2: A possible state in 2d state space, with on one axis human fitness (HF) and on the other axis autonomous fitness (AF). The point represents the expected state, the circle the area of uncertainty, and the cross the worst-case state. Green states are safe, red states are unsafe.



a) An unsafe state, due to low HF and a high uncertainty in AF.



b) showing that an action that increases AF would result in a safe state.

Figure 4.3: A possible state in 2d state space, with on one axis human fitness (HF) and on the other axis autonomous fitness (AF). The point represents the expected state, the circle the area of uncertainty, and the cross the worst-case state. Green states are safe, red states are unsafe.

approach will not be able to utilize these situations, which is a limitation of the approach proposed.

In this thesis, we will only use the worst-case points method. The way the uncertainty is computed for each included factor is part of the modeling task.

4.3. Formal definition of the model

A formal definition of all model components defined in section 3.1 will be given. Due to the various proposed adjustments, several additional elements are now part of the model; namely, factor to state mapping functions $V_{af/hf}$, a cost function C together with a budget B, and options and primitives replacing the action space A.

The model is defined by a tuple $< S, F, V, Z, O, T, R, C, B >$:

- S is the set of as states defined in equation 4.9.
- F is the set of factors defined in 2.3.1, one can argue that this is not part of the model, but since other elements of the model depend on it, it is included. It should be noted that the impact described in 2.3.1 depends on the primitives Z and not on the options O
- V is a set of functions mapping F to variables in S, namely Autonomous Fitness and Human Fitness.
- Z is the set of primitive actions the system can take.
- O is the set of options each defined by tuples $< I, \pi_o, \mathrm{B} >$ where:
    - $I \subseteq S$ the set of states in which the option can be invoked.
    - π_o is a function which maps $s \in S \mapsto a \in Z$, describing the primitive action to take in a state
    - B is a function which maps $(s \in S, a \in Z, s' \in S) \mapsto 0, 1$, representing a Boolean value whether a state transition from state s to s' as a consequence of primitive a terminates an option.
    - $k \in \mathbb{N}$ defining the maximum number of primitive actions an option can perform before it fails.
- T is the transition function, T can be computed through F since each factor has an expected future value and an expected impact of a primitive a, hence it is possible to define T as $T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$.
- R is the reward function $R(s, a, s') \mapsto r \in \mathbb{R}$, this is the function which the model will attempt to optimize for.
- C is the cost function $C(s, a, s') \mapsto r \in \mathbb{R}$, this is used to constraint the model on certain actions when the cost of a path becomes bigger than B
- $B \in \mathbb{R}$ is the budget of the model, when an action overruns the budget the action becomes impossible to take.

In chapter 5, a implementation of this model will be given to show that it is capable of meeting the requirements. It should be noted that depending on the complexity of the factors and the primitives, this model can easily be made much more complex.

5
Experimental Setup

The model described in the previous chapters was validated using a set of experiments, details of which are presented in this chapter. Section 5.2 provides details on the solver used to find an optimal action. Because it is important to check whether the added complexity associated with the Markov decision process (MDP) is really needed, we compare its performance with a simpler approach based on a set of heuristics. The algorithm and the expected behavior of this simpler approach are described in section 5.3. A flexible simulator that was built which allows running scenarios. An explanation of the workings of this simulator is given in 5.4.

The current chapter ends with a listing of the experiments we designed to validate the feasibility of the MDP approach and to compare its performance with the heuristic approach. For each experiment, a hypothesis formulated stating the expected behavior of the MPD approach.

5.1. Implementation of the model

Some model elements are dynamic over the course of the current experiments, while others are static for each experiment. Table 5.1 list all model elements and indicates whether they are dynamic or static. Below we describe the implementation details of these elements, where for the dynamic elements it is explained how these are varied across experiments.

Model Element	**Static or Dynamic**
S: state space	*Static*
F: factors	*Dynamic*
V: mapping functions	*Dynamic*
Z: primitives	*Dynamic*
O: options	*Dynamic*
T: transition functions	*Static*
R: reward function	*Dynamic*
C: cost function	*Static*
B: budget	*Static*

Table 5.1: A listing of which elements of the model are dynamic per experiment and which are static for each experiment.

State space S

The state space is a static model element which, as defined by equation 4.9, consists of 4 automation states, and thus 4 human fitness and automation fitness levels. In contrast with the formulation of the previous chapter, for these experiments, human fitness impact (HFI) is implemented, while autonomous fitness impact (AFI) is not. The reason for this choice is that all experiments concerning the impact of actions are related to human fitness, making the autonomous fitness impact redundant. The time horizon t_h is set to be 20. The exact time scale of each time step is not relevant for this experiment. The only thing that matters is that 20 time steps will be sufficient to make all maneuvers for which we want to test the two approaches.

Factors F

The set of factors will be defined per scenario. Table 5.2 lists the variables that need to be set for each factor. The remaining of this section is used to describe these elements.

The most important element of a factor is its value function. The *Value* function 5.1 defines the worst-case expected value v at time t of factor f:

$$v_{f_t}(v_{f_{i<t}}, t, F_t) \mapsto \mathbb{R} \tag{5.1}$$

It takes as input the history of the factor $v_{f_{i<t}}$, the current time t, and the values of other factors F_t by which it is affected, and then returns a value in $\mathbb{R}$. This definition allows the system to work with very complex submodels. It should be noted that because a factor can depend on other factors, it is possible to create a circular dependency between factors, which is not allowed by the system. At the start of each scenario the values for the history $v_{f_{i<t}}$ for each factor will be empty, and *Start Value* is used as a substitute for the history.

The current implementation of the system does not account for measurement or estimation errors in the factors. Moreover, our implementation assumes all future estimates are the worst-case estimates, as was described in section 4.2.5.

Field	**Description**
Name	The name of the factor
Value	A function as in equation 5.1
Start Value	The value at the start of the simulation
Impacts	The list of actions which have an impact on the factor and the value of the impact

Table 5.2: A list of the elements required to define a factor

Mapping functions V

While the functions mapping factors F to state variables human fitness (HF) and autonomous fitness (AF) could be varied per scenario, here a fixed mapping function will be used. It is based on creating two factors, $f_{AF} \in [0, 1]$ and $f_{HF} \in [0, 1]$, which depend on other factors in the system. Step functions are then used to map these factors to the discrete HF and AF. These step functions are displayed in equation 5.2 and 5.3, respectively:

$$af = \begin{cases} 0 & f_{AF} \leq .5 \\ 1 & .5 < f_{AF} \leq .7 \\ 2 & .7 < f_{AF} \leq .9 \\ 3 & > .9 \end{cases} \tag{5.2}$$

and

$$hf = \begin{cases} 0 & f_{HF} \leq .1 \\ 1 & .1 < f_{HF} \leq .2 \\ 2 & .6 < f_{HF} \leq .8 \\ 3 & > .8 \end{cases} \tag{5.3}$$

The $Value$ function of f_{AF} and $f_H F$ changes per scenario to test for desired behavior.

Many alternative implementations of these mapping functions possible. For example, in a real-life setting, it may be more realistic to let the discrete fitness level vary per underlying factor of the system. However, for testing the feasibility of the model, this simple step function implementation suffices.

Transition function T

Even though the value of the transition function depends heavily on the definition of the (dynamic) factors, the transition function itself is static because the way it is computed is the same for each experiment. The transition function maps state-action pairs to a probability of being in a new state. Since the factors can be used to create expected values towards the future, the factor value expectations can be used to compute the transition probabilities towards the future. And, because the factors contain the impact of actions, we can include that impact in future expected values when an action has been taken. Equations 5.4 to 5.8 describe this process:

$$T(< hf, hfi, af, al, t >, a) =< hf', hfi', af', al', t+1 > \tag{5.4}$$

$$F' = E[F_{t+1}|a] \tag{5.5}$$

$$hfi' = V_{hfi}(F') \tag{5.6}$$

$$hf' = V_{hf}(F') \tag{5.7}$$

$$af' = V_{af}(F') \tag{5.8}$$

Switching automation levels is not part of this definition via the factors because it is defined as a special action case.

Equation 5.4 describes transitions always go from a state with time value t to a state with time value $t+1$. If $t+1$ equals the time horizon, then all values will transition to the sink state as described in equation 4.8. Equation 5.5 describes that the new factor values are dependent on the action taken. Subsequently, these new factor values are used to compute the new state variables hf', hfi', af', al', as is shown in equations 5.6, 5.7, and 5.8. With this full set of equations, one can compute the transition probabilities.

Primitives Z

In our implementation, we use the minimal set of primitive actions necessary to show the feasibility of the model. The set concerned is listed in Table 5.3. The first primitive, *do_nothing*, is a passive action in which the system does not interfere with the driver or the automation level (AL). The actions *al_up* and *al_down* change the AL, increasing and decreasing, respectively, with a probability which may vary per scenario. The last action, *hf_up*, has a positive impact on the current human fitness. The exact value of the impact and how it affects human fitness changes per scenario.

Primitive	**Description**
do_nothing	Nothing happens
al_up	An increase the automation level (AL)
al_down	A decrease the automation level (AL)
hf_up	An attempt to increase human fitness is made

Table 5.3: A list of all primitive actions

Options O

Similar to the implementation of the primitives, the options also have the minimal implementation for the same reason. Table 5.4 lists the options implemented together with the relevant details. Here, k_{up} and k_{down} are the maximum number of steps the options can take. Since the options policy is simply defined as calling a primitive in every situation, this can be viewed as the maximum number of attempts an agent can take for changing the automation level.

Name	**Invoke subset**	**Policy**	**Termination Condition**	k
Passive	$I = S$	$\forall s \in S(s, do\_nothing)$	$B(s,a,s') = 1$	1
Upgrade AL	$I = \forall s \in S \mid s_{al} < 3$	$\forall s \in S(s, al\_up)$	$B(s,a,s') = \begin{cases} 1 & s'_{al} > s_{al} \\ 0 & otherwise \end{cases}$	k_{up}
Downgrade AL	$I = \forall s \in S \mid s_{al} > 0$	$\forall s \in S(s, al\_down)$	$B(s,a,s') = \begin{cases} 1 & s'_{al} < s_{al} \\ 0 & otherwise \end{cases}$	k_{down}
Wake Up	$I = \forall s \in S \mid s_{hf} < 3$	$\forall s \in S(s, hf\_up)$	$B(s,a,s') = \begin{cases} 1 & s'_{hf} > s_{hf} \\ 0 & otherwise \end{cases}$	1

Table 5.4: A list of all options

Reward function R

One of the system requirements is optimization flexibility depending on the users. To achieve this goal, multiple reward functions are implemented, each of which has a different interpretation of comfort. Consequently, the system will optimize for the interpretation of comfort associated with the chosen reward function. Each reward function consists of three components, as is shown in the following equation:

$$R(s, as') = R_s(s') + R_{al}(a) + R_{wake\_up}(a). \tag{5.9}$$

These concern: 1) a state component, which gives reward for the automation level of the resulting state after an epoch. 2) an automation level change component, which gives a penalty for changing the automation level at any point in time, and lastly, 3) the wake-up component, which gives a penalty for waking up the user. Changing the values of these components results in a system which optimizes differently for comfort. Table 5.5 shows the three implemented functions together with their intended goal.

	R1 (max_automation)	**R2 (min_automation)**	**R3 (min_transitions)**
$s_a l = 0$	80	110	90
$s_a l = 1$	90	100	90
$s_a l = 2$	100	90	90
$s_a l = 3$	110	80	90
$a =' al'_{down} \vert\vert ' al'_{up}$	-10	-10	-100
$a =' wake_u p'$	-10	-10	-10

Table 5.5: A List of the implemented reward functions

Cost C and budget B

Cost and budget are used to constrain the system from transitioning to unsafe states. When the cost of transitioning to an unsafe state is higher than the budget, this constraint is applied. By defining the budget as $x \in \mathbb{R}^+$, we can define the cost function as follows:

$$C(s,a,s') = \begin{cases} x + \delta & s' \in S_{unsafe} \\ 0 & otherwise \end{cases} \tag{5.10}$$

where $\delta \in \mathbb{R}^+$ represents some arbitrary small positive number. This implementation guarantees costs of transitions towards unsafe states to be higher than the budget, making them unreachable. The set S_{unsafe} used in equation 5.10 is defined in as follows:

$$S_{unsafe} = \{\forall s \in S \mid s_{af} \geq s_{al} \& s_{hf} \geq (|al| - s_{al})\}, \tag{5.11}$$

yielding a more formal definition of the phenomenon shown in Table 4.1.

