To Err is Human: Designing Correct-by-Construction Driver Assistance Systems using Cognitive Modelling

Francisco Girbal Eiras

Linacre College, University of Oxford

Supervised by:

Dr. Morteza Lahijanian and Prof. Marta Kwiatkowska

A thesis submitted for the degree of

Master of Science in Computer Science

Trinity 2018

Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

Contents

List of Figures

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

Chapter 2

Literature Review and Definitions

This chapter presents the concepts and theory required to understand the methodology, rationale and relevance of this dissertation. It starts by defining a Cognitive Architecture, and goes into more detail about Adaptive Control of Thought - Rational, which is the underlying framework used in the Integrated Driver Modelling described in Section 2.2 and the basis for the system developed in Chapter 3: *Human Driver Modelling*. The chapter concludes with the introduction of several important concepts in the area of Probabilistic Model Checking, which are essential for modelling, verification and synthesis in this context.

2.1 Cognitive Architectures

Different definitions of a cognitive architecture can be found in the literature [12]. According to Sun, a cognitive architecture is a *"broadly-scoped, domain-generic computational cognitive model, capturing the essential structure and process of the mind, to be used for a broad, multiple-level, multiple-domain analysis of behavior"* [24]. In that sense, a cognitive architecture is no more than a formalised framework of the perception, memorisation, decision making, reasoning and execution of the human mind. It is important to note that this architecture should capture both the abilities (e.g. memory storage and recall, perception or motor action) and constraints (e.g. memory decay and limited motor performance) of the human system [22].

The idea of a cognitive architecture originated in the early 1970s. A survey by Korseruba and Tsotsos [12] revealed that, since then, an estimated 195 different cognitive

architectures with different assumptions, structural organisations and implementations have been developed. Applications of these range from the fields of Robotics to Natural Language Processing [12]. One of the most well known of those architectures is Adaptive Control of Thought - Rational.

2.1.1 Adaptive Control of Thought - Rational (ACT-R)

Adaptive Control of Thought - Rational (or ACT-R as it is commonly known) is a particular cognitive architecture first presented by John R. Anderson in 1983 [2] and further developed by Anderson *et al.* in [3].

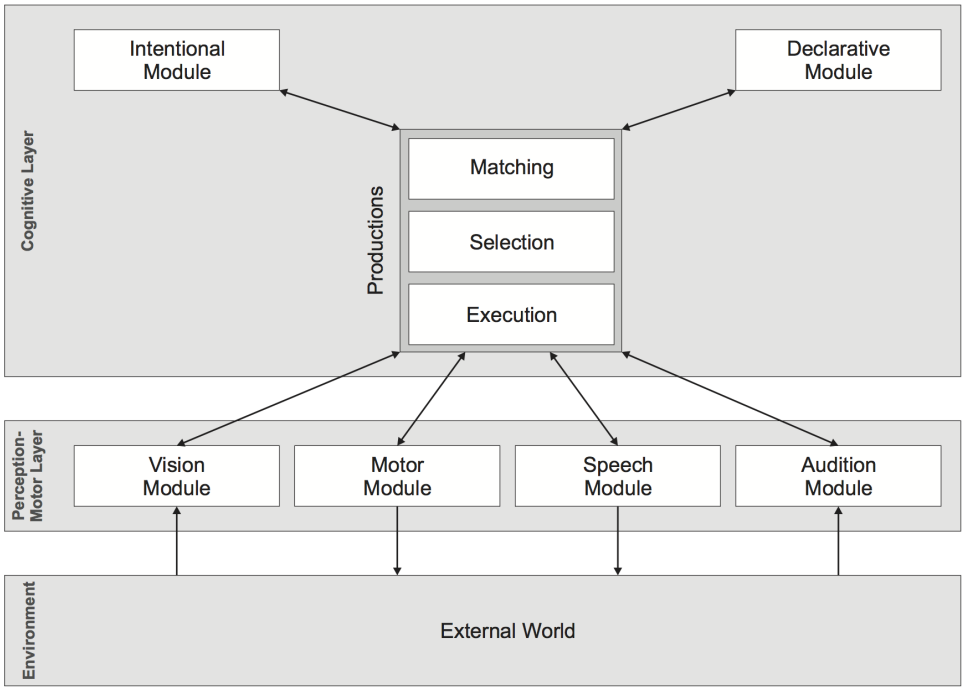


Figure 2.1 Overview of the ACT-R high level architecture (from [5]).

The architecture can be generally described as two distinct modules: a *perceptual-motor* layer and a *cognitive* layer, as presented in Figure 2.1. The perceptual-motor module corresponds to the interface of the cognition with the environment (which plays a key role in ACT-R), being comprised of sub-modules such as vision and motor actions. The cognitive layer is focused on memory, which can be divided into two different categories: *declarative* (consisting of factual knowledge and goals - e.g. *"The maximum driving speed in a typical UK motorway is 70 mph"* or *"Try get to point*

B") and *procedural* (consisting of rules/procedures - e.g. *"If the lead vehicle is going slowly, attempt an overtake"*) [5]. Furthermore, declarative memory is composed of *chunks*, which are smaller pieces of information divided into categories or attributes that form more complex memories. In the example *"The maximum driving speed in a typical UK motorway is 70 mph"*, `maximum driving speed in a typical UK motorway` is a category, whereas `70 mph` is an attribute. Another possible attribute for this category could be `112 km/h` (since $70mph = 112km/h$).

2.2 Integrated Driver Modelling in ACT-R

The first integrated approaches to this problem, proposed in [18, 6, 23], rely on modelling human behaviour through continuous controllers and specifically engineered frameworks. However, these methods lack the human behaviour variability attributed to the discrete nature of the control actions performed by the drivers. With this in mind, in [21] Salvucci *et al.* proposed a proof-of-concept of the introduction of human constrains in the driver model through the use of the cognitive architecture ACT-R. In [22], Salvucci proposed an updated version of the human driver model initially introduced in [21], which was improved using the advances in ACT-R (the model uses version 5.0 of the architecture) and re-designed elements (e.g. lane changing behaviour or distraction) resulting from validation performed by empirical studies.

The model Salvucci introduced in [22] was developed with a specific scenario in mind: *"(...) driving a standard midsize vehicle on a multilane highway with moderate traffic"* [22]. It consists of three distinct modules interacting in a sequential way: **control**, **monitoring**, and **decision making**. A high level schematic of the model can be found in Figure 2.2.

The **control** component manages both the lower level perception cues and the physical manipulation of the vehicle (e.g. steering, accelerating or breaking). The module can be divided into lateral (i.e. steering) and longitudinal (i.e. speed) control, each modelled by a separate control law.

The lateral control is determined by the existence of two artefacts that the driver obtains using lower level perception cues: the *near point* and the *far point*. In this model, *"the near point represents the vehicle's current lane position, used to judge how close the vehicle is to the center of the roadway (...) [and] is characterized as a*

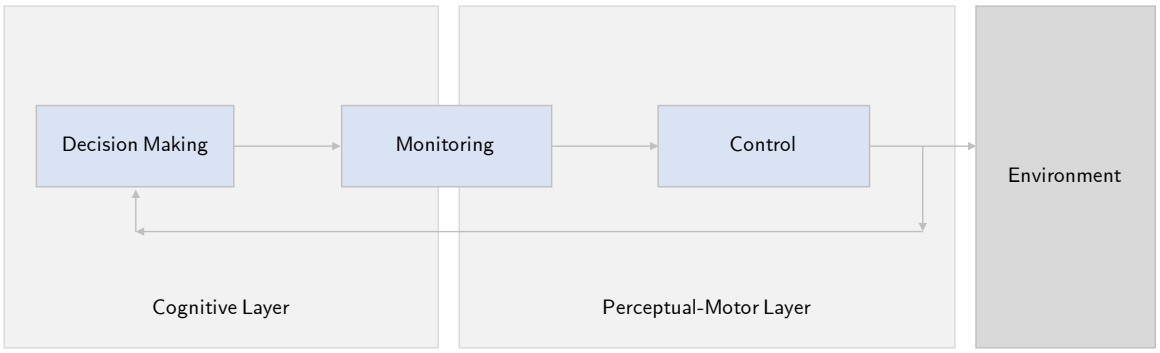


Figure 2.2 High level representation of the architecture presented in [22].

point in the center of the near lane visible in front of the vehicle, set at a distance of 10 m from the vehicle's center" while *"the far point indicates the curvature of the upcoming roadway, used to judge what the driver should execute to anticipate the upcoming curvature and remain in the current lane* [22]. At any cycle $i > 0$, the model works by using perception to determine the visual angles $\theta_{near}^{(i)}$ and $\theta_{far}^{(i)}$ and calculates:

$$\begin{aligned} \Delta\theta_{near} &= \theta_{near}^{(i)} - \theta_{near}^{(i-1)} \\ \Delta\theta_{far} &= \theta_{far}^{(i)} - \theta_{far}^{(i-1)} \end{aligned} \tag{2.1}$$

The control law for the steering angle φ can be defined as:

$$\Delta\varphi = k_{far}\Delta\theta_{far} + k_{near}\Delta\theta_{near} + k_I \min\left(\theta_{near}^{(i)}, \theta_{\max}\right)\Delta t \tag{2.2}$$

with k_{far}, k_{near}, k_I and $\theta_{\max}$ defined as in [22].

The process for the longitudinal control is very similar. At any cycle $i > 0$, the model starts by encoding the position of the lead vehicle and calculating the time headway $thw_{car}^{(i)}$ to it. It then computes:

$$\Delta thw_{car} = thw_{car}^{(i)} - thw_{car}^{(i-1)} \tag{2.3}$$

The control law for the linear acceleration ψ can be written as:

$$\Delta\psi = k_{car}\Delta thw_{car} + k_{follow}(thw_{car} - thw_{follow})\Delta t \tag{2.4}$$

with k_{car}, k_{follow} and thw_{car} defined as in [22].