5.2. MDP solver

Because the focus of the current study is on the model feasibility rather than model solution speed at each time step, a simple solver is used to solve the model, namely value iteration [5]. One

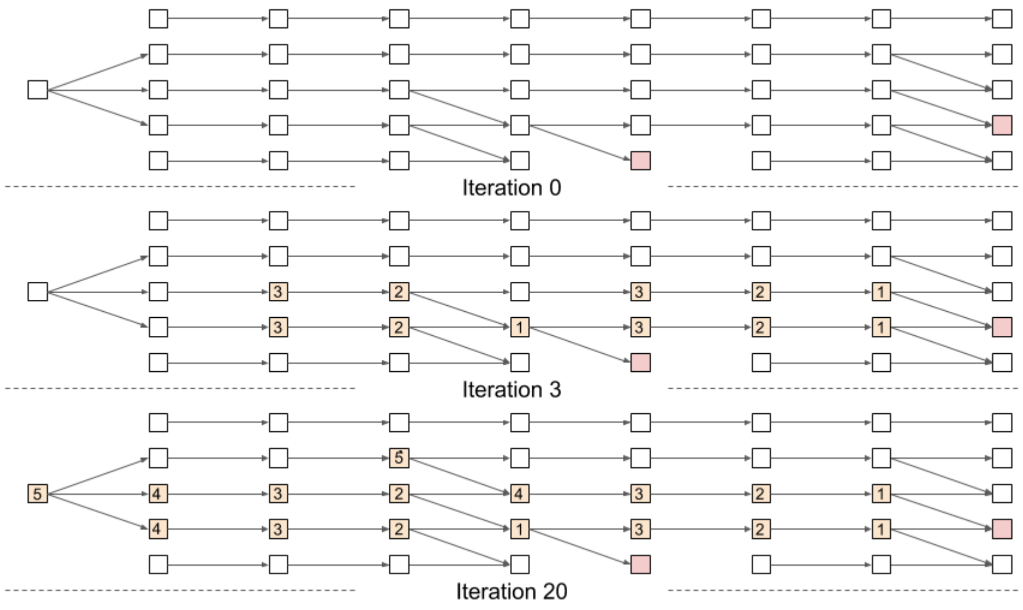


Figure 5.1: A diagram showing three point within a value iteration process. The diagram shows how an inevitable unsafe state (red) propagates through the state space, given the time to perform an emergency stop is set to be 5 time steps.

adjustment had to be made to the standard version of value iteration due to the safety constraint. It has been stated that when the system calls for an emergency stop, it must have enough time in a safe state to perform the emergency stop. For the experiments that are run, this minimal time is set at 5 time steps. When performing value iteration, not only will the current value of the states be tracked, but also how many steps from potential unsafety the optimal action of a state is. When this is less than the required time to perform an emergency stop, the value keeps getting propagated, a process which is displayed in Figure 5.1. When the action at the current state of the agent is less than the required amount of time steps from possible unsafety, the emergency stop is called. For the value iteration, the discount factor was set at .97 and the maximum difference of the value function between iterations was set at 1.

5.3. Heuristic approach

While our main goal is to show that an MDP is a feasible model for the decision logic problem, we also wish to make sure that it does not overcomplicate the problem. For comparison, we therefore also implemented a simpler approach, which is based on heuristics (and referred to as the heuristic approach). In all our experiments we use both the MDP approach and the heuristic approach, were the results of the heuristic approach may illustrate why the more complex planning of the MDP is needed.

The heuristic approach is a very simple algorithm. While it has access to the same data as the MPD model, it uses a rule-based system to decide what level of automation to operate in. The discretization used by the MDP is used to determine the safety of states.

Listing 5.1 describes the heuristic algorithm to determine the next action, which has a preference for an as high as possible automation level, as long as it is safe. In line 2, the function 'nextState' is used for the first time, which returns the expected new simulation state given the current simulation state and an action that will be taken. The order in which the possible actions are checked for a safe outcome determines the behavior of the agent. In line 11, the last resort of waking up the human driver is checked before calling for an emergency stop.

Listing 5.1: A stay safe and automate as much as possible heuristic

```
1 function nextAction(simState):
```

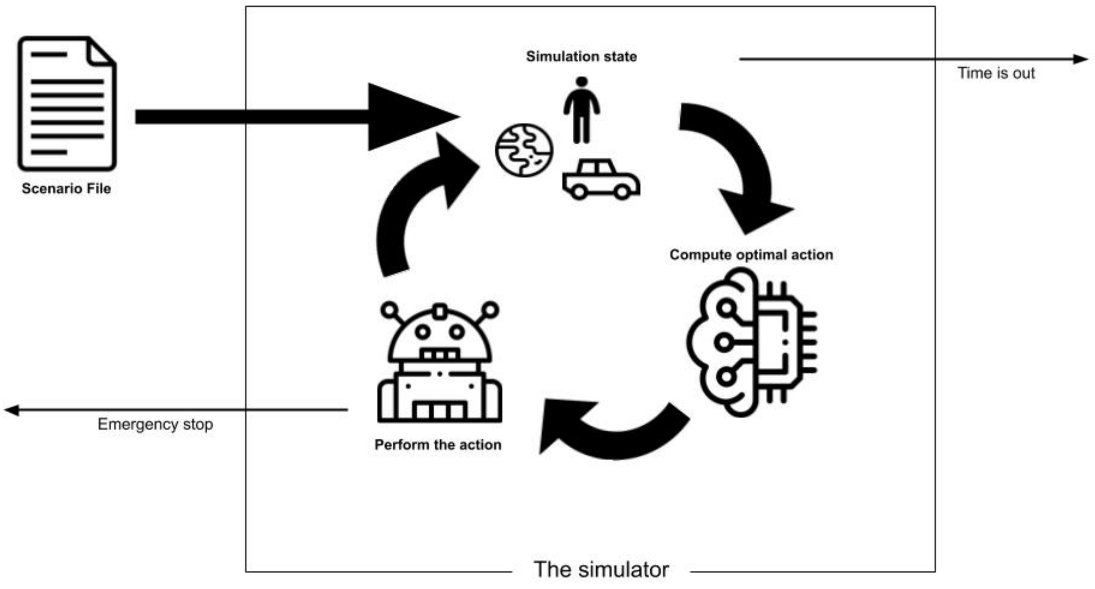


Figure 5.2: A diagram showing what one simulation run looks like.

```
    if (nextState(simState, 'upgrade').isSafe()) {
        return 'upgrade'
    }
    if (nextState(simState, 'passive').isSafe()) {
        return 'passive'
    }
    if (nextState(simState, 'downgrade').isSafe()) {
        return 'downgrade'
    }
    if (nextState(simState, 'wake_up').isSafe()) {
        return 'wake_up'
    }
    else return 'EMERGENCY_STOP'
```

5.4. The simulator

One of the contributions of this thesis is that a simulator is created in which the models, either an MDP approach or the heuristic approach, can operate and be tested. It should be able to simulate the input requirements as defined in chapter 2 and handle the output of the system appropriately. Figure 5.2 gives a visual representation of a simulation run. It starts with the definition of a scenario in the form of a scenario file. Then a simulation loop starts, where at each time step, an action needs to be computed based on the current simulation state, after which the action is performed, resulting in a new simulation state. This process continues until either an emergency stop is called or the predefined time of the simulation is reached.

5.4.1. Scenario definition

A scenario file contains all information the simulator requires to run an entire simulation. It is defined as a JSON file. The schema of such a JSON file is given in Listing 5.2. The schema begins with the more basic features of a scenario, the start automation level (*startLoa*), the total time the simulation should run (*totalT*), and the time it takes to perform an emergency stop (*timeOfES*).

The array of factors is the more complex element of the schema. It defines all factors of the scenario to be used. Each factor should have a *name*, *type*, *value*, and *dependence*, while a *start_value* is optional. If the value is a number, the factor will behave as a constant value. However, when the factor is a string and its type is 'modeled', the value is evaluated as a function, which can be any JavaScript code that evaluates to a number. Because JavaScript code can directly be used while defining a scenario, it is possible to create complex behavior.

Three additional variables the function may access are defined in 5.2. By using the "Current time" variable, it is possible to create timed events in a scenario. This is useful when creating scenarios where a user gets a call, for example, that after 20 time steps the road conditions will become terrible. The "earlier values" can be used to define a factor recursively. A factor that can benefit from this is, for example, fatigue. The fatigue submodel may be a function that slowly increases. The last variable, "other factor", can be used to combine factors. This is useful for defining how the final autonomous fitness is computed, e.g. $1 - \${fatigue} - \${visibility}$.

Variable	**Name**	**Explanation**
${_t}	Current time	The current time of the simulation, starts at 0 at the start of the simulation.
${-x}	Earlier value	Previous value x time steps back, so ${-1} represents the previous value and ${-2} is two time steps back, if not defined yet the *start_value* is used.
${name}	Other factor	References the value of another factor at the current time only available if the dependence is set to *endogenous*.

The factor definitions are followed by the definitions of the automation level changes. For each of the two actions, the probability of successfully changing the automation level and the maximum number of attempts for one option are defined.

The last element of the scenario file is the array of action effects, i.e., the impact an action has on a given factor. The impact consists of the primitive that invokes the action, a name of impact, and the factor it influences, as well as the actual difference made on the factor defined using a number in $[-1, 1]$. An action effect only has a durable impact on factors which are a function of their own history, otherwise the impact is only effective for a single time step.

Listing 5.2: A schema for defining Scenarios in JSON format

```
{
    description: string,
    startLoa: number, default: 0,
    totatT: number,
    timeOfES: number, default: 5,
    factors: Array<Factor>,
        Factor: {
            name: string,
            type: 'static' | 'modeled',
            dependence: 'exogenous' | 'endogenous',
            value: number | string,
            start_value: number
        },
    alActionParams: {
        up: {
            successOfPrimitive: number, default: 1,
            numberOfAttempts: number, default: 1,
        },
        down: {
            successOfPrimitive: number, default: 1,
            numberOfAttempts: number, default: 1,
        },
    },
    actionEffects: Array<ActionEffect>,
        ActionEffect: {
            primitive_name: string,
            name: string,
            on_factor: string,
            diff: number,
        }
}
```

5.4.2. The simulation loop

The simulator runs a single simulation loop per time step until the end of the scenario is reached. In Listing 5.3, pseudocode for the full simulation is given. Each simulation starts with an initialization of the simulation state using the information supplied in the scenario file. The simulation state holds all relevant information for the simulation, which includes the static and dynamic model elements. Next, in line 4, the simulation loop starts. The remaining of this section is used to describe the steps in the simulation loop.

The simulation loop starts by checking whether it is performing an option (line 6). If this is the case, the policy option policy is continued. Note that options can take multiple time steps to finish, while the simulation loop runs only one time step. This is the reason why it is required to check this. If we continue executing an option, the rest of the loop can be skipped.

Next, in line 10, the current state is computed by combining the factors and the current automation level, which are both part of the simulation state. The computation is done according to equations 5.2 and 5.3.

After computing the current state, the available actions are computed in line 11, which are needed to be able to obtain the transition probabilities. The transition probabilities are obtained using equations 5.4 to 5.8, which requires the simulation to give future expected values.

The following phase in the loop has the goal of finding the optimal action the agent can take. For this purpose, a new solver is initiated for either the heuristic or the Markov decision process approach. Subsequently, the solver is used to find the next action.

Each simulation loop ends with performing the action which is found. This yields an update of the environment according to the underlying submodels, resulting in a new simulation state. After the simulation state is updated, the simulation loop repeats. The simulation ends when either the total time defined for the scenario is reached or the action of the agent is the emergency stop.

Listing 5.3: The simulation loop

```
1  function runSimulation(scenarioFile):
2      simState = createSimState(scenarioFile)
3
4      while simState.currentT < simState.totalT:
5
6          if simState.isPerformingOption:
7              simState.continueOption()
8              break
9
10         currentState = computeCurrentState(simState)
11         actions = computeActions(simState)
12
13         solver = new Solver(simState.stateSpace, options, currentState)
14         nextOption = solver.getOptimalAction()
15
16         simState.performAction(nextOption)
```

Variable	Explanation
k_{up}	The amount of attempts an agent can take to increase automation level while executing a single options
P(up)	The transition probability of successfully increasing automation level
k_{down}	The amount of attempts an agent can take to decrease automation level while executing a single options
P(down)	The transition probability of successfully decrease automation level

Table 5.6: Listing of parameters for the actions with respect to changing automation.

5.5. List of experiments

Eight experiments are created to test whether the Markov decision process (MDP) approach is a feasible model. The eight experiments make use of six different scenarios. This section presents these experiments and scenarios, as well discusses the desired behavior of the decision logic. We also hypothesize how we expect the MDP approach to perform.

A secondary goal of the experiments is to show how the system operates under different levels of uncertainty with respect to actions. For this purpose, for a subset of the experiments, we varied four of the variables influencing changes in automation state. These four variables are displayed in Table 5.6.

In Table 5.7, we list the experiments that are going to be used, together with a few basic features of each experiment. For each scenario used, two plots will display the human fitness (HF) and autonomous fitness (AF) during the simulation. For HF, we display the values given the agent does not interfere using an action that increases the HF. Appendix A lists the scenario files used.

# Experiment	Scenario	Reward system R
1	Base case, Figure 5.3	*max_automation*
2	Simple Scenario, 5.4	*max_automation*
3	Simple Scenario, 5.4	*min_automation*
4	Simple Scenario, 5.4	*min_transitions*
5	Different types of impact, 5.5	*max_automation*
6	Solvable by cumulative impact, 5.7	*max_automation*
7	One optional increase in automation , 5.7	*max_automation*
8	Room for choice, 5.7	*ALL*

Table 5.7: A list of experiments with its basic features.

5.5.1. Test base requirements

The first experiment uses the most extended scenario, which we refer to as the "Base case". It is a scenario that requires the system to exhibit all regular behavior the decision logic should be able to deal with, except for an emergency stop. Bram Bakker provided this scenario as a test case for the MEDIATOR project. However, it can also serve as a general showcase of the capabilities of the decision logic.

In Figure 5.3, the scenario is displayed as a timeline. It is important to note that the human fitness (HF) and autonomous fitness (AF) are derived from the underlying factors. However, HF and AF are the best indicators of the expected behavior from the system. Table 5.8 lists the actions the MDP system is expected to make.

The "Base case" scenario starts with two periods in which the system must increase the automation level to be able to deal with a decrease in human fitness (HF). This happens around $T = 9$ and $T = 24$. We expect the MDP to perform these actions, because it can increase the automation level (AL) and decrease it again a little while later, that is, when AF does not longer allow for the higher automation levels.

Around $T = 42$, again an increase in AL is expected due to an increase in AF. However, this time HF is not expected to be back at the desired level before AF decreases. This could result in a

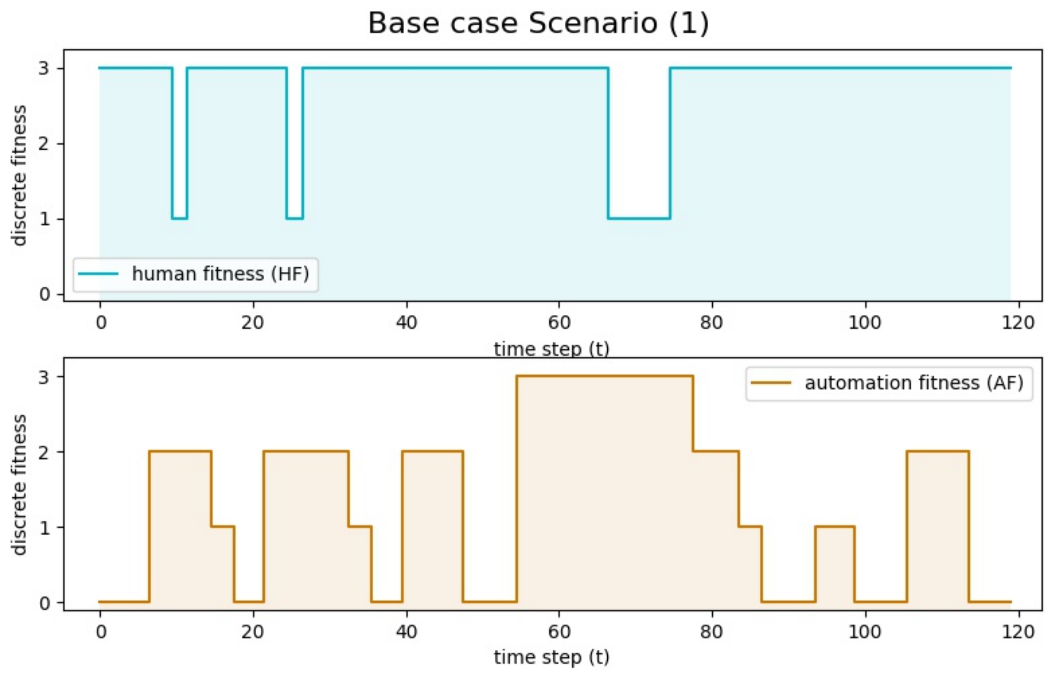


Figure 5.3: A visual representation showing the human fitness (HF) and autonomous fitness (AF) when no actions are performed during the time of the "Base case".

Time (t)	**Action**
9	Increase automation level (AL) to allow lack of concentration of driver
42	Increase automation level (AL) to allow lack of concentration of driver
45	Wake up user before autonomous fitness drops
57	Go up to max automation due to long stroke of autonomous confidence
96	Comfort increase automation level (AL)
108	Comfort increase automation level (AL)

Table 5.8: Subset of actions expected from a successful implementation of the decision logic for experiment 1, the Base case

situation where the system comes in an unsafe state. Therefore, we expect that the system takes action to increase HF, and get it at a level where it is safe to reduce the automation level.

After these first hurdles, the scenario goes towards a longer time period with high AF. This could, for example, be a long stroke of highway within a ride, where it is possible to go towards the highest automation level in the system. The system is expected to go to the max automation level and decrease it again as AF starts decreasing.

The last part of the scenario describes two situations where the increase in automation level is completely by preference. First, at $T = 96$, the automation fitness increases to 1, and second, around $T = 108$, the automation level increases to 2. The system does not need to increase automation level because the human is perfectly fit to drive the entire time. However, due to the setup of the reward system, we expect it will increase AL.

All hypotheses formulated in this section concern the situation where the success of an automation change action is guaranteed. Next to the baseline experiment, for the first scenario a set of variations regarding the automation change actions are tested. Table 5.9, shows the parameter options tested for each variable of interest. A full factorial design obtained by crossing these options, yields 81 experiments in addition to the earlier mentioned baseline.

5.5.2. Test preference options

As stated in the requirements, the system must be adjustable for preference in comfort. Experiments 2 to 5, are designed to test whether this requirement is met. When the MDP approach is

Variable	**Variations**
k_{up}	[1, 3, 5]
P(up)	[.4, .6, .8]
k_{down}	[1, 3, 5]
P(down)	[.4, .6, .8]

Table 5.9: Variations of parameters for scenario 1. A cross product of all variations will be tested.

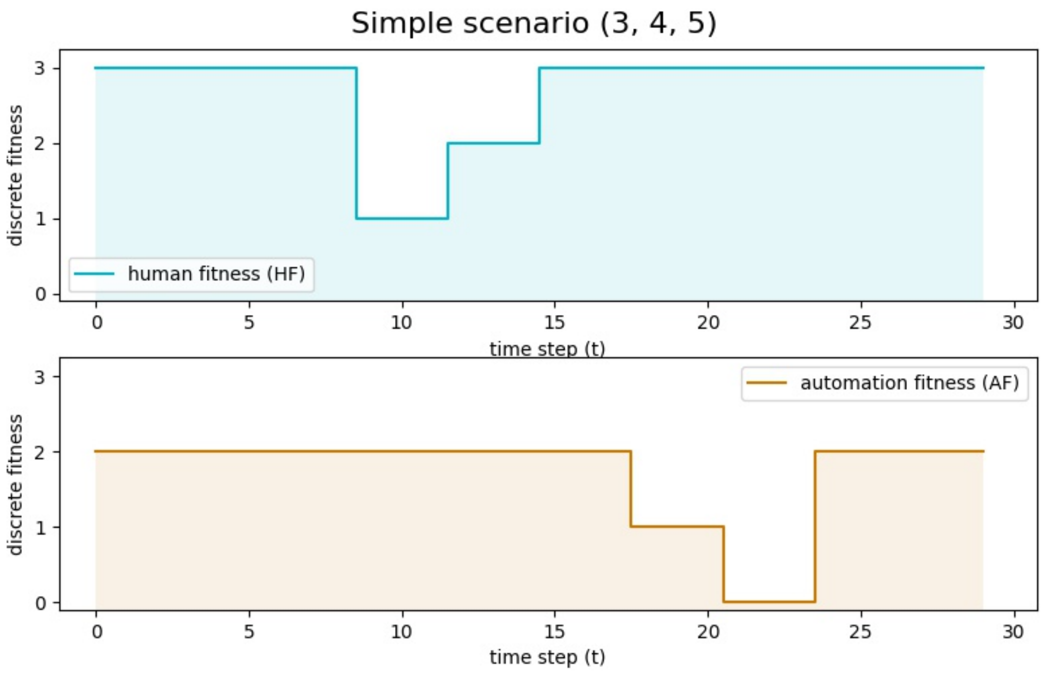


Figure 5.4: A visual representation showing the human fitness (HF) and autonomous fitness (AF) when no actions are performed during the time of the Simple scenario

used, the preference for comfort is adjustable by changing the reward function. Three reward functions have been defined in section 5.1, each named by its intended need. Each experiment uses the same scenario, the "Simple scenario", to test the impact of changing the reward system. The HF and AF of this scenario are displayed in Figure 5.4. The heuristic approach is not adjustable for preference. Therefore, it is less interesting how it behaves in these experiments.

The first two reward functions tested are 'max_automation' and 'min_automation'. These represent a reward system where the goal is to either maximize or minimize the automation level (AL). This means that in the first experiment, it will operate at an AL of 2 until the decrease of the AF, when it will have to decrease the automation in order to stay in a safe state. The opposite is expected from the experiment where the 'min_automation' reward system is used in the same scenario. Namely, the system will only increase the AL when the HF is about to decrease.

During the last experiment using the "Simple scenario" the systems goal is to minimize the number of transitions between automation levels. If the MDP approach works properly, it should change the automation level only 3 times. It should increase automation level, in the situation where keeping the human in charge causes a dangerous situation at $T = 9$. Next, when keeping the autonomous system in charge becomes unsafe around $T = 15$, it should change again. The system is expected to step from 2 to 1 and back to 0, because changing from 2 to 0 in a single step is not possible. Otherwise, we would have expected the system to go to directly to $AL = 0$.

5.5.3. Test dealing with impact

The goal of experiment 5 and 6 is to test how well the system is able to use its impact on factors it can influence. As described earlier, only the 'wake_up' action can be set to have an impact on the

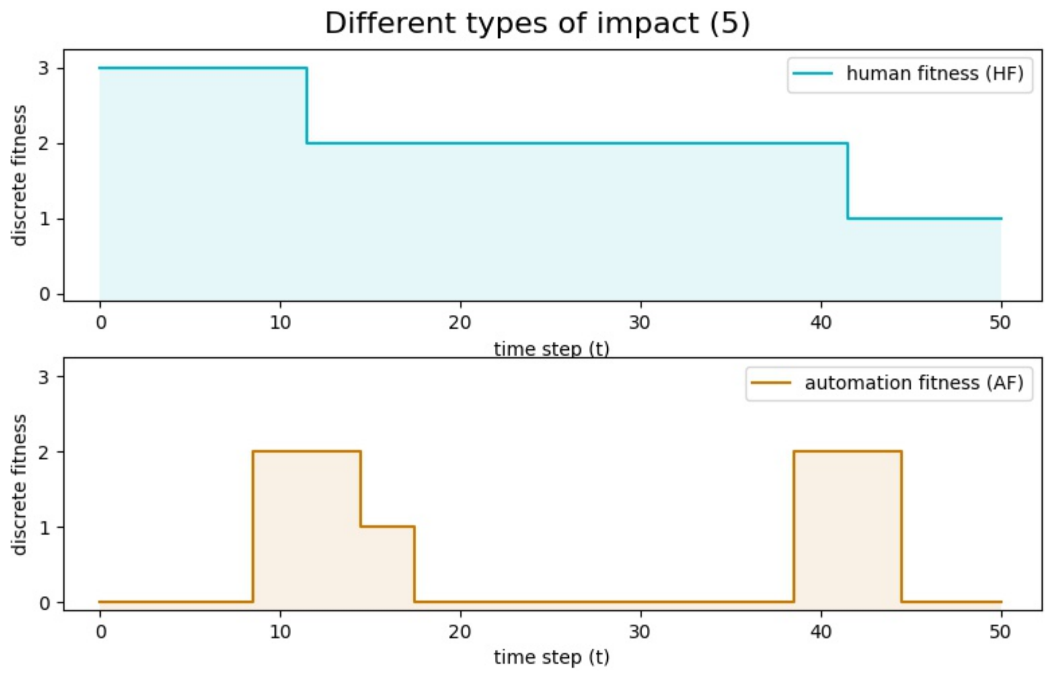


Figure 5.5: A visual representation showing the human fitness (HF) and autonomous fitness (AF) when no actions are performed during the time of the Different types of impact scenario