These control laws can be applied in order to model the behaviour of the driver in terms of lane keeping and curve negotiation, and generalise to lane changing actions in a straightforward manner. In order to initiate a lane change, the driver just starts following the near and far points of the destination lane instead of the current lane, as Salvucci and Liu presented in [23].

The **monitoring** component maintains situational awareness through the awareness of the position of other vehicles around the driver's vehicle. It accomplishes this through a random-sampling, with probability $p_{monitor}$, of one of four areas: either the left or right lane, and either forward or backward (with equal probability). After deciding which area to sample (if any), the model uses visual perception to determine whether a vehicle is present or not. If it is, the distance, lane and direction are saved in ACT-R's declarative memory. The value of $p_{monitor}$ is defined in [22].

Finally, the **decision making** module uses the information gathered in the monitoring and control stage (if any is available) to determine which tactical decision should be taken. In this model, the decision corresponds to determining when and where a lane change should occur. Salvucci describes this process in [22] as: *"If the driver's vehicle is in the right lane, the model checks the current time headway to the lead vehicle (if any); if the lead car time headway drops below a desired time headway thw_{pass}, the model decides to change lanes to pass the vehicle. If the driver vehicle is in the left lane, the model checks instead simply for the presence of a lead vehicle. If there is a lead vehicle, the model remains in the left lane (because this vehicle is also passing other vehicles); otherwise it decides to [try to] change lanes to return to the right lane"*. This process is summarised in the flowchart presented in Figure 2.3, and the value of thw_{pass} is defined in [22].

Even after this decision to change lanes is taken, the model must then determine whether it is actually safe to do so. In order to verify that it is, the model initially attempts to recall, from the declarative memory, the nearest vehicle to it in the destination lane. If one is recalled at a distance closer than d_{safe}, then the change is aborted. If not, then the vehicle performs a safety monitoring of the destination lane.

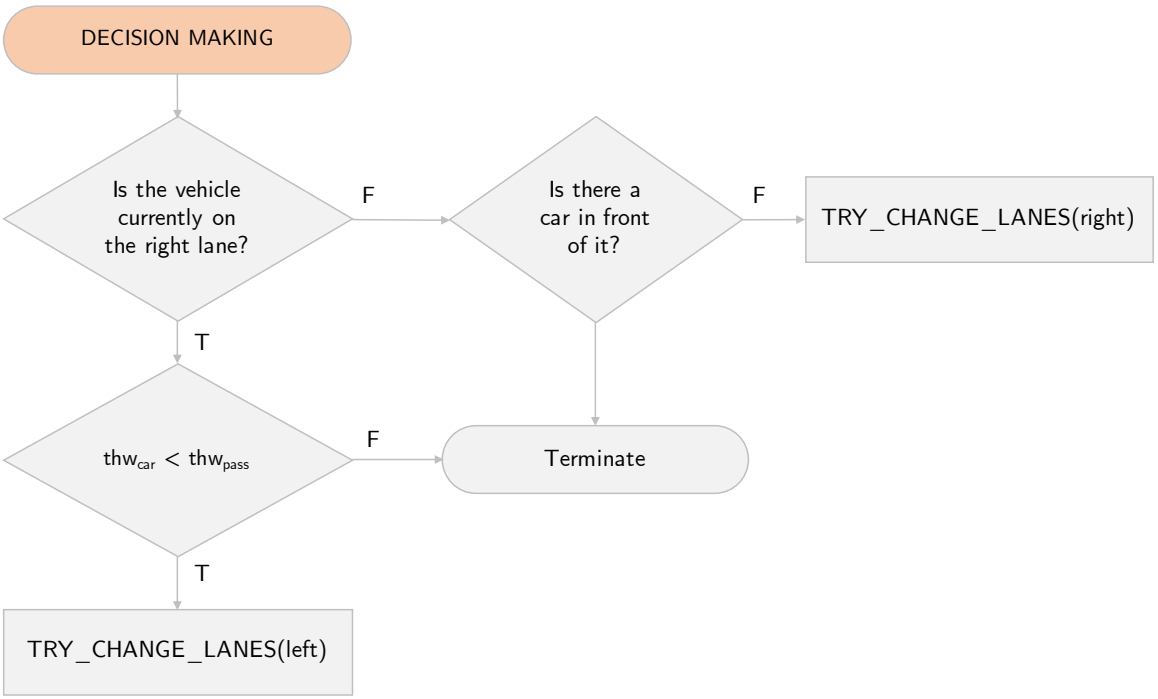


Figure 2.3 Flowchart of part of the decision making process of the model.

If this monitoring does not observe a vehicle at a distance closer than d_{safe}, then the vehicle initiates the execution of the lane change. The parameter d_{safe} is as defined in [22].

2.3 Probabilistic Model Checking

Model checking is a formal verification technique consisting of the process of automatically verifying, through exhaustive search, whether a model meets a given specification [4]. A model in this context is considered to be a labelled finite state-transition system, with states corresponding to configurations of the system and transitions between states representing the evolution between the said configurations.

Probabilistic model checking is a generalisation of model checking for the automated verification of systems that exhibit probabilistic behaviour [11, 4]. It should be noted that this is not to be confused with *probabilistic verification*, which amounts to partial state-space exploration [4]. Within this section, important concepts of probabilistic model checking related to modelling of systems, and verification and synthesis of models will be explained in detail.

2.3.1 Markov Chains and Decision Processes

Probabilistic model checking begins with the appropriate modelling of the system in a stochastic finite state-transition representation. There are several ways to do this according to the type of system to be modelled.

In *discrete-time Markov Chains* (DTMCs), all choices (i.e. transitions) are probabilistic [4]. While this is useful in some cases, others require nondeterminism, which is where *Markov Decision Processes* (MDPs) come in. In MDPs, nondeterministic probabilistic choices coexist. Essentially, a DTMC is an MDP with a uniquely determined probabilistic distribution [4]. Both DTMCs and MDPs can be augmented with reward structures, which correspond to real positive values that can be interpreted as bonuses or costs [4].

The formal definitions of DTMCs, MDPs and reward structures follow (adapted from [4, 11, 14]).

Definition 1. Discrete-time Markov Chain (DTMC)

A *discrete-time Markov chain* is a tuple $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ such that:

- S is a countable, non-empty set of states
- $\mathbf{P} : S \times S \to [0, 1]$ is a transition probability function such that for all states $s \in S$:
$$\sum_{s' \in S} \mathbf{P}(s, s') = 1 \tag{2.5}$$
- $p_{init} : S \to [0, 1]$ is the initial state distribution, such that:
$$\sum_{s \in S} p_{init}(s) = 1 \tag{2.6}$$
- AP is a set of atomic propositions and $L : S \to 2^{AP}$ is a labelling function.

$\mathcal{M}$ is denoted *finite* if S and AP are finite sets. ■

Definition 2. Markov Decision Process (MDP)

A *Markov decision process* is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ such that:

- S is a countable, non-empty set of states
- Act is a set of actions
- $\mathbf{P} : S \times Act \times S \to [0, 1]$ is a transition probability function such that for all states $s \in S$ and actions $\alpha \in Act$:

$$\sum_{s' \in S} \mathbf{P}(s, \alpha, s') \in \{0, 1\} \tag{2.7}$$

- $p_{init} : S \to [0, 1]$ is the initial state distribution, such that:

$$\sum_{s \in S} p_{init}(s) = 1 \tag{2.8}$$

- AP is a set of atomic propositions and $L : S \to 2^{AP}$ is a labelling function.

An action $\alpha \in Act$ is enabled in a state s if, and only if, $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$. Let $Act(s)$ denote the set of actions enabled in a state s. It must be that for all states $s \in S$, $Act(s) \neq \emptyset$. ■

Definition 3. Reward Structures for DTMCs and MDPs

A *reward structure* for a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ is a tuple $\mathcal{C} = (\rho, \iota)$ such that:

- $\rho : S \to \mathbb{R}_{\geq 0}$ is a state reward function
- $\iota : S \times S \to \mathbb{R}_{\geq 0}$ is a transition reward function

A *reward structure* for a Markov decision process $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ is a tuple $\mathcal{C} = (\rho, \iota)$ such that:

- $\rho : S \to \mathbb{R}_{\geq 0}$ is a state reward function

- $\iota : S \times Act \to \mathbb{R}_{\geq 0}$ is an action reward function

■

There are some additional concepts related to DTMCs and MDPs which are relevant to the understanding of temporal logics and model checking. These are the concepts of paths in Markov chains or decision processes, and adversary in a Markov decision process (adapted from [4, 11]).

Definition 4. Paths in DTMCs and MDPs

An *(infinite) path* π in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ is defined as an infinite sequence of states $\pi = s_0 \to s_1 \to s_2...$ (also written as $\pi = s_0 s_1 s_2...$) such that $\forall i \geq 0 : \mathbf{P}(s_i, s_{i+1}) > 0$.

A *finite path* ρ in the discrete-time Markov chain $\mathcal{M}$ is defined as the prefix of an infinite path π, $\rho = s_0 \to s_1 \to ... \to s_n$ for $n > 0$.

An *(infinite) path* π in a Markov decision process $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ is defined as an infinite sequence of states and actions $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2...$ (also written as $\pi = s_0 a_0 s_1 a_1 s_2...$) such that $\forall i \geq 0 : \mathbf{P}(s_i, a_i, s_{i+1}) > 0$.

A *finite path* ρ in the Markov decision process $\mathcal{M}$ is defined as the prefix of an infinite path π, $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} s_n$ for $n > 0$.