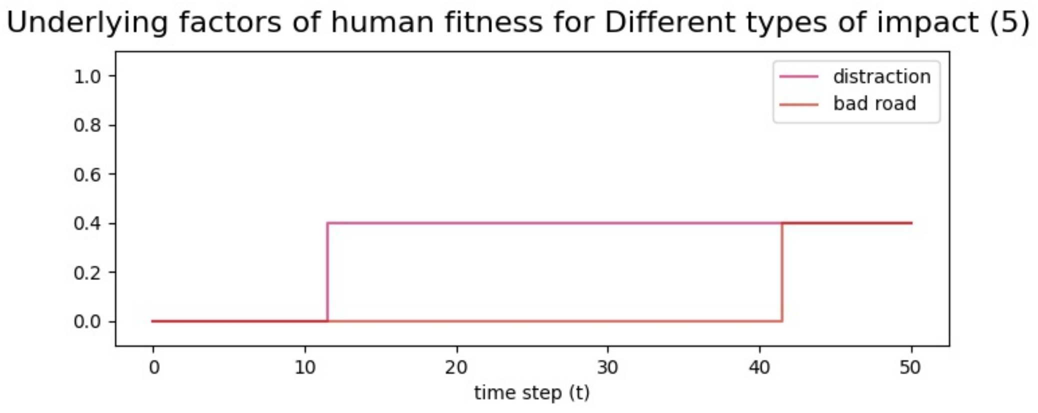


Figure 5.6: The values of factors causing a decrease in human fitness (HF) for the scenario: Different types of impact

underlying submodels. In both scenarios, this action is assumed to impact the *distraction* factor, which contributes negatively to human fitness.

The timeline of the first scenario, "Different types of impact", is displayed in Figure 5.5. In this scenario, the system becomes unsafe around $t = 28$ due to a low HF and a AF of 0. In Figure 5.6, the values of the underlying factors are displayed. It shows that the first decrease in HF is caused by an increase in *distraction*. The system should decrease the distraction before AF decreases to keep the system in a safe state. The MDP approach is expected to perform these actions appropriately.

Later in the scenario, HF decreases further due to an increase of the *bad road* factor. However, there is no action available for the agent to solve this problem. Consequently, an unsafe situation becomes inevitable, and the system should call for an emergency stop before the AF decreases to a level where no situation is safe.

Experiment 6 is the only scenario where the MDP approach is not to call for an emergency stop without certainty in the success of actions. The scenario is visually represented in Figure 5.7. In this scenario, the *distraction* is increased to a level that a single 'wake_up' action does not have enough impact to change the HF. However, performing the 'wake_up' action twice would result in a sufficiently fit driver, and prevent an emergency stop. The reason this scenario is not expected to

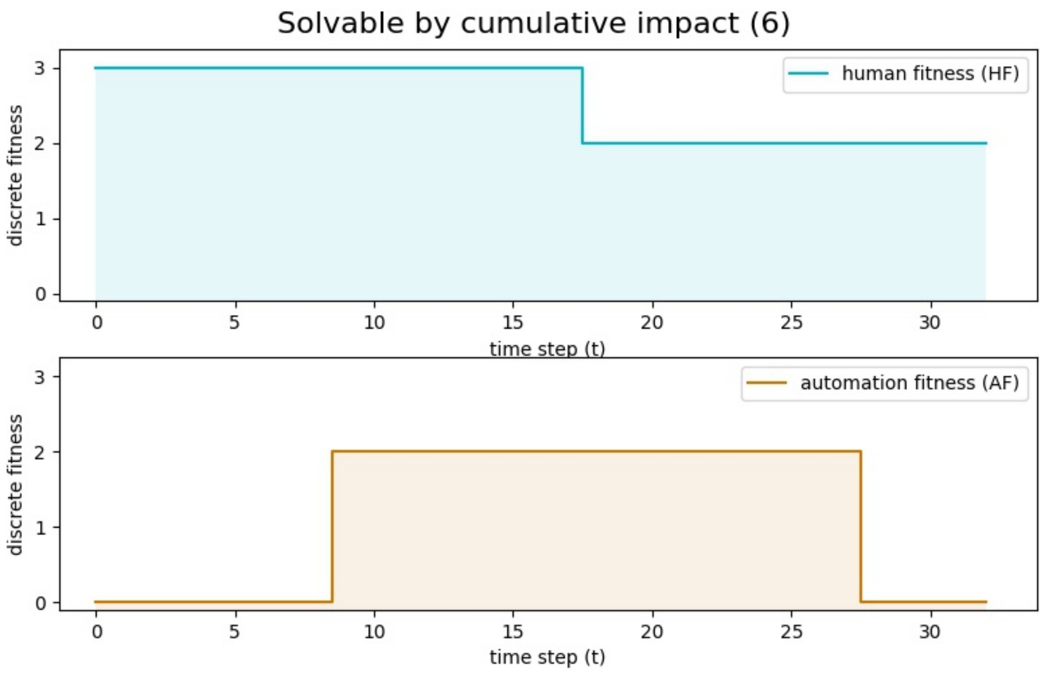


Figure 5.7: A visual representation showing the human fitness (HF) and autonomous fitness (AF) when no actions are performed during the time of the Solvable by cumulative impact.

Variable	**Variations**
k_{down}	[1, 3, 5]
P(down)	[.4, .6, .8]

Table 5.10: Variations of parameters for scenario 7. A cross product of all variations will be tested.

be adequately dealt with by the MDP approach is described in more detail in chapter 6.

5.5.4. Test the reward/risk trade-off

The last two experiments are designed to test how the system is going to behave under different levels of uncertainty in the action space. More specifically, scenario 7 has the goal to test to what level of risk the system is willing to increase automation level in order to optimize for comfort. This scenario is visually represented in Figure 5.8, and is the simplest one implemented. It has a driver that is constantly fit to drive and an autonomous system that is able to take over control for a period of 10 time steps.

In order to test what level of risk the system is willing to take for an optimization regarding comfort, both the probability of decreasing automation and the number of time steps decreasing automation may take are varied. The options for these two parameters are displayed in table 5.10. Next to these variations, a baseline is also set using $k_{down} = 1$ and a success probability of 1. It will be interesting to see how long the system stays in the higher level of automation, and whether it even takes the risk to increase the automation level.

5.5.5. Test dealing with room for choice

The last experiment has the goal to test how the system deals with choice when the probabilities of changing automation level are allowed to vary. The scenario is setup to give more freedom to optimize for automation than earlier scenarios have been. A visual representation is given in Figure 5.9. This situation is again simple, with a long stretch of increased automation fitness and a short period of decreased human fitness. This scenario allows an agent to choose which automation level it will operate in. Similar to the previous scenario, it will be most interesting to see how different

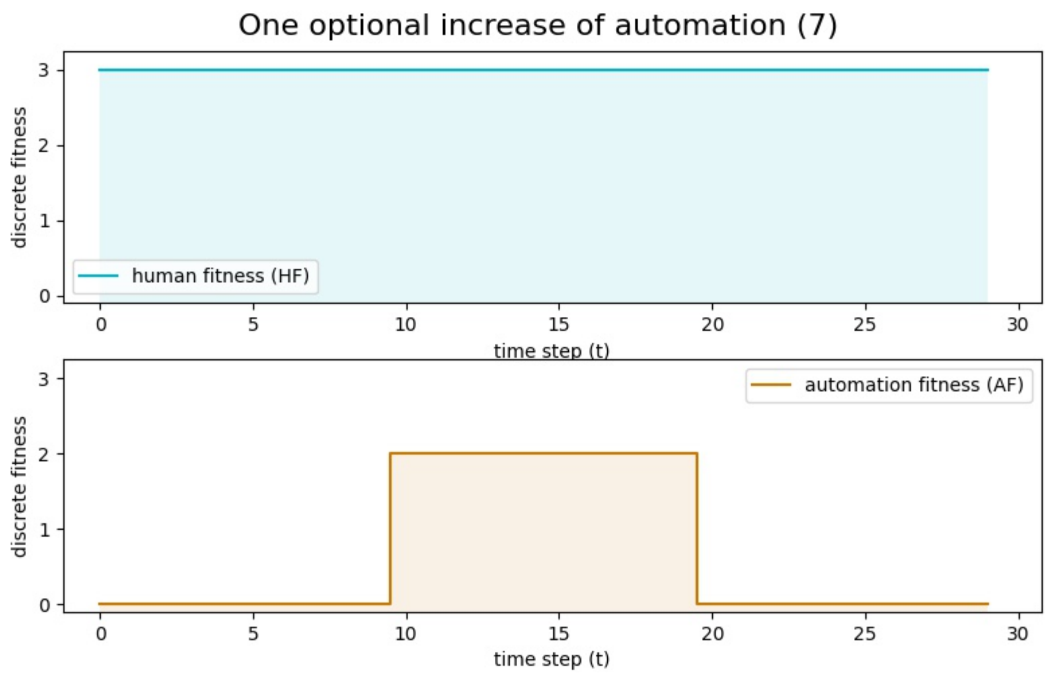


Figure 5.8: A visual representation showing the human fitness (HF) and autonomous fitness (AF) when no actions are performed during the time of the "One optional increase in automation" case.

#	($k_{up/down}$, P(up/down))
0	(1, 1)
1	(2, .5)
2	(2, .7)
3	(2, .95)
4	(3, .7)
5	(3, .8)
6	(3, .95)

Table 5.11: Variations of parameters for scenario 8.

values for the success probability and the number of attempts impact the agents behavior. To fully research the behavior change, this scenario will be tested for each reward system. Table 5.11 lists the parameter combinations that are tested.

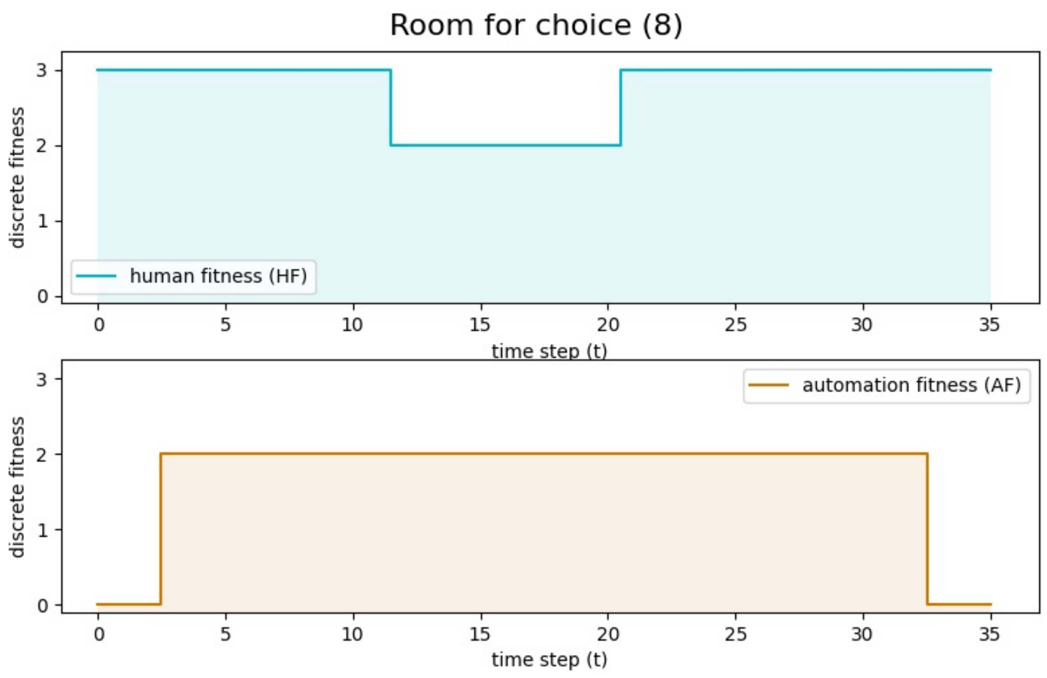


Figure 5.9: A visual representation showing the human fitness (HF) and autonomous fitness (AF) when no actions are performed during the time of the "Room for choice" case.