In both Markov chains and decision processes, the set of all infinite paths is denoted by $Paths(\mathcal{M})$, while the set of finite paths is denoted by $Paths_{fin}(\mathcal{M})$ ■

Definition 5. Adversary (alternatively Strategy or Scheduler)

An *adversary* for an MDP $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ is a function $\sigma : S^+ \to Act$ such that $\sigma(s_0 s_1 ... s_n) \in Act(s_n)$ for all $s_0 s_1 ... s_n \in S^+$.

An adversary σ is called memoryless if, and only if, for any $\pi_1 = s_0 s_1 ... s_n$ and $\pi_2 = s'_0 s'_1 ... s_n$ it is true that $\sigma(\pi_1) = \sigma(\pi_2) = \sigma(s_n)$. ■

2.3.2 Temporal Logics

In order to verify properties in models, there needs to be a way of expressing them formally and unequivocally. While some reachability and reward-based properties come directly from the definition of DTMCs, MDPs and reward structures [11], more interesting and complex properties require the use of a temporal logic. A propositional temporal logic is essentially an extension of propositional logic by temporal operators [4]. In this context, one can consider time as being *linear* or *branching*. In linear time, at every moment in time there must only exist a single successor moment, whereas from the branching perspective, there may exist multiple ones, forming a tree-like structure of alternatives. Linear Temporal Logic (LTL) is a temporal logic which assumes the first, while Computation Tree Logic (CTL) is based on the branching perspective. Probabilistic Computational Tree Logic (PCTL) is an extension of CTL that takes into account a probabilistic operator.

Given the definition of a path in an MC and MDP, it is possible to define the syntax for LTL, CTL and PCTL (adapted from [4, 11]).

Definition 6. Syntax and Semantics of LTL

An *LTL formulae* φ over a set of atomic propositions AP can formed according to the following grammar:

$$\varphi ::= \texttt{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathrm{X}\varphi \mid \varphi_1 \ \mathrm{U} \ \varphi_2 \tag{2.9}$$

where $a \in AP$, X is the next operator and U is the until operator as defined below for Markov chains and decision processes.

For a given path $\pi = s_0 s_1 s_2 ...$ in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ (generalising trivially to a Markov decision process):

- $\pi \models \texttt{true}$ always.
- $\pi \models a$ if, and only if, $a \in L(s_0)$.
- $\pi \models \varphi_1 \wedge \varphi_2$ if, and only if, $\pi \models \varphi_1 \wedge \pi \models \varphi_2$.
- $\pi \models \neg\varphi$ if, and only if, $\pi \not\models \varphi$.

- $\pi \models \mathrm{X}\varphi$ if, and only if, $s_1 s_2 ... \models \varphi$.
- $\pi \models \varphi_1 \ \mathrm{U} \ \varphi_2$ if, and only if:

$$\exists i \geq 0 \text{ s.t. } (s_i s_{i+1} ... \models \varphi_2) \wedge (\forall j < i : s_j s_{j+1} ... \models \varphi_1) \tag{2.10}$$

The operator F (eventually) can also be defined as:

$$\mathrm{F}\varphi = \texttt{true} \ \mathrm{U} \ \varphi \tag{2.11}$$

■

It can be noticed that, as a result of the linear time consideration in LTL, all the formulas are taken over paths. This is not the case in CTL which can have both path and state formulae, as shown below.

Definition 7. Syntax and Semantics of CTL

A *CTL state formulae* Φ over a set of atomic propositions AP can formed according to the following grammar:

$$\Phi ::= \texttt{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathrm{E}\varphi \mid \mathrm{A}\varphi \tag{2.12}$$

where $a \in AP$ and φ is a path formula.

For a given state $s \in S$ in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ (generalising trivially to a Markov decision process):

- $s \models$`true` always.
- $s \models a$ if, and only if, $a \in L(s)$.
- $s \models \Phi_1 \wedge \Phi_2$ if, and only if, $s \models \Phi_1 \wedge s \models \Phi_2$.
- $s \models \neg\Phi$ if, and only if, $s \not\models \Phi$.
- $s \models \mathrm{E}\varphi$ if, and only if, $\exists \pi \in Paths(s) : \pi \models \varphi$.

- $s \models \mathrm{A}\varphi$ if, and only if, $\forall \pi \in Paths(s) : \pi \models \varphi$

A *CTL path formulae* φ over a set of atomic propositions AP can formed according to the following grammar:

$$\varphi ::= \mathrm{X}\Phi \mid \Phi_1 \ \mathrm{U} \ \Phi_2 \tag{2.13}$$

where $a \in AP$ and Φ is a state formula.

For a given path $\pi = s_0 s_1 s_2 ...$ in a Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ (generalising trivially to a Markov decision process):

- $\pi \models \mathrm{X}\Phi$ if, and only if, $s_1 \models \Phi$.
- $\pi \models \Phi_1 \ \mathrm{U} \ \Phi_2$ if, and only if:

$$\exists i \geq 0 \text{ s.t. } (s_i \models \Phi_2) \wedge (\forall j < i : s_j \models \Phi_1) \tag{2.14}$$

It is also possible to define the operators F (eventually) and G (globally) as:

$$\begin{aligned} \mathrm{F}\Phi &= \texttt{true} \ \mathrm{U} \ \Phi \\ \mathrm{G}\Phi &= \neg \mathrm{E}(\neg \Phi) \end{aligned} \tag{2.15}$$

■

Definition 8. Syntax and Semantics of PCTL

A *PCTL state formulae* Φ over a set of atomic propositions AP can formed according to the following grammar:

$$\Phi ::= \texttt{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid \mathbb{P}_{\triangleright p}(\varphi) \tag{2.16}$$

where $a \in AP$, $p \in [0, 1]$, $\triangleright \in \{>, <, \geq, \leq\}$ is a probability bound and φ is a path formula.

For a given state $s \in S$ in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ (generalising trivially to a Markov decision process):

- $s \models \texttt{true}$ always.
- $s \models a$ if, and only if, $a \in L(s)$.
- $s \models \Phi_1 \wedge \Phi_2$ if, and only if, $s \models \Phi_1 \wedge s \models \Phi_2$.
- $s \models \neg\Phi$ if, and only if, $s \not\models \Phi$.
- $s \models \mathbb{P}_{\triangleright p}(\varphi)$ if, and only if, $P_s(\pi \in Paths(s) : \pi \models \varphi) \triangleright p$ (that is, the probability, from state s, that φ is true for an outgoing path satisfies $\triangleright p$).

A *PCTL path formulae* φ over a set of atomic propositions AP can formed according to the following grammar:

$$\varphi ::= \mathrm{X}\Phi \ \mid \ \Phi_1 \ \mathrm{U}^{\leq k} \, \Phi_2 \ \mid \ \Phi_1 \ \mathrm{U} \ \Phi_2 \tag{2.17}$$

where $a \in AP$, $k \in \mathbb{N}$ and Φ is a state formula.

For a given path $\pi = s_0 s_1 s_2 ...$ in a discrete-time Markov chain $\mathcal{M} = (S, \mathbf{P}, p_{init}, AP, L)$ (generalising trivially to a Markov decision process):

- $\pi \models \mathrm{X}\Phi$ if, and only if, $s_1 \models \Phi$.
- $\pi \models \Phi_1 \ \mathrm{U}^{\leq k} \, \Phi_2$ if, and only if:

$$\exists i \leq k \text{ s.t. } (s_i \models \Phi_2) \wedge (\forall j < i : s_j \models \Phi_1) \tag{2.18}$$

- $\pi \models \Phi_1 \ \mathrm{U} \ \Phi_2$ if, and only if:

$$\exists i \geq 0 \text{ s.t. } (s_i \models \Phi_2) \wedge (\forall j < i : s_j \models \Phi_1) \tag{2.19}$$

Similarly to CTL, it is possible to define the operators F (eventually) and G (globally) as:

$$\begin{aligned} \mathrm{F}\Phi &= \texttt{true} \ \mathrm{U} \ \Phi \\ \mathrm{G}\Phi &= \neg\mathrm{E}(\neg\Phi) \end{aligned} \tag{2.20}$$

■

While PCTL path formulas are defined above, it should be mentioned that all PCTL formulas are state formulas, and that path formulas should only appear inside the probabilistic operator.

In relation to the probabilistic operator in PCTL, the generalisation from the Markov chain to the Markov decision process can be done using the concept of adversary (also denoted strategy or scheduler), initially introduced in Section 2.3.1. In the case of an MDP, the operator $\mathbb{P}_{\triangleright p}$ should be interpreted under the following semantics:

$$s \models \mathbb{P}_{\triangleright p}(\varphi) \text{ if, and only if, } \forall \sigma \in Adv : Pr_s^{\sigma}(\pi \in Paths^{\sigma}(s) : \pi \models \varphi) \triangleright p \qquad (2.21)$$

that is, the probability, from state s, that φ is true for an outgoing path satisfies $\triangleright p$ for all adversaries $\sigma \in Adv$ (with Adv constituting the set of all adversaries that the MDP admits).

2.3.3 PRISM Language

PRISM is a probabilistic model checking tool developed in cooperation between the Universities of Oxford and Birmingham [15]. It was one of the first tools within stochastic model checking to be released and it is currently maintained by David Parker. In order to use PRISM (and other alternative model checking techniques), it is essential to be able to specify the model. To achieve this, the team behind PRISM developed the PRISM Language, initially described in [13].

A model described in the PRISM Language (typically a `.pm`, `.nm` or `.prism` text file) should contain a keyword that indicates the kind of model to expect. For example, the keyword `dtmc` indicates the model is a DTMC (discrete-time Markov chain) and `mdp` indicates a MDP (Markov decision process).

The PRISM language uses two different basic components to define a model *modules* and *variables*. A module is composed of a set of local variables whose behaviour is described by *commands*. A general command is of the form:

```
[] guard -> prob_1: update_1 + ... + prob_n: update_n;
```

where *guard* is a boolean expression over the variables, *prob_i* is the probability that *update_i* will be taken (with $\sum_i$ prob_i = 1) and *update_i* is the the new value of local and/or global variables if the guard is true and a transition is taken to this configuration.

A state of the module is defined as a configuration of its local variables, and a state of the model is a configuration of the states of all modules which compose it and the global variables.

In Listing 2.1, an example of a simple MDP model described in PRISM Language is shown. This example is comprised of two modules, `M1` and `M2`, each with their own local variables, and no global variables are declared. It should be noted that the variables in PRISM Language should either be integers (`int`) or booleans (`bool`) and can be initialised by specifying `init` after the initial declaration. Due to the fact that we are dealing with finite models, the integers must be in a finite range as well.

```
// Example 1 (modified)
// Two process mutual exclusion

mdp

module M1

    x : [0..2] init 0;

    [] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
    [] x=1 & y!=2 -> (x'=2);
    [] x=2 -> 1:(x'=2);
    [] x=2 -> 1:(x'=0);

endmodule

module M2

    y : [0..2] init 0;

    [] y=0 -> 0.8:(y'=0) + 0.2:(y'=1);
    [] y=1 & x!=2 -> (y'=2);
    [] y=2 -> 1:(y'=2);
    [] y=2 -> 1:(y'=0);

endmodule
```

Listing 2.1 Example of a simple MDP model from [1]

Models in the PRISM Language can also include *formulas*, which are expressions that avoid repetition of code, and *labels*, which are boolean expressions that identify sets of states of interest.

It is also possible to define rewards structures using the following convention:

```
rewards "name"
    guard: state_reward;
    ...
    [action] guard: transition_reward;
    ...
endrewards
```

where the *guards* are similarly defined as before, but the distinction between state and transition rewards is defined by the *action* (which may be empty).

2.3.4 Model Checking in DTMCs and MDPs

After the description of the model using a language such as the PRISM Language, it is possible to verify properties on the said model. These properties can be of different types according to the model and the model checking tool chosen.

In this dissertation, the models built will be represented through discrete-time Markov chains and Markov decision processes, so the properties of interest tend to be best expressed in the temporal logics LTL or PCTL. The PRISM model checker is one of the tools which allows a user to import a text file with the probabilistic model, build it and verify LTL and PCTL properties on it [15]. Another tool used for this purpose is the Storm model checker [10]. Each of these tools has it strengths and weaknesses. PRISM is an extremely robust tool, with lots of options in terms of engines to perform the required calculations, and a graphical interface. Storm is a relatively recent tool which builds on top of some of PRISM's features and introduces new ones, such as an optimised Sparse engine (a great advantage when dealing with models with a small state space) and conditional properties in DTMCs [10]. However, it does not provide a graphical interface and is limited in terms of capabilities and I/O. Due to the fact that Storm builds on top of PRISM, it uses the PRISM Language for model input and the properties are specified using the same convention.

In order to illustrate properties, consider the example of Listing 2.1. A PCTL property in this case could be defined as:

```
P>=0.5 [F x = 2]
```

This property can be interpreted as *"with probability greater or equal to 0.5, eventually the model reaches a state where* `x = 2`*"*. Building the model and verifying the property yields the value `false`, which means that, according to this model, there exists at least one adversary under which this property is not satisfied. Both PRISM and Storm allow for quantitative probabilistic properties as well. These properties take different formats in DTMCs and MDPs, as presented below.

```
// DTMC
P=? [F x = 2]

// MDP
Pmin=? [F x = 2]
Pmax=? [F x = 2]
```

Instead of returning a boolean value of satisfaction, these return the actual values of the probabilities of the properties being satisfied. They are, therefore, extremely useful metrics for model comparison. For the case of the DTMC, due to the fact that there is no nondeterminism, the probability of a path is uniquely defined (as seen in Section 2.3.2), and thus can be expressed simply as `P=?`. For MDPs, this is not the case. In order to verify the probabilistic operator $\mathbb{P}_{\triangleright p}$ on the path formula φ, a model checker needs to calculate the following quantities:

$$\begin{aligned} p_{\max,s}(\varphi) &= \sup_{\sigma \in Adv} P_s^\sigma(\varphi) \\ p_{\min,s}(\varphi) &= \inf_{\sigma \in Adv} P_s^\sigma(\varphi) \end{aligned} \tag{2.22}$$

where $P_s^\sigma(\varphi) = Pr_s^\sigma(\pi \in Paths^\sigma(s) : \pi \models \varphi)$.

Therefore, PRISM and Storm allow users to access these values by writing properties using `Pmax` and `Pmin` (boolean or quantitative).

Storm allows for conditional probability properties of the type:

```
P=? [ F x=2 || F y=2 ]
```

This property can be interpreted as *"what is the probability that the model eventually reaches the state* `x = 2`*, given that it eventually reaches the state* `y = 2`*"*. These properties, in DTMCs, correspond simply to the application of Bayes' theorem of conditional probability, which states that:

$$P_s(\varphi_1 \mid \varphi_2) = \frac{P_s(\varphi_1 \wedge \varphi_2)}{P_s(\varphi_2)} \tag{2.23}$$

For $P_s(\varphi_2) \neq 0$. It must be emphasised that this is **not** the case for MDPs, where the existence of multiple adversaries complicates the calculations of conditional probabilities.

This does not mean that it is not possible to reason over conditional probabilities in MDPs. For some specific scenarios, it is possible to obtain meaningful bounds which allow comparisons between the ones obtained in DTMCs and MDPs. One of those scenarios is defined in the proposition below, with the proof following it.

Proposition 2.3.1. Consider a Markov decision process $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$ which admits adversaries $\sigma \in Adv$. Let φ_1, φ_2 be two PCTL path formulas, such that, for any path $\pi \in Paths(s)$: $\pi \models \varphi_1 \Rightarrow \pi \models \varphi_2$. Assume as well that:

$$\sigma' \in \arg \sup_{\sigma \in Adv} P_s^{\sigma}(\varphi_1 \mid \varphi_2) \tag{2.24}$$

And that:

$$P_s^{\sigma'}(\varphi_1) = p_{\max,s}(\varphi_1) \tag{2.25}$$

That is, an adversary which maximises the probability of $\varphi_1 \mid \varphi_2$ also maximises the probability of φ_1. In this case it holds that:

$$p_{\max,s}(\varphi_1 \mid \varphi_2) \geq \frac{p_{\max,s}(\varphi_1)}{p_{\max,s}(\varphi_2)} \tag{2.26}$$

with:

$$p_{\max,s}(\varphi_1 \mid \varphi_2) = \sup_{\sigma \in Adv} P_s^{\sigma}(\varphi_1 \mid \varphi_2) \tag{2.27}$$

Proof. Consider $\sigma_2 \in Adv$ such that:

$$\sigma_2 \in \arg \sup_{\sigma \in Adv} P_s^{\sigma}(\varphi_2) \tag{2.28}$$

Assume, for the sake of contradiction, that:

$$p_{\max,s}(\varphi_1 \mid \varphi_2) < \frac{p_{\max,s}(\varphi_1)}{p_{\max,s}(\varphi_2)} \tag{2.29}$$

Under adversary σ', it can be written that, through Bayes' theorem of conditional probability:

$$p_{\max,s}(\varphi_1 \mid \varphi_2) = P_s^{\sigma'}(\varphi_1 \mid \varphi_2) = \frac{P_s^{\sigma'}(\varphi_1 \wedge \varphi_2)}{P_s^{\sigma'}(\varphi_2)} = \frac{P_s^{\sigma'}(\varphi_1)}{P_s^{\sigma'}(\varphi_2)} \tag{2.30}$$

Therefore, it follows that:

$$\frac{P_s^{\sigma'}(\varphi_1)}{P_s^{\sigma'}(\varphi_2)} < \frac{p_{\max,s}(\varphi_1)}{p_{\max,s}(\varphi_2)} \tag{2.31}$$

For this to be true, it must be that $P_s^{\sigma'}(\varphi_2) > p_{\max,s}(\varphi_2)$, since $P_s^{\sigma'}(\varphi_1) = p_{\max,s}(\varphi_1)$. This constitutes a contradiction, since in that case σ_2 would not maximise the probability of φ_2 ($p_{\max,s}(\varphi_2) = P_s^{\sigma_2}(\varphi_2)$, by definition), a violation of the assumption described in Equation 2.28. Thus, the proposition is proven. ■

In MDPs, however, it is also possible to perform multi-objective verification, whereas multiple properties will be tested simultaneously (both in PRISM and Storm). One example of this would be:

```
multi(P>=0 [F x=2], P>=0.2 [F y=2])
```

The value of the verification of this property corresponds to the boolean conjunction of the individual properties. It is also possible to optimise over a single property while constraining other properties, restricting the possibilities in terms of outcomes. An example of this would be:

```
multi(Pmax=? [F x=2], P>=0.2 [F y=2])
```

Finally, it is also possible to perform multi-objective optimisation over the properties by defining them as (for example):

```
multi(Pmax=? [F x=2], Pmax=? [F y=2])
```

This leads to a situation where compromises between the optimisation of either one of the properties might have to be reached. Pareto curves are the result of the verification of such properties [8].