6 Results

In the majority of the experiments performed, the Markov decision process (MDP) approach, implemented as described in the previous chapter, behaved as hypothesized. This chapter reports the results obtained with the various experiments, where for the baseline experiment without uncertainty in the action space, these are visually displayed by plotting the state of the system over the period of the simulation. For the experiments with variation in the uncertainty of the action space, the impact of the different levels of uncertainty will be displayed. For each experiment, we will also discuss the implication of the results for the research question.

6.1. Base requirement

We start by discussing the results of experiment 1, which uses the "Base case" as a scenario. The results will be discussed in two separate parts. A baseline implementation where there is no uncertainty in the action space, followed by a discussion of how changing levels of uncertainty changes the behavior of the agent.

Baseline results

The baseline results of the MDP and the heuristic approach, displayed in Figures 6.1 and 6.2, show the MDP approach is able to complete the scenario safely, while the heuristic approach calls for an emergency stop at $t = 47$. The latter occurs because the heuristic approach is not able to plan ahead. At that point in the scenario, the autonomous fitness (AF) decreases, while the human fitness is still low. The MDP approach planned for this point in time by calling the 'wake_up' action earlier, and, moreover, started the decrease of the automation level in advance. This shows the MDP approach is capable of performing the planning task properly, whereas the heuristic approach is unable to deal with a scenario where planning is required.

Varying levels of uncertainty

Within the first experiment, the four parameters related to the uncertainty of changing automation were varied; that is, the cross-product of three options per factor were tested. Table 6.1 summarizes the variables of interest and their variations. Crossing these results in a total of 81 different combinations/runs.

As a summary of the outcomes of these 81 (sub)experiments, the three numeric results of main interest are: 1) the combinations of variables in which, either using the MDP or heuristic approach, the system is able to complete the run without calling for an emergency stop; 2) when an emergency stop is made, the point at which the system called for the emergency stop and why this happened; and 3) the mean automation level for the different combinations of settings. Because in this scenario the system is set to maximize the automation level, it is interesting to see how much reward it loses in order to guarantee safety with an uncertain action space.

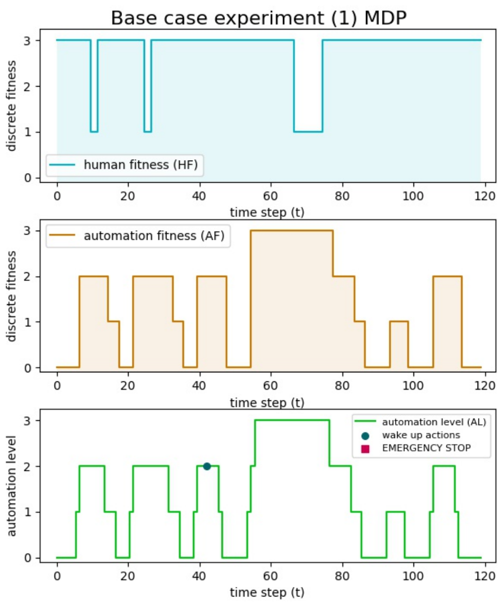


Figure 6.1: The human fitness, automation fitness, automation level and actions during the simulation of experiment 1, using the MDP approach.

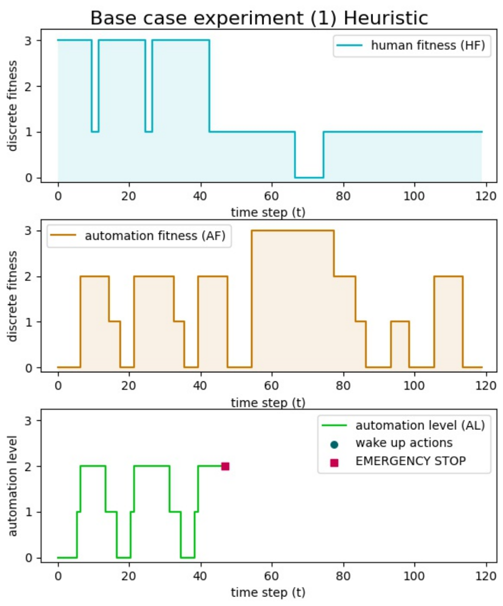


Figure 6.2: The human fitness, automation fitness, automation level and actions during the simulation of experiment 1, using the heuristic approach.

Variable	**Variations**
k_{up}	[1, 3, 5]
P(up)	[.4, .6, .8]
k_{down}	[1, 3, 5]
P(down)	[.4, .6, .8]

Table 6.1: Variations of parameters for scenario 1. A cross product of all variations will be tested.

T_{max}	**Condition**
9	*if* $k_{up} \geq 3$
14	*if* $k_{down} \geq 5$
120	*otherwise*

Table 6.2: The maximum point of time the simulation reaches under different parameters for scenario 1, the 'Base case'.

The first result worth mentioning is that the heuristic approach called for an emergency stop at $T = 14$ in all cases. This happens due to its lack of planning, which becomes even more of a problem when the action space is uncertain. At $T = 14$ the system comes in a state where the only way to stay in a safe situation is to decrease automation level, but the success of this change cannot be guaranteed. Because the system cannot guarantee safety anymore, it must call for an emergency stop.

The MDP approach was more successful (see Table 6.2). It turned out to be able to finish a run without calling for an emergency stop under two conditions; that is, when both the number of steps it can take to decrease automation level is less than three and the number of steps it can take to increase automation level is less than five. The reason the system calls for an emergency stop at $T = 9$ can be explained by the fact that the system must increase automation faster than can be guaranteed due to the number of steps a change of automation may take. The same happens at $T = 14$ if the number of steps it may take to decrease automation is five steps or more. This shows that longer options limit the agility of the system, namely the time scale at which it can handle increases or decreases of automation.

Now we will focus on the settings where the system run finished without calling for an emergency stop. Figure 6.3 displays the mean automation level over the course of the simulation for those settings. The first thing that can be observed is a confirmation of the hypothesis that decreasing the probability of success results in a system that takes less risk so overall has a lower mean automation level. Another interesting observation is that increasing the number of attempts the agent has generally speaking results in a decrease in the mean automation level. This could be explained by the fact that the agent has to start the action earlier as it may take longer before the execution is successful. However, for the lower success probability conditions, the effect reverses, because the impact of having more attempts in succession increases the total probability of success for the option. At this point, the positive effect of the cumulative probability of multiple attempts outweighs the negative effect of maybe needing more time to complete the action.

Another interesting observation is that increasing the number of attempts the agent has generally speaking results in a decrease in the of the mean automation level. That would be explained by the fact that the agent would have to start the action earlier as it may take longer before the execution is successful. However at the lower success probability the effect reverses, because the impact of having more attempts in succession increases the total probability of success for the option. At this point the positive effect of the cumulative probability of multiple attempts outweighs the negative effect of maybe needing more time to complete the action.

6.2. Showcasing preference

Figures 6.4, 6.5, and 6.6 show how the MDP changes its behavior with different preferences, by varying the reward function. Whereas each scenario has the same human fitness and autonomous fitness, changing the reward function reaches its goal of drastically changing the behavior while always staying in a safe operation mode. The heuristic approach does not have this capability because it does not have the notion of reward. However, small adjustments in the heuristic could result in similar behavior.

Figure 6.4 shows the system leverages all possible time it can to be in a higher automation level (AL) when the rewards function is set to 'max_automation'. The opposite happens when the rewards function is 'min_automation', as can be seen from Figure 6.5. The system only increases AL when the HF decreases. The last case, where the reward function is 'min_transition', shows it only changes the automation level when necessary due to safety constraints.

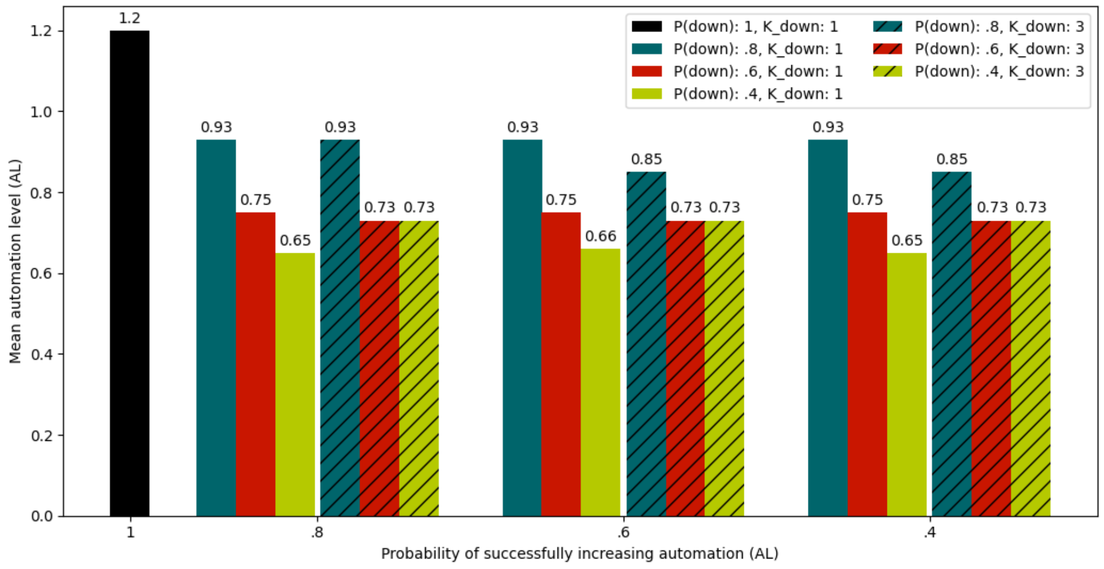


Figure 6.3: A graph showing the mean automation level for scenario 1 under different probability of success of decreasing/increasing automation, together with a variation of the number of steps a decrease in automation might take. These concern the set of variations resulting in a simulation run without an emergency stop.

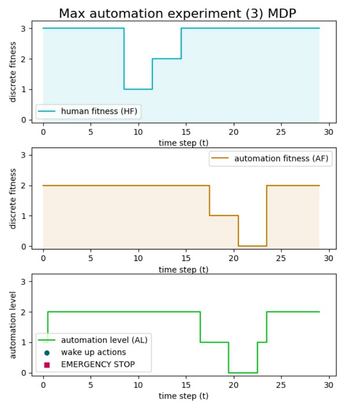


Figure 6.4: HF, AF, AL and actions during the simulation of experiment 2, using the MDP approach, with the 'max_automation' reward function.

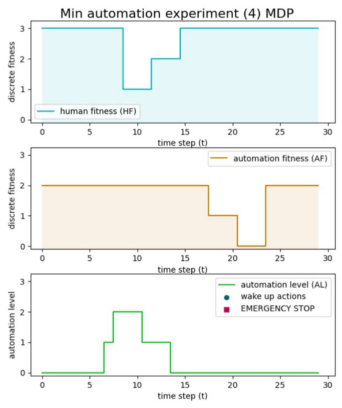


Figure 6.5: HF, AF, AL and actions during the simulation of experiment 3, using the MDP approach, with the 'min_automation' reward function.

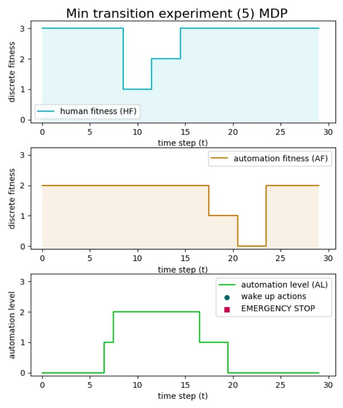


Figure 6.6: HF, AF, AL and actions during the simulation of experiment 4, using the MDP approach, with the 'min_transition' reward function.