Definition 9. Multi-objective Query, Pareto Vector and Pareto Curve (alternatively Set)

A *multi-objective query* (MQ) ϕ of n objectives is a positive boolean combination of n predicates of the form $r_i \geq v_i$, where r_i is a reward function, $v_i \in \mathbb{Q}$ is a bound. The notation $\phi[\mathbf{x}]$, $\mathbf{x} \in \mathbb{R}^n$ is used to denote ϕ in which each $r_i \geq v_i$ is replaced by $r_i \geq x_i$

Consider a multi-objective query MQ ϕ of n objectives. The vector $\mathbf{x} \in \mathbb{R}^n$ is a *Pareto vector* of ϕ if, and only if:

1. $\phi[\mathbf{x} - \varepsilon]$ is achievable for all $\varepsilon > 0$, and
2. $\phi[\mathbf{x} + \varepsilon]$ is not achievable for any $\varepsilon > 0$

A *Pareto curve* of ϕ is the set of all Pareto vectors that ϕ admits.

■

Thus, the result of the verification of a multi-objective optimisation property corresponds to a Pareto curve (this curve might contain a single vector). Both PRISM

and Storm allow for the verification of properties containing up to 2 objectives each. As a result, 2D plots are the best way to represent these solutions. In Figure 2.4, the result of the verification of the property `multi(Pmax=? [F G x=2], Pmax=? [F G y=2])` is shown. As it can be observed, either property can be satisfied with certainty, but not both simultaneously (mutual exclusion). The green area under the curve corresponds to the achievable set.

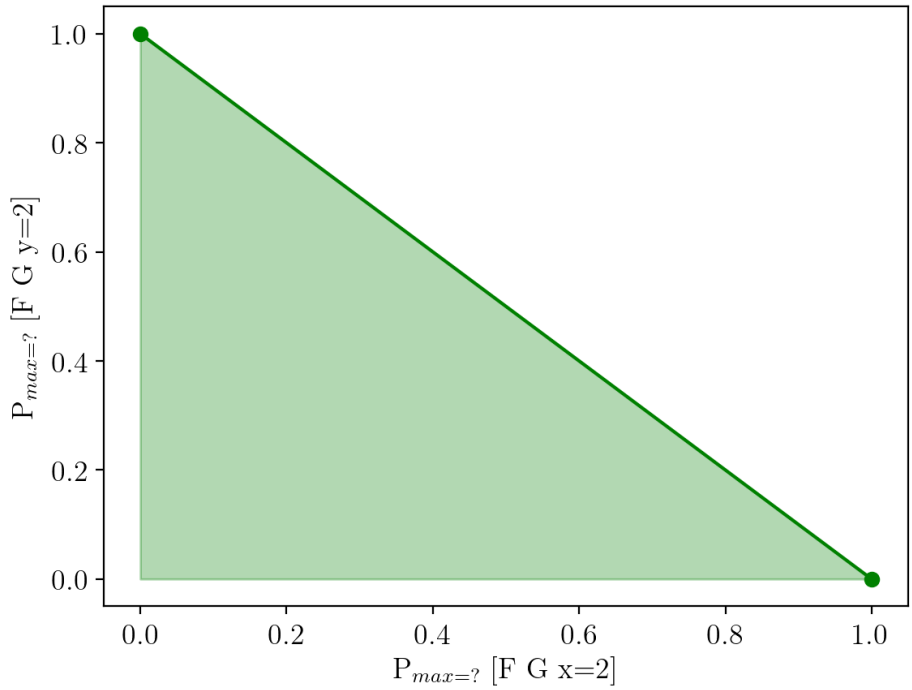


Figure 2.4 Example of a Pareto curve represented as a 2D plot.

Each of the values resulting of the verification of quantitative properties in MDPs (whether they be a single value from a single optimisation, or a Pareto vector) is the result of an adversary which solves the nondeterminism and converts the MDP into a DTMC [4]. The process of obtaining the adversary which satisfies certain properties is denoted by **strategy synthesis** or **adversary generation**, and it is formally defined below.

Definition 10. Strategy Synthesis (alternatively Adversary Generation)

Consider a Markov decision process $\mathcal{M} = (S, Act, \mathbf{P}, p_{init}, AP, L)$. The process of maximal (similarly minimal) *strategy synthesis* of the path formula φ is defined as a process that yields an adversary $\sigma \in Adv$ such that:

$$\sigma \in \arg\sup_{\sigma' \in Adv} P_s^{\sigma'}(\varphi) \tag{2.32}$$

(respectively $\arg\inf$ for minimal), where Adv is the set of all adversaries that $\mathcal{M}$ admits.

■

Chapter 3

Human Driver Modelling

In this chapter, the human driver model that Salvucci presents in [22] is implemented and the metrics used to verify the model are established. Initially, the model is implemented in *Matlab* based on the integrated driver model. To be able to verify it using the probabilistic model checking techniques presented in Chapter 2, the model undergoes an abstraction process, out of which a discrete-time Markov chain is obtained using several different approaches and assumptions. Metrics are then established to be able to evaluate whether the model meets the requirements and how well the human performs in a given situation. Finally, a simulator is presented in Section 3.4 in order to allow visualisation of paths in the model.

3.1 Continuous Driver Model using ACT-R

In order to obtain a continuous driver model, we interpret Salvucci's integrated driver model as presented in [22]. The scenario considered is the one Salvucci envisioned: a multilane highway with moderate (or low) traffic [22]. The model uses the three modules presented in Figure 2.2, that is, monitoring, decision making and control. All these modules rely on the constant update of the environment, as some are based on visual or low-level perception cues. While the human model is presented clearly in [22], the dynamics of the car (which is part of the environment) are left to the reader, as it is outside the scope of the paper. In that sense, some assumptions had to be made about the environment.

The main initial assumption is that the driver is perfectly aware of its surroundings,

and thus it is able to obtain the positions of other vehicles (within a certain distance) and the near and far points perfectly (this assumption is challenged later in the abstraction process). Additionally, the environment is assumed to change at the same rate as a cycle of the ACT-R model.

The overall view of the system is presented in Figure 3.1. The information flow is marked using the dotted arrows, while the sequential flow of the program is marked in the filled, lighter gray ones. Both the monitoring and the decision making make use of the perception which corresponds to querying the environment for the information (e.g. near and far point or another vehicle's position). The control both queries and sends updated information (e.g. vehicle position, velocity and acceleration) to the environment. Modules are updated sequentially, following the order: control, monitoring, decision making and environment (e.g. position, velocity and acceleration of other vehicles).

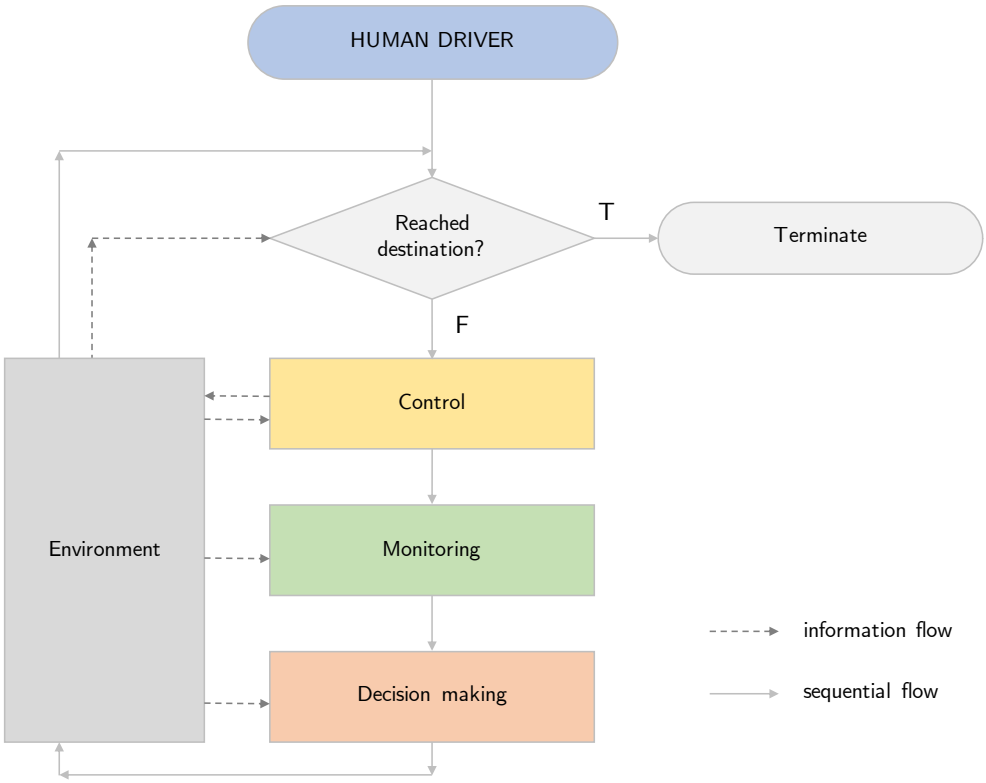


Figure 3.1 Continuous Driver Model overview.

The monitoring module is implemented according to the flowchart presented in Figure 3.2, from the description given in [22]. The threshold for monitoring is set at 0.2,

as this is the value suggested by Salvucci in [22]. The output of this process corresponds to altering the global variable a corresponding to the declarative memory cell composed of k chunks. No specific value for k is given by Salvucci in [22], but it is taken to be 8 from [16]. The function `GET_DISTANCE` is not described in depth due to the fact that it is trivial under the assumption stated in the previous paragraph.

The overall flowchart of the decision making module is shown in Figure 3.3. The high level part is similar to the flowchart presented in Figure 2.3, with small differences related to the loading of variables and information from the different memories available. It is worth noticing that the control makes use of the function `LOOK_VEHICLE` initially presented in the monitoring module to verify the presence of a vehicle in a relative position of a lane. The flow of the function `SET_LANE_FOLLOWING` is also omitted due to the trivial nature of this function under the assumptions made.

Finally, a flowchart for the control module is presented in Figure 3.4. This module is responsible for the calculation of $\Delta\varphi(t)$ and $\Delta\psi(t)$ and the update of the position. Given these two values, the position is updated following discrete laws of motion applied to rigid objects, particularly:

$$\begin{aligned} \mathbf{v}(t) &= \mathbf{v}(t-1) + \mathbf{a}(t)\Delta t \\ \mathbf{x}(t) &= \mathbf{x}(t-1) + \mathbf{v}(t-1)\Delta t + \frac{1}{2}\mathbf{a}(t)\Delta t^2 \\ t &= t + \Delta t \end{aligned} \tag{3.1}$$

where $\mathbf{x}(t) = (x(t), y(t))$, $\mathbf{v}(t) = (v_x(t), v_y(t))$, and $\mathbf{a}(t) = (a_x(t), a_y(t))$, with:

$$\begin{aligned} a_x(t) &= \Delta\psi(t) \\ a_y(t) &= \sin(\Delta\varphi(t)) \end{aligned} \tag{3.2}$$

In this case, as Salvucci mentions in [22], $\Delta t = 0.5s$. It is also assumed that the vehicle does go over or under the speed limits, and therefore, $v_x \in [15, 34]\ m/s$ (between $50km/h$ and $120km/h$ - typical limits in highway speeds).

Using these assumptions, the continuous driver model is implemented in *Matlab* as presented in Appendix A.1.

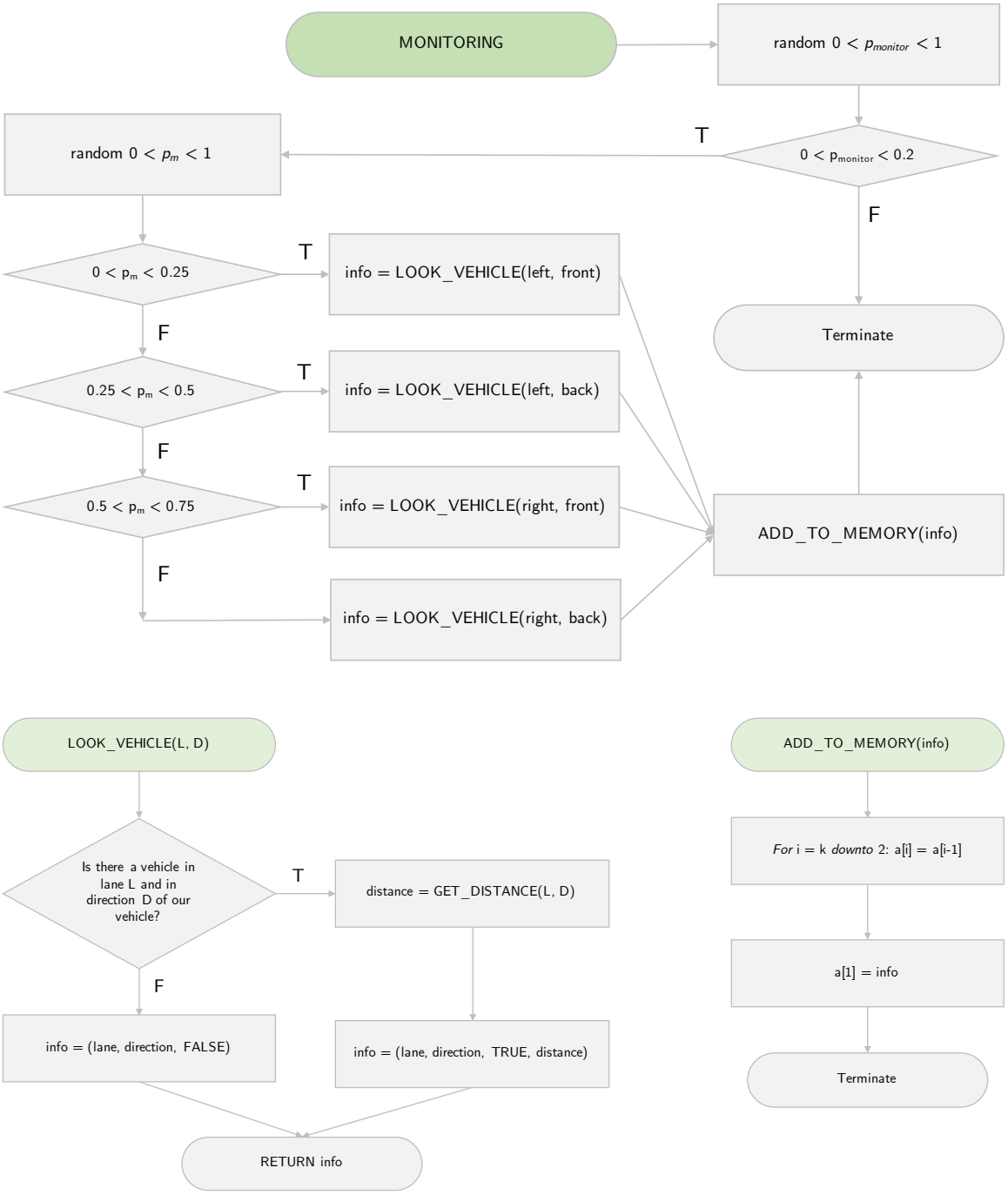


Figure 3.2 Flowchart of the Monitoring module.

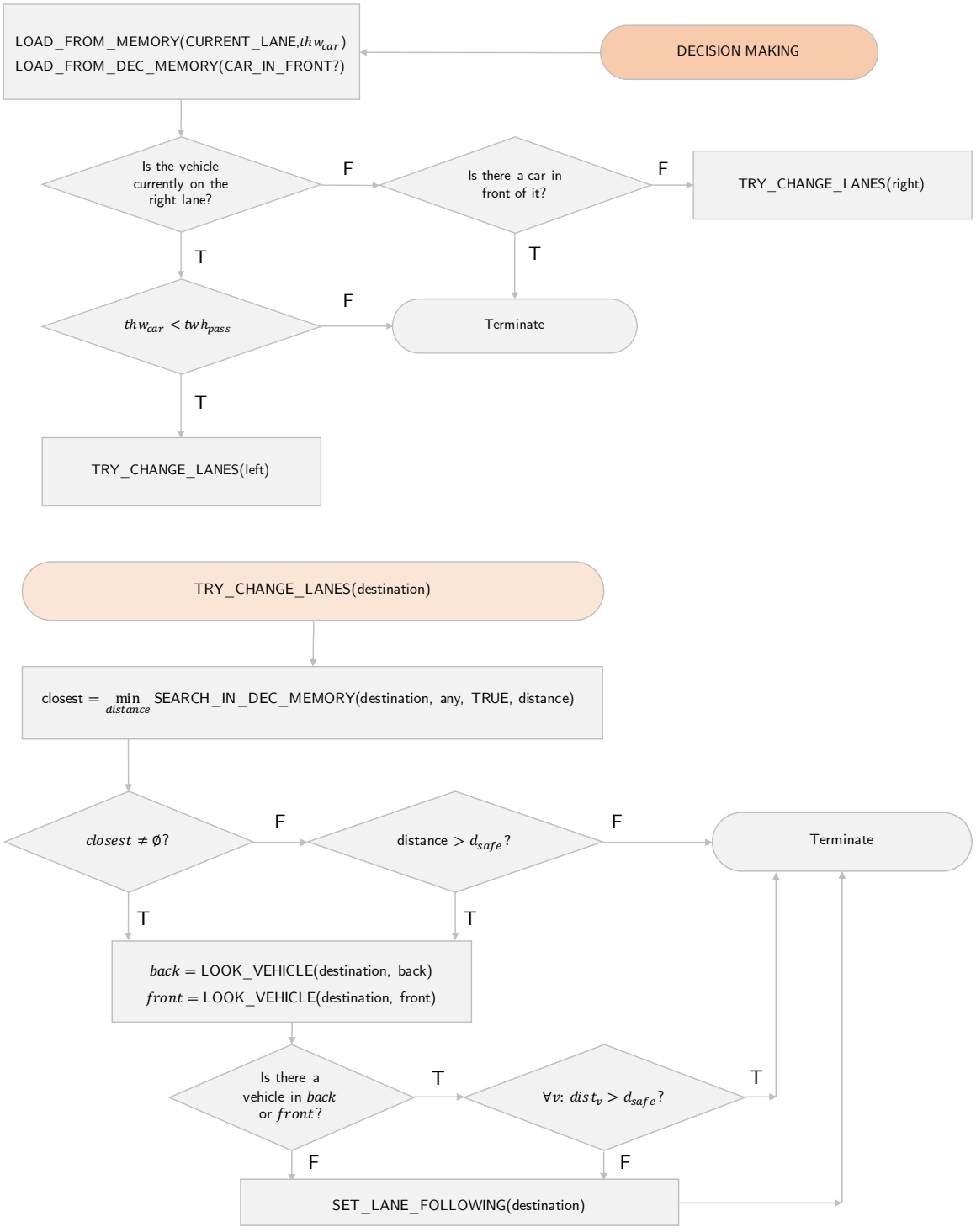


Figure 3.3 Flowchart of the Decision Making module.