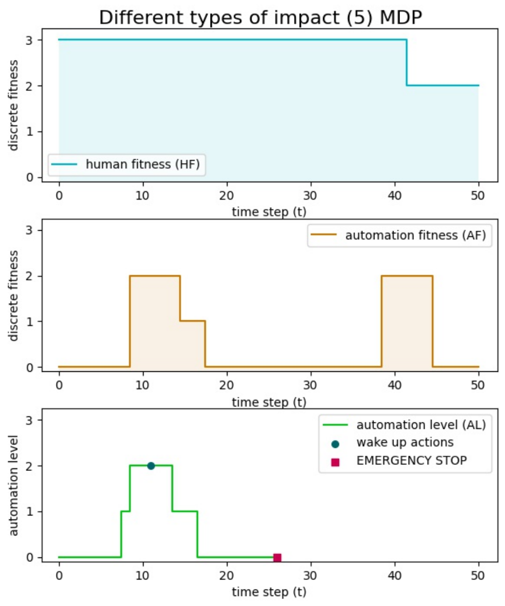


Figure 6.7: HF, AF, AL and actions during the simulation of experiment 5, Different types of impact, using the MDP approach

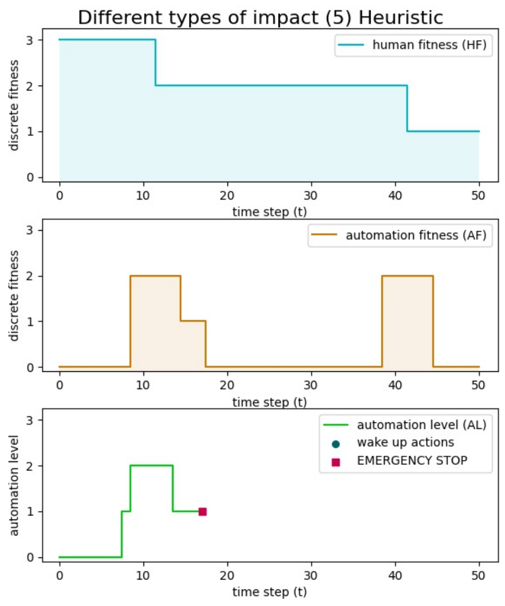


Figure 6.8: HF, AF, AL and actions during the simulation of experiment 5, Different types of impact, using the heuristic approach

6.3. Showcasing impact

The goal of experiment 5, which uses scenario "Different types of impact", was to demonstrate that certain human fitness (HF) decreases are solvable using action, while others are not. Figure 6.7 shows that the MDP approach indeed comes past the first decrease in HF by calling a 'wake_up' action in time. However, later at $t = 27$, the system calls for an emergency stop. The system sees that it will be inevitable to come in an unsafe state into the future.

The heuristic approach, displayed in Figure 6.8, calls for an emergency stop much earlier. Again the heuristic approach fails to plan, which creates a situation where it has to do a 'wake_up' action and a decrease of automation level at the same time. Since this is impossible, an unsafe situation has become inevitable.

Figures 6.9 and 6.10 are visualization of experiment 6, which uses scenario "Solvable by cumulative impact". The figures show that the system calls for an emergency stop, even though calling a 'wake_up' action twice could theoretically solve the problem. This turned out to be the case for both the heuristic and the MDP approach. However, it is not evident that it is desirable that the system calls the action twice and avoids an emergency stop. Within the set framework, there are several ways to prevent a too large decrease in human fitness. One is to have multiple 'wake_up' actions, with different amounts of impact. Another is to have a waking up consisting of a first and second alarm, which when called consecutive can be seen as different actions by the system. However, these design choices should be made in a more detailed and realistic implementation of the system.

It is interesting to look at the difference in the time the two approaches call for the emergency stop. The MDP approach turned out to call it much earlier. Because it plans up to the time horizon, the MDP knows earlier that an emergency stop is necessary. Consequently, the vehicle that uses the MDP approach would have more time to perform the emergency stop than the vehicle using the heuristic approach.

6.4. Showcasing the reward risk trade-off

Next, the results of experiment 7, 'One optional increase in automation', are discussed. This experiment consists of no more than a single stretch of increased autonomous fitness. The main goal of this experiment was to see up to what level of risk the agent is still willing to increase the automation level to optimize for comfort. The risk is varied over different runs by changing both the probability

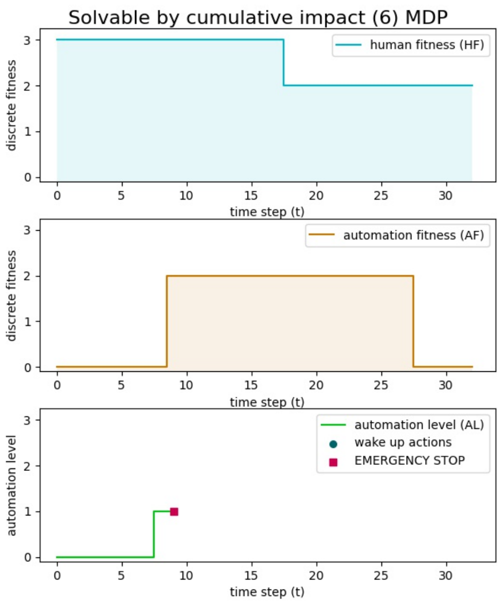


Figure 6.9: HF, AF, AL and actions during the simulation of experiment 5, Solvable by cumulative impact, using the MDP approach.

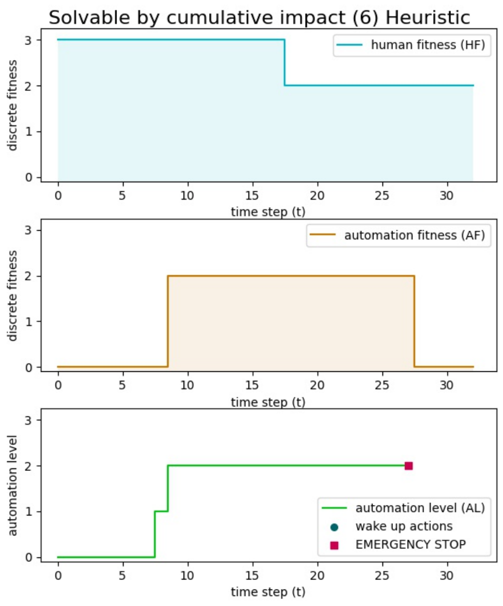


Figure 6.10: HF, AF, AL and actions during the simulation of experiment 5, Solvable by cumulative impact, using the heuristic approach.

Variable	**Variations**
k_{down}	[1, 3, 5]
P(down)	[.4, .6, .8]

Table 6.3: Variations of parameters for scenario 7. A cross product of all variations will be tested.

of successfully decreasing automation and the number of time steps a decrease action might take. In Table 6.3 the variations that were tested are displayed. A cross product was tested together with a baseline case where both variables are equal to 1.

Figure 6.11 displays the mean automation level of the different runs. The two phenomena with respect to increasing risk observed earlier can be viewed clearly again. The first being that the system will call for an action that brings the system back to guaranteed safety earlier as the number of attempts increases. This can be seen from the decrease in mean automation level for the experiments where P(down) is $.8$. The second phenomenon is that with increasing risk the expected reward increase never happens, resulting in a mean automation level of zero.

6.5. Showcasing dealing with room for choice

The goal of the last experiment was to show how different reward systems deal with uncertainty in the state space. The experiment is set up in such a way that the agent has a choice in which automation level it operates in. To test how the reward systems deal with the uncertain state space, six different configurations of probability of success and number of attempts an action can take are tested.

The results displayed in Figure 6.12 show the agent becomes less successful in optimizing for comfort as the action space becomes more uncertain. This results in a decrease in mean automation level when the system optimizes for a maximum automation level and an increase in automation level when the agent attempts to minimize the automation level.

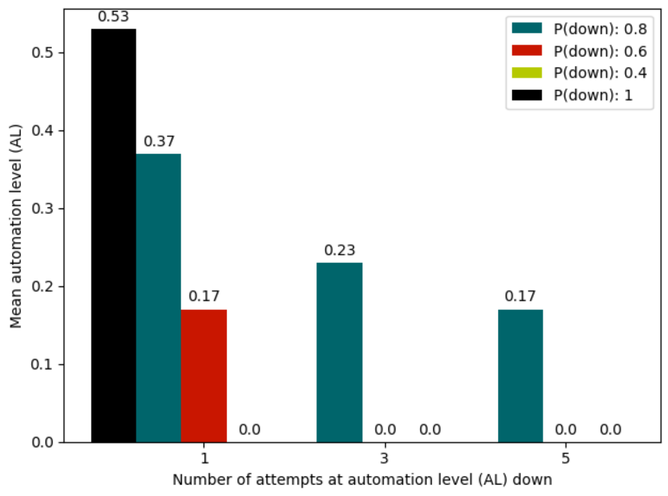


Figure 6.11: A graph showing the mean automation level for scenario 7 under for different probabilities of success of decreasing and different numbers of steps a decrease in automation might take.

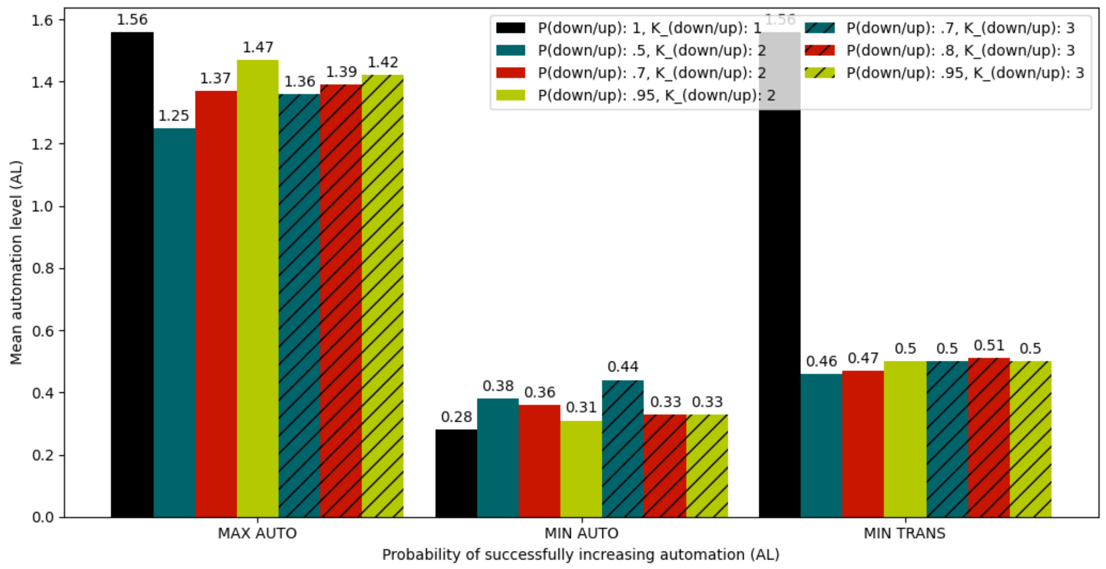


Figure 6.12: A graph showing the mean automation level for scenario 8 for multiple rewards systems, with different variations of probability of success decreasing and increasing automation, and a variation of the number of steps a decrease or increase in automation might take.

7 Conclusion

To conclude the research described in the previous chapters, in this chapter, we first evaluate whether the proposed model can meet the system requirements set at the beginning of this thesis. Next, we look in more detail into the answer obtained for the main research question, which is followed by a discussion of the limitations of the current study and recommendations for future research.

7.1. Requirement satisfaction

As indicated in chapter 2, the relevant system requirements consists of model input and output requirements, and requirements regarding the predictability and explainability of the system. Below, we evaluated whether the proposed models fulfill each of these requirements using the results described in chapter 6. All conclusion which are drawn in this section are under the assumption that the underlying models of the system are accurate.

7.1.1. Behavioral requirement evaluation

The behavioral requirements described in section 2.2 imply the system should both operate safely and optimize for comfort. The "operate safely" requirement is fulfilled due to the design of the system, which guarantees the model can never go to an unsafe situation. The comfort requirement is harder to deal with, as its specific meaning will depend on the end-user of the system. However, as shown in chapter 6, the proposed model design allows for multiple definitions of comfort, implying that it can be adapted to the preferences of an end-user. Therefore, it can be concluded that the proposed system meets both types of behavioral requirements.

7.1.2. Model requirements evaluation

In section 2.3, it was indicated the essential model requirements are that its input and output should be appropriate and that it should be predictable and explainable. As far as the input requirements are concerned, most aspects are dealt with appropriately. However, it should be noted that in the model definition we assumed all input values are worst-case estimates. While this guarantees safety under these kinds of scenarios, in real-life, this may yield a system that constantly calls for emergency stops, as the worst-case estimate is always too pessimistic. The only way to tackle that problem, is to have accurate enough underlying models.