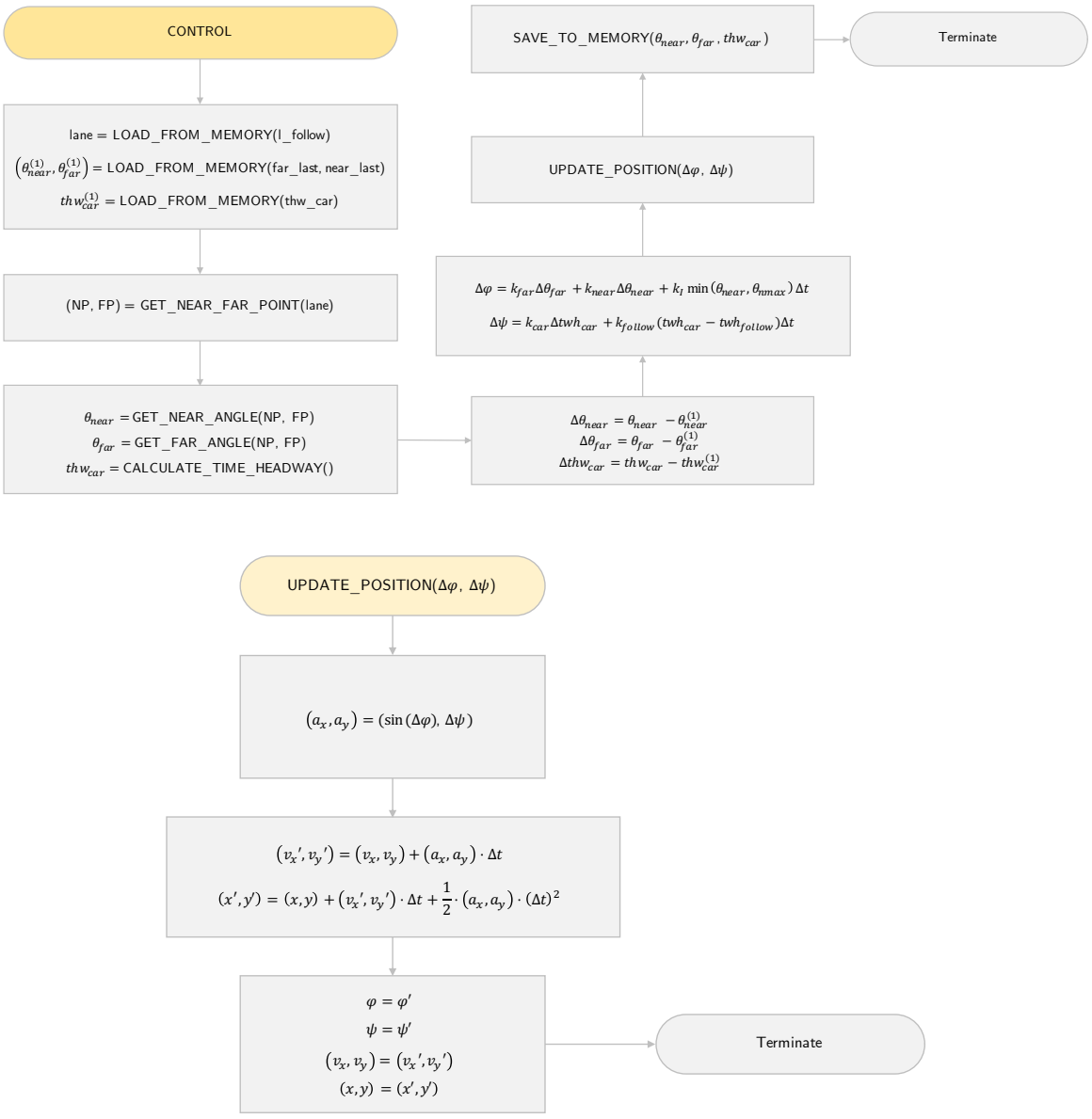


Figure 3.4 Flowchart of the Control module.

3.2 Model Abstraction

The process of abstraction consists in transforming the evolution of the continuous variables that constitute the model and discretising it in order to obtain a finite discrete model. In this case, the goal is to go from the continuous model of driver behaviour presented in the previous section to a discrete-time Markov chain (DTMC) which represents the same process. This transformation is less than trivial due to the fact that, in model checking of finite state abstractions, as the number of state variables in the system increases, the size of the system state space grows exponentially [9]. This problem is known as state explosion, and there has been some research into efficient ways of dealing with it [17, 20].

In the following sections, the abstraction process for the different modules of ACT-R is presented in detail, as well as the thought process behind the choices made. Finally, the unified model is presented using the different modules designed.

3.2.1 Non-Probabilistic Control Module

The control module, as presented in the previous section and following Salvucci's approach in [22], is fully deterministic, in the sense that there is no probabilistic reasoning involved. It makes use of perception (for the near and far points) and two simple control laws which influence the position, velocity and acceleration of the vehicle in the environment.

The straightforward approach to this problem would be to represent it as a simple $N \times M$ grid, in which a tuple (x, y) with $x \in \{0, ..., N\}$ and $y \in \{0, ..., M\}$ would describe the position of the vehicle, and all other variables (i.e. v_x, v_y, a_x, a_y and t) would be integer versions of its continuous model representation. In this approach, the control laws could be applied directly over the grid and the results would be the movement of the vehicle in it. However, there is an intrinsic problem with this approach. The error associated with the use of the grid as a way to discretise space would, logically, be reduced with an increase of N and M for a representation of the same road segment (i.e. increase in resolution). For example, the error associated with a road segment of $500 \times 7m^2$ represented by $N = 500$ and $M = 7$ (each grid square corresponds to $1m^2$) would be significantly greater than if $N = 5000$ and $M = 70$ (each grid square corresponds to 10^{-2} m^2). The conclusion would be that

one should increase N and M in order to obtain an appropriate resolution, but this incurs in the problem of state explosion presented in the beginning of the section: as the variable ranges increase, the number of states in the system grows exponentially. Thus, while the situation of $N = 5000$ and $M = 70$ is appealing in terms of error minimisation, it would be intractable to actually build and perform model checking on it (e.g. for $v_x, v_y \in \{15, ..., 34\}^2$ and $a_x, a_y \in \{-3, ..., 3\}^2$, the control module alone would have approximately 6.89×10^9 states, making it impossible to add decision making and monitoring on top of it).

The straightforward strategy is not, by any standards, efficient in terms of state space and does not use any of the information of the problem to simplify it. In particular, one should notice that, in this scenario, the movement in the y direction is used simply for the purpose of lane changing, a process which is purely deterministic (as described in [22]). Therefore, a more interesting approach to the abstraction of the control module could take this into account and **pre-simulate the lane changing operations**.

In this approach, the vehicle's position is represented by two discrete integer variables $x \in \{0, ..., length\}$ for a given $length$ and $lane \in \{right, left\}$ (can be extended to more than two lanes), and the acceleration and velocity of the vehicle are simply $a = a_x$ and $v = v_x$. Time is still represented in the same way, with $t \in \{0, ..., max\_time\}$. If a lane change is decided by the decision making module, then, **within one transition of the control**, the $lane$ variable is updated with the final destination lane, and the variables x, v, a and t are updated to reflect the obtained values after a lane change. These values are obtained from a look-up table and depend on the distance to the other vehicle (d) and the velocities of both the vehicle in question (v) and the other vehicle (v_1) (an obstacle on the road can be modelled by setting $v_1 = 0$). Additionally, the variable $crashed$ is used to represent whether the vehicle has collided in the process or not. The linear acceleration is implemented similarly using pre-simulation to determine the value of acceleration and the motion law is updated in the model itself.

The continuous model of driver behaviour obtained in the previous section in *Matlab* is used as the basis for the simulation of lane changes and linear acceleration. From this model, look-up tables are obtained which, given an origin lane (o_{lane}), a distance to the other vehicle and the velocities of both the vehicle in question and the other vehicle determines whether a crash happened or not, what is the Δx and ΔT incurred

(how did the position in x changed and how long did it take), and the final velocity of the vehicle in question. The code presented in Appendix A.2 corresponds to a similar version of the control (but probabilistic, as presented later in this section). An example of the simulation of the scenario $o_{lane} = 1$, $d = 20m$, $v = 15m/s$ and $v_1 = 15m/s$ is shown in Figure 3.5.

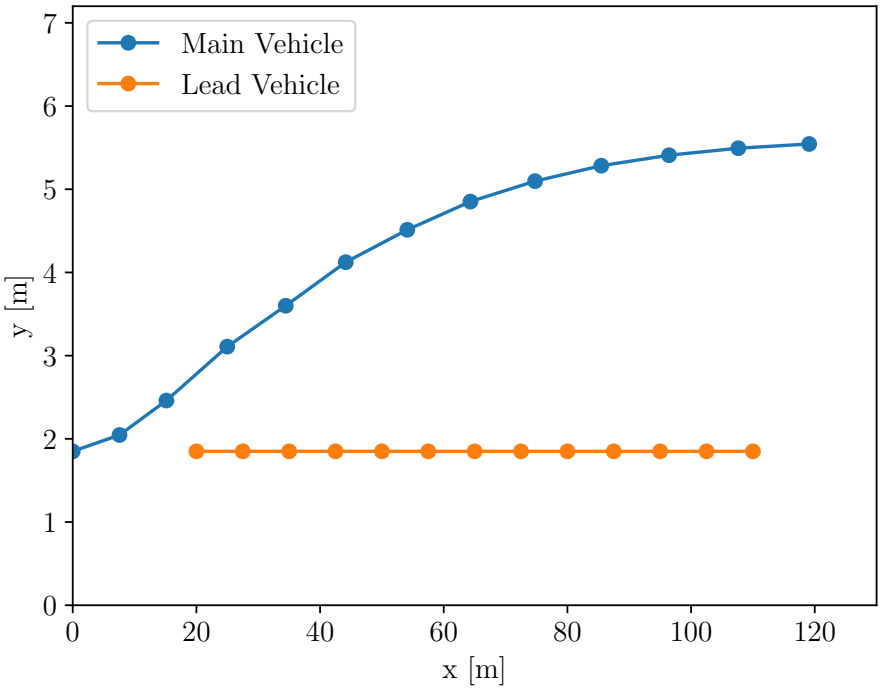


Figure 3.5 Example of the simulation of the lane change for $o_{lane} = 1$ (right lane), $d = 20m$, $v = 15m/s$ and $v_1 = 15m/s$. In this case, after the change is complete the variable assignments are $crashed = 0$ (no collision happened), $\Delta x = 119m$, $\Delta T = 6s$ and $v = 23m/s$.