The output that the system needs to be able to deal with is all properly covered by the model. In chapter 6 it has been shown how the system executes a variety of actions, which covers the requirements. Also, the experiments have shown that the system is able to deal with the uncertainty with respect to changing automation level. However, the idea of uncertainty of the impact that actions can have is not fully explored and should be further investigated.

As far as the explainability and predictability of the system is concerned, this turns out to depend strongly on the complexity of the underlying models of the environment. Since the system depends on the submodels used for the underlying factors, specifying these submodels to be black-boxes will yield an overall system based on a Markov decision process (MDP), which becomes itself a black-box. However, as long as this is not the case, we can fully draw out the boundaries for the evaluation of a system state to be safe or not. In the same way, without ambiguity, it can be predicted what the system is going to do, under set conditions.

7.2. Answer to research question

The research question, as posed in section 1.5 was as follows:

Is a Markov decision process a feasible model for the decision logic in a semi-autonomous vehicle, which has the task to mediate control between the human driver and the autonomous system?

Looking at section 7.1, it can be seen that the requirements are met. This means we can conclude that using a Markov decision process to model the decision logic as defined by the MEDIATOR project is feasible. Moreover, our results show that an MDP approach can deal with the specific challenges posed by this problem. However, the current study also comes with certain limitations, which are discussed in the following section.

7.3. Limitations of the current study

To show that an MDP is a feasible model, various assumptions and simplifications had to be made.

The first limitation concerns the current implementation of the simulator. The simulator provides a perfect estimate of the worst-case future, and this worst-case also always comes out. However, it would be interesting to see what happens if the values read at the current state are more positive than the expected values.

The main limitation of the way the worst-case has currently been implemented is that it behaves rather linearly, whereas in real-life, one may expect estimates to diverge more strongly from their real values as the estimation gets further into the future. This effect might cause a system to call for emergency stops too often. Future implementations will have to to deal with this by having underlying models that are accurate enough for values in the future.

Another important assumption we made is that the decision logic is in full control of the automation level. However, one can also think of a system allowing the user to determine what automation level he prefers at a specific moment during a ride. This would indeed be a good feature from a user perspective. However, this makes the planning for the MDP much more difficult, as it can no longer assume the computed policy is always executed as determined by the system.

7.4. Recommendations for future research

To conclude this thesis, a set of recommendations for further research is given. What is clear is that to make the model more usable in real-life test settings, the underlying submodels need to be further developed. Therefore, my recommendation to the MEDIATOR project is to give high priority to the development of these underlying models. No matter how sophisticated the decision logic itself is modeled, its quality will always, to a large extent, be determined by the quality of the underlying submodels.

As these underlying submodels get more advanced, an important question remains unanswered: how the decision logic operates under the resulting added complexity. Therefore, future research should go into understanding how well an MDP approach would hold under much more complex, or even black-box, models of the environment.

Together with the development of those models, it should also be researched how to determine whether a state is safe. This is a two-sided problem. The first side concerns determining the acceptable level of uncertainty, because no matter how good the estimate, it will never be entirely sure about the current state. This implies that a certain level of risk needs to be accepted. The second part concerns the actual computation of safety; that is, a function determining which level of distraction is considered unsafe or until at what point the lane visibility is not a problem. The model developed in this thesis may also be used to derive possible implications of decisions regarding those safety functions for the system as a whole.

The last few recommendations concerned the state space, however, it will also be necessary to further investigate the complexity of the action space. The current study has a very simplistic implementation of the options available for the system. However, it is very well imaginable that these will become more complex. It is important to investigate how well the system performs under more complexity in the action space.

Another issue, essential for real-life versions of the system has not been dealt within the current study. This concerns the time scale in which the system operates, which is an issue with multiple facets. The first is the actual time horizon for which the system is supposed to plan. If the system needs to be able to take into account events in the future which are hours away, but at the same time the system needs to reason on the level of seconds, the total state space of our model will become unfeasibly large. Future research may look into using a time hierarchical structuring of the state space to deal with this issue [9] [23] [8].

An alternative manner to set the time horizon is to approach it from the point of view that it does not matter how far ahead the time horizon is as long as the current decision is optimal. [21] looked into such an approach, and future research in the decision logic may go into this direction as well.

When dealing with uncertainty using the worst-case approach, some information is lost in the simplification. It could be of interest to turn the MDP into a partially observable Markov decision process (POMDP). The reason this might be interesting is that even though a POMDP is not able to deal with safety differently, it might operate better in situations where the system has room to optimize for comfort, i.e., when the entire uncertainty region lies in safe spaces with choice in automation level (AL).

Our final recommendation concerns the method used to solve the model. While in the current implementation time to solve the model has not been part of the requirements, in a future system this will become important. Depending on whether the transition to a POMDP is made or not, there are many possible ways to solve the model. Our recommendation would be to find a method which utilizes the fact that many states in the state space are unreachable: ignoring those states when searching for an optimal next action can give great improvements in the speed of solving the model.

A
Scenario Files

Listing A.1: Scenario 1, The full travel experience

```
{
  ”description”: ”Experiment 1, Scenario: Base Case. After the
      presentation given by Bram Bakker on Feb 3 2020, we will try to
      invoke most required actions of the mediator.”,
  ”totalT”: 120,
  ”timeToES”: 5,
  ”rewardSystem”: ”max_automation”,
  ”factors”: [
    {
      ”name”: ”D_LoA0”,
      ”value”: ”${HC} > .8 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA1”,
      ”value”: ”${HC} > .6 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA2”,
      ”value”: ”${HC} > .2 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA3”,
      ”value”: ”${HC} > .1 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },

    {
      ”name”: ”A_LoA0”,
      ”value”: ”${AC} > 0 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”A_LoA1”,
      ”value”: ”${AC} > .3 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”A_LoA2”,
      ”value”: ”${AC} > .7 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”A_LoA3”,
      ”value”: ”${AC} > .9 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },

    {
      ”name”: ”HC”,
      ”value”: ”1 − .5 ⋆ ${sleepiness} − .5 ⋆ ${distraction}”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”AC”,
      ”value”: ”${road_condition} + ${traffic_predictability} + ${
          long_term_safety}”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },

    {
      ”name”: ”sleepiness”,
      ”value”: ”${_t} === 43 ? ${−1} + 1 : ${−1}”,
      ”start_value”: 0.01,
      ”type”: ”modeled”,
      ”dependence”: ”exogenous”,
      ”error”: 0.5
    },
    {
      ”name”: ”distraction”,
      ”value”: ”(9 < ${_t} && ${_t} < 12) || (24 < ${_t} && ${_t} < 27) ||
           (66 < ${_t} && ${_t} < 75) ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”exogenous”,
      ”error”: 0.5
    },

    {
      "name": "long_term_safety",
      "value": " (54 < ${_t} && ${_t} < 78) ? .2 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },

    {
      "name": "road_condition",
      "value": "(6 < ${_t} && ${_t} < 18) || (21 < ${_t} && ${_t} < 36) ||
          (39 < ${_t} && ${_t} < 48) || (54 < ${_t} && ${_t} < 87)  || (93
          < ${_t} && ${_t} < 99) || (105 < ${_t} && ${_t} < 114) ? .4 : 0"
        ,
      "type": "modeled",
      "dependence": "exogenous",
      "error": 0.5
    },
    {
      "name": "traffic_predictability",
      "value": "(6 < ${_t} && ${_t} < 15) || (21 < ${_t} && ${_t} < 33) ||
          (39 < ${_t} && ${_t} < 48) || (54 < ${_t} && ${_t} < 84)  || (10
        5 < ${_t} && ${_t} < 114) ? .4 : 0",
      "type": "modeled",
      "dependence": "exogenous",
      "error": 0.5
    }
  ],
  "alActionParams": {
    "up": {
      "numberOfAttempts": X,
      "successOfPrimitive": X
    },
    "down": {
      "numberOfAttempts": X,
      "successOfPrimitive": X
    }
  },
  "actionEffects": [
    {
      "primitive_name": "hc_up",
      "name": "alarmAction",
      "on_factor": "sleepiness",
      "diff": -1.0,
      "std": 0.0
    }
  ]
}
```

Listing A.2: Scenario 2, Showcasing a maximum automation reward system

```
{
  "description": "Experiment 2, Scenario: Simple scenario. Showcases the
      max automation reward system.",
"startLoa": 0,
"totalT": 30,
"timeToES": 5,
"rewardSystem": "max_automation",
"factors": [
  {
    "name": "D_LoA0",
    "value": "${HC} > .8 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "D_LoA1",
    "value": "${HC} > .6 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "D_LoA2",
    "value": "${HC} > .2 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "D_LoA3",
    "value": "${HC} > .1 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },

  {
    "name": "A_LoA0",
    "value": "${AC} > 0 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "A_LoA1",
    "value": "${AC} > .5 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "A_LoA2",
    "value": "${AC} > .7 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "A_LoA3",
    "value": "${AC} > .9 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
    },

    {
      "name": "HC",
      "value": "1 − ${dip9} − ${dip12}",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "AC",
      "value": ".8 − ${dip18} − ${dip21}",
      "type": "modeled",
      "dependence": "exogenous"
    },

    {
      "name": "dip9",
      "value": "9 <= ${_t} && ${_t} < 12 ? .7 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "dip12",
      "value": "12 <= ${_t} && ${_t} < 15 ? .3 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },

    {
      "name": "dip18",
      "value": "18 <= ${_t} && ${_t} < 21 ? .2 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "dip21",
      "value": "21 <= ${_t} && ${_t} < 24 ? .5 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    }

  ],
  "actionEffects": [
  ]
}
```

Listing A.3: Scenario 3, Showcasing a minimum automation reward system

```
{
  "description": "Experiment 3, Scenario: Simple scenario. Showcases the
     min automation reward system.",
  "startLoa": 0,
  "timeToES": 5,
  "totalT": 30,
  ”rewardSystem”: ”min_automation”,
  ”factors”: [
    {
      ”name”: ”D_LoA0”,
      ”value”: ”${HC} > .8 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA1”,
      ”value”: ”${HC} > .6 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA2”,
      ”value”: ”${HC} > .2 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA3”,
      ”value”: ”${HC} > .1 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },

    {
      ”name”: ”A_LoA0”,
      ”value”: ”${AC} > 0 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”A_LoA1”,
      ”value”: ”${AC} > .5 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”A_LoA2”,
      ”value”: ”${AC} > .7 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”A_LoA3”,
      ”value”: ”${AC} > .9 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },

    {
      "name": "HC",
      "value": "1 - ${dip9} - ${dip12}",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "AC",
      "value": ".8 - ${dip18} - ${dip21}",
      "type": "modeled",
      "dependence": "exogenous"
    },

    {
      "name": "dip9",
      "value": "9 <= ${_t} && ${_t} < 12 ? .7 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "dip12",
      "value": "12 <= ${_t} && ${_t} < 15 ? .3 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },

    {
      "name": "dip18",
      "value": "18 <= ${_t} && ${_t} < 21 ? .2 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "dip21",
      "value": "21 <= ${_t} && ${_t} < 24 ? .5 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    }

  ],
  "actionEffects": [
  ]
}
```

Listing A.4: Scenario 4, Showcasing a minimum transition reward system

```
{
  "description": "Experiment 4, Scenario: Simple scenario. Showcases the
      min transition reward system.",
  "startLoa": 0,
  "timeToES": 5,
  "totalT": 30,
  "rewardSystem": "min_transitions",
  "factors": [
    {
```

```
    "name": "D_LoA0",
    "value": "${HC} > .8 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "D_LoA1",
    "value": "${HC} > .6 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "D_LoA2",
    "value": "${HC} > .2 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "D_LoA3",
    "value": "${HC} > .1 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },

  {
    "name": "A_LoA0",
    "value": "${AC} > 0 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "A_LoA1",
    "value": "${AC} > .5 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "A_LoA2",
    "value": "${AC} > .7 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },
  {
    "name": "A_LoA3",
    "value": "${AC} > .9 ? 1 : 0",
    "type": "modeled",
    "dependence": "endogenous"
  },

  {
    "name": "HC",
    "value": "1 - ${dip9} - ${dip12}",
    "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "AC",
      "value": ".8 - ${dip18} - ${dip21}",
      "type": "modeled",
      "dependence": "exogenous"
    },

    {
      "name": "dip9",
      "value": "9 <= ${_t} && ${_t} < 12 ? .7 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "dip12",
      "value": "12 <= ${_t} && ${_t} < 15 ? .3 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },

    {
      "name": "dip18",
      "value": "18 <= ${_t} && ${_t} < 21 ? .2 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "dip21",
      "value": "21 <= ${_t} && ${_t} < 24 ? .5 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    }

  ],
  "actionEffects": [
  ]
}
```

Listing A.5: Scenario 5, Showcasing how impact can change model behavior

```
{
  "description": "Experiment 5, Scenario: Different types of impact. Two
      types of Human Confidence degradation, one we can solve and one we
      cannot",
  "timeToES": 20,
  "totalT": 51,
  "rewardSystem": "max_automation",
  "factors": [
    {
      "name": "D_LoA0",
      "value": "${HC} > .8 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "D_LoA1",
      "value": "${HC} > .6 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "D_LoA2",
      "value": "${HC} > .2 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "D_LoA3",
      "value": "${HC} > .1 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },

    {
      "name": "A_LoA0",
      "value": "${AC} > 0 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA1",
      "value": "${AC} > .5 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA2",
      "value": "${AC} > .7 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA3",
      "value": "${AC} > .9 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },

    {
      "name": "HC",
      "value": "1 - ${distraction} - ${badRoad}",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "AC",
      "value": "(9 <= ${_t} && ${_t} < 18) || (39 <= ${_t} && ${_t} < 45)
         ? .8 − ${ACDegrade} : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "ACDegrade",
      "value": "(15 <= ${_t} && ${_t} < 18) ? .2 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "badRoad",
      "value": "${_t} === 42 ? .3 : ${−1}",
      "start_value": 0.00,
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "distraction",
      "value": "${_t} === 12 ? .3 : ${−1}",
      "start_value": 0.00,
      "type": "modeled",
      "dependence": "exogenous"
    }
  ],
  "actionEffects": [
    {
      "primitive_name": "hc_up",
      "name": "alarmAction",
      "on_factor": "distraction",
      "diff": −0.5,
      "std": 0.0
    }
  ]
}
```

Listing A.6: Scenario 6, Showcasing the result of not having cumulative impact modeled

```
{
  "description": "Experiment 6, Scenario: Solvable by cumulative impact. A
       scenario where a single wake up action does not have enough impact
     to solve human fitness decrease.",
  "timeToES": 20,
  "totalT": 33,
  "rewardSystem": "max_automation",
  "factors": [
    {
      "name": "D_LoA0",
      "value": "${HC} > .8 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "D_LoA1",
      "value": "${HC} > .6 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "D_LoA2",
      "value": "${HC} > .2 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "D_LoA3",
      "value": "${HC} > .1 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },

    {
      "name": "A_LoA0",
      "value": "${AC} > 0 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA1",
      "value": "${AC} > .5 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA2",
      "value": "${AC} > .7 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA3",
      "value": "${AC} > .9 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },

    {
      "name": "HC",
      "value": "1 − ${distraction}",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "AC",
      "value": "(9 <= ${_t} && ${_t} < 28) ? .8 : 0",
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },

    {
      ”name”: ”distraction”,
      ”value”: ”${_t} === 18 ? .35 : ${−1}”,
      ”start_value”: 0.00,
      ”type”: ”modeled”,
      ”dependence”: ”exogenous”
    }
  ],
  ”actionEffects”: [
    {
      ”primitive_name”: ”hc_up”,
      ”name”: ”alarmAction”,
      ”on_factor”: ”distraction”,
      ”diff”: −0.1,
      ”std”: 0.0
    }
  ]
}
```

Listing A.7: Scenario 7, One optional increase in automation

```
{
  ”description”: ”Experiment 7, Scenario: A simple scenario where the
      agent has to option to go up in automation for a short period of time
      .”,
  ”timeToES”: 5,
  ”startLoa”: 0,
  ”totalT”: 30,
  ”rewardSystem”: ”max_automation”,
  ”factors”: [
    {
      ”name”: ”D_LoA0”,
      ”value”: ”${HC} > .8 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA1”,
      ”value”: ”${HC} > .6 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA2”,
      ”value”: ”${HC} > .2 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA3”,
      "value": "${HC} > .1 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },

    {
      "name": "A_LoA0",
      "value": "${AC} > 0 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA1",
      "value": "${AC} > .5 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA2",
      "value": "${AC} > .7 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA3",
      "value": "${AC} > .9 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },

    {
      "name": "HC",
      "value": "1",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "AC",
      "value": ".0 + ${upgrade}",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "upgrade",
      "value": "10 <= ${_t} && ${_t} < 20 ? .8 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    }

  ],
  "actionEffects": [
  ],
  "alActionParams": {
    ”up”: {
      ”numberOfAttempts”: 1,
      ”successOfPrimitive”: 1
    },
    ”down”: {
      ”numberOfAttempts”: X,
      ”successOfPrimitive”: X
    }
  }
}
```

Listing A.8: Scenario 8, Room for choice

```
{
  ”description”: ”Experiment 8, Scenario: Flexibility for interesting
      behaviour”,
  ”timeToES”: 5,
  ”startLoa”: 0,
  ”totalT”: 36,
  ”rewardSystem”: ”max_automation”,
  ”factors”: [
    {
      ”name”: ”D_LoA0”,
      ”value”: ”${HC} > .8 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA1”,
      ”value”: ”${HC} > .6 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA2”,
      ”value”: ”${HC} > .2 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”D_LoA3”,
      ”value”: ”${HC} > .1 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },

    {
      ”name”: ”A_LoA0”,
      ”value”: ”${AC} > 0 ? 1 : 0”,
      ”type”: ”modeled”,
      ”dependence”: ”endogenous”
    },
    {
      ”name”: ”A_LoA1”,
      "value": "${AC} > .5 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA2",
      "value": "${AC} > .7 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },
    {
      "name": "A_LoA3",
      "value": "${AC} > .9 ? 1 : 0",
      "type": "modeled",
      "dependence": "endogenous"
    },

    {
      "name": "HC",
      "value": "1 - ${hc_dip}",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "AC",
      "value": ".0 + ${upgrade_big} + ${upgrade_small}",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "hc_dip",
      "value": "12 <= ${_t} && ${_t} < 21 ? .3 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "upgrade_big",
      "value": "3 <= ${_t} && ${_t} < 27 ? .8 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    },
    {
      "name": "upgrade_small",
      "value": "27 <= ${_t} && ${_t} < 33 ? .8 : 0",
      "type": "modeled",
      "dependence": "exogenous"
    }
  ],
  "alActionParams": {
    "up": {
      "numberOfAttempts": X,
      "successOfPrimitive": X
    },
    ”down”: {
      ”numberOfAttempts”: X,
      ”successOfPrimitive”: X
    }
  },
  ”actionEffects”: [
  ]
}
```

Bibliography

[1] Eitan Altman. *Constrained Markov decision processes*. CRC Press, 1999.

[2] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete Problems in AI Safety. *arXiv preprint*, 6 2016. URL http://arxiv.org/abs/1606.06565.

[3] James M Anderson, Nidhi Kalra, Karlyn D Stanley, Paul Sorensen, Constantine Samaras, and Oluwatobi A Oluwatola. Brief History and Current State of Autonomous Vehicles. In *Autonomous Vehicle Technology: A Guide for Policymakers*, pages 55–74. RAND Corporation, 2014. ISBN 9780833083982. doi: 10.7249/j.ctt5hhwgz.11.

[4] Yossi Aviv and Awi Federgruen. The value iteration method for countable state Markov decision processes. *Operations Research Letters*, 24(5):223–234, 1999. URL www.elsevier.com/locate/orms.

[5] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[6] Steven J Bradtke and Michael O Duff. Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In *Advances in neural information processing systems*, pages 393–400, 1995.

[7] Torpong Cheevaprawatdomrong, Irwin E. Schochetman, Robert L. Smith, and Alfredo Garcia. Solution and forecast horizons for infinite-horizon nonhomogeneous Markov decision processes. *Mathematics of Operations Research*, 32(1):51–72, 2 2007. ISSN 0364765X. doi: 10.1287/moor.1060.0224.

[8] Peter Dayan and Geoffrey E Hinton. Feudal Reinforcement Learning. *Advances in neural information processing systems*, 1993.

[9] Thomas G Dietterich. The MAXQ Method for Hierarchical Reinforcement Learning. *ICML*, 98: 118–126, 1998.

[10] Yasin Gocgun, Brian W. Bresnahan, Archis Ghate, and Martin L. Gunn. A Markov decision process approach to multi-category patient scheduling in a diagnostic facility. *Artificial Intelligence in Medicine*, 53(2):73–81, 10 2011. ISSN 09333657. doi: 10.1016/j.artmed.2011.06.001.

[11] Kamal Golabi, Ram B. Kulkarni, and George B. Way. A Statewide Pavement Management System. *Interfaces*, 12(6):5–21, 12 1982. doi: 10.1287/inte.12.6.5. URL http://pubsonline.informs.org/doi/abs/10.1287/inte.12.6.5.

[12] Bryce Goodman and Seth Flaxman. European Union regulations on algorithmic decision-making and a "right to explanation". *AI magazine*, 38(3):50–57, 6 2017. doi: 10.1609/aimag.v38i3.2741. URL http://arxiv.org/abs/1606.08813http://arxiv.org/abs/1606.08813.

[13] Wallace J. Hopp, James C. Bean, and Robert L. Smith. NEW OPTIMALITY CRITERION FOR NON-HOMOGENEOUS MARKOV DECISION PROCESSES. *Operations Research*, 35(6):875–883, 1987. ISSN 0030364X. doi: 10.1287/opre.35.6.875.

[14] SEA International. Automated driving levels of driving automation are defined in new sae international standard j3016, 2020. https://web.archive.org/web/20161120142825/http://www.sae.org/misc/pdfs/automated_driving.pdf, Accessed: 3-4-2020.

[15] Rodrigue T Kuate, Minghua He, Maria Chli, and Hai H Wang. An Intelligent Broker Agent for Energy Trading: An MDP Approach. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[16] Mike Lemanski. A stepwise approach to reach full automation. URL `SAE.org`.

[17] Michael L Littman, Thomas L Dean, and Leslie Pack Kaelbling. On the Complexity of Solving Markov Decision Problems. *arXiv preprint arXiv:1302.4971*, 2013.

[18] Amy Mcgovern, Doina Precup, Balaraman Ravindran, Satinder Singh, and Richard S Sutton. Hierarchical Optimal Control of MDPs. *Proceedings of the Tenth Yale Workshop on Adaptive and Learning Systems*, pages 186–191, 1998.

[19] Mediator. Mediator site, 2019. URL `http://mediatorproject.eu/`.

[20] John A Michon. A CRITICAL VIEW OF DRIVER BEHAVIOR MODELS: WHAT DO WE KNOW, WHAT SHOULD WE DO? *L. Evans & R. C. Schwing (Eds.). Human behavior and traffic safety*, 485-520, 1985.

[21] Grigory Neustroev, Mathijs De Weerdt, and Remco Verzijlbergh. Discovery of Optimal Solution Horizons in Non-Stationary Markov Decision Processes with Unbounded Rewards. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 292–300, 2019. URL `www.aaai.org`.

[22] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[23] Doina Precup and Richard S Sutton. Multi-time Models for Temporally Abstract Planning. In *Advances in neural information processing systems*, pages 1050–1056, 1998.

[24] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[25] Christian Ruf and Peter Stütz. Model-driven sensor operation assistance for a transport helicopter crew in manned-unmanned teaming missions: Selecting the automation level by machine decision-making. In *Advances in Intelligent Systems and Computing*, volume 499, pages 253–265. Springer Verlag, 2017. ISBN 9783319419589. doi: 10.1007/978-3-319-41959-6{_}21.

[26] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 00043702. doi: 10.1016/S0004-3702(99)00052-1.

[27] SWOV. Werkelijk aantal verkeersdoden, 2020. `https://theseus.swov.nl/single/?appid=5dbac35a-5fbd-401f-b711-682176941688&sheet=ZjpemJ`, Accessed: 23-4-2020.

[28] Chao Tang, Xin Chen, Ying Chen, and Zhuo Li. A MDP-Based Network Selection Scheme in 5G Ultra-Dense Network. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 823–830. IEEE, 2018. ISBN 9781538673089.

[29] Georgios Theocharous Sridhar Mahadevan and East Lansing. Approximate Planning with Hierarchical Partially Observable Markov Decision Process Models for Robot Navigation. *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, 2: 1347–1352, 2002.

[30] Chris Van Winden and Rommert Dekker. Rationalisation of building maintenance through Markov decision models. In *Operations Research Proceedings 1993*, pages 536–536. Springer, 1994.