This process focuses the computational effort on the pre-calculations, simplifying the final model significantly in terms of the state space, while removing the error made by discretising in the y direction of the grid (the error in the x direction is still present though). With regards to the size of the lane changing table, for the considered scenario of 2 lanes, $v, v_1 \in \{15, ..., 34\}^2$ and for any discrete $d \in \{1, ..., length\}$, the size of the table would be $2 \times 20^2 \times length = 800 \times length$. Given that the only value of the table that is effectively altered with a change in d is the value of $crashed$ (as the Δx, ΔT and final v depend solely on the initial velocities of both vehicles), it is possible to define $d_{\max}$ such that:

$$d_{\max} < length, \text{ s.t. } \forall v, v_1 \in \{15, ..., 34\}^2, d' \geq d_{\max} : crashed_{d'} = false \tag{3.3}$$

that is, $d_{\max}$ is a great enough value of d such that, regardless of the speed of the two vehicles, no crash will occur between them. With this change, it is possible to reduce the table from $800 \times length$ to $800 \times d_{\max}$ (since all other rows would be equal to the one for d_{max}). For the given ranges of v and v_1, $d_{\max}$ is determined to be $43m$, and thus the obtained look-up table contains $34,400$ rows.

The linear acceleration table depends on the thw_{car} which varies only with the distance to the other vehicle and the current velocity of the vehicle in question. Considering the distance in this case to be up to $80m$ (otherwise maximum acceleration will be applied), the look-up table generated has $1,600$ rows.

3.2.2 Decision Making and Monitoring Module

Due to the fact that, in the model presented by Salvucci in [22], the monitoring module influences the declarative memory which is then used exclusively for decision making purposes (and not for control), logically these two can be incorporated into one for abstraction purposes. Again, taking the straightforward approach of trying to include the declarative memory and the process associated with monitoring and decision making into the model directly would make it intractable.

In its simplest form, the decision making process for the right lane consists in localising the other vehicle and deciding whether or not to change lanes based on the time headway, thw_{car}, to the lead vehicle (if there is one). This value essentially corresponds to the time the vehicle in question has before it crashes into the lead vehicle, assuming that latter came to a full stop ($v = a = 0$) [19]. The lower this time headway, the more likely a driver is to perform the manoeuvre. While Salvucci presents a fully deterministic driver in [22] based on the average driver, in this dissertation it was decided to try and analyse drivers according to different profiles in order to simulate how different parts of the population of drivers might make decisions. The decision making in such a case follows from a stochastic reasoning which can be simulated and incorporated in the model using look-up tables.

In this case, for a driver in the right lane, it was decided to model the probability of the driver performing a lane change based on the time headway as an exponentially decreasing function, writing it as:

$$\mathrm{P}[lC = true]_{thw} = P_{lC}(d, v) = e^{-\alpha \cdot thw} = e^{-\alpha \cdot d/v} \tag{3.4}$$

where α is a parameter unique to each population of drivers.

The same logic can be applied for a driver in the left lane overtaking a vehicle behind it in the left lane, except in this case the opposite effect will be seen in the decision making. In such a case, the probability of changing lane can be modelled as a normalised logarithmic function over the distance of the vehicles (it doesn't make sense to define time headway in this context):

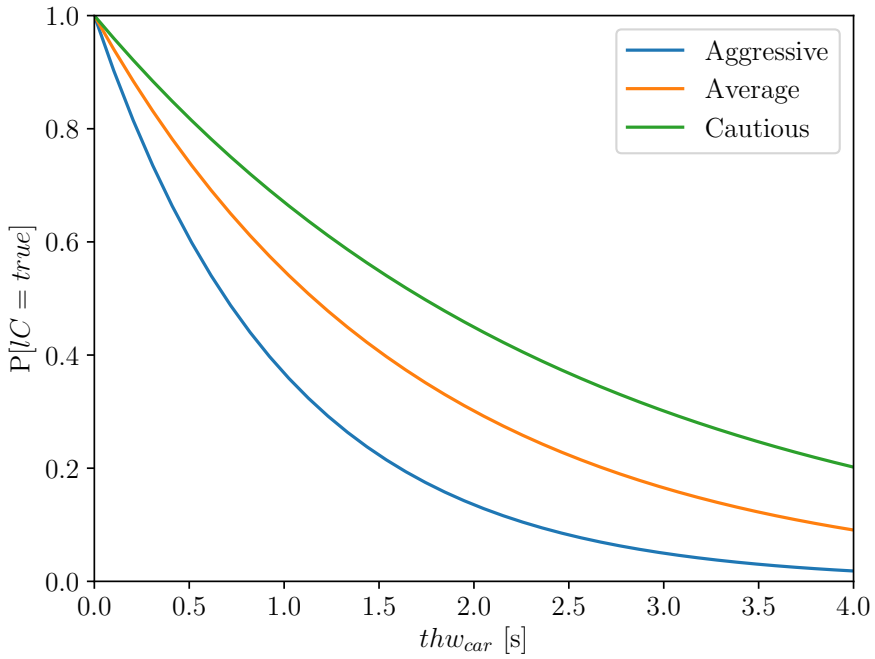
$$\mathrm{P}[lC = true]_{d} = P_{lC}(d) = \frac{1}{\log(\beta * \max_d + 1)} \cdot \log(\beta * d + 1) \tag{3.5}$$

where $\max_d$ is the maximum length considered and β is a parameter unique to each population of drivers.

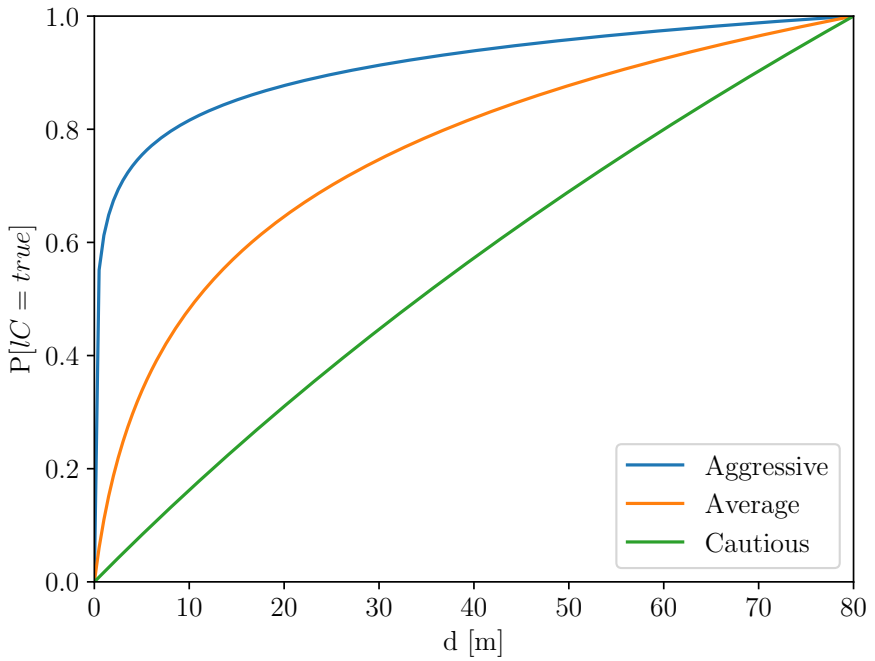
It should be noted that these functions, while logical, are an assumption of the abstraction, since the model presented by Salvucci in [22] does not make any reference to these parameterisations. It is also worth referring that, while there was no real-world data involved in this project, this function could have been learnt from real data without changing the fundamental paradigm of the abstraction.

For the purposes of analysis, three classes of drivers are considered: **Aggressive**, **Average** and **Cautious** drivers (with different values for the parameter α and β). In Figure 3.6, the functions are presented for the three profiles of drivers assumed in this project, for the lane changes originating in the right (Figure 3.6 (a)) and left lane (Figure 3.6 (b)).

So far, this version of decision making is not influenced by the monitoring module at all, and it relies on the correct calculation of the value of thw_{car} and d. This assumption is unrealistic, and in order to simulate these issues, it was decided that stochastic noise could be added to the measurement of the distance (as this is what



a. Aggressive ($\alpha = 1$), Average ($\alpha = 0.6$) and Cautious ($\alpha = 0.4$) drivers.



b. Aggressive ($\beta = 1000$), Average ($\beta = 0.5$) and Cautious ($\beta = 0.01$) drivers.

Figure 3.6 P[$lC = true$] as a function of thw_{car} and d for vehicles originating from the right lane (a) and left lane (b).

human drivers have to instinctively measure through perception). It was chosen that this noise, n, would be normally distributed as:

$$n \sim \mathcal{N}(0, \sigma) \tag{3.6}$$

such that:

$$thw'_{car} = thw_{car} + n(thw_{car}) \tag{3.7}$$

Alternatively, this can be represented directly as (using discrete integration steps of width δ):

$$\begin{aligned} P'_{lC}(d, v) &= \sum_{i=-\infty}^{+\infty} [N(d + i + \delta/2) - N(d + i - \delta/2)] \cdot P_{lC}(d + i, v) \\ &= \sum_{i=-\max_d}^{+\max_d} [N(d + i + \delta/2) - N(d + i - \delta/2)] \cdot P_{lC}(d + i, v) \end{aligned} \tag{3.8}$$

where $\max_d$ is the maximum length considered and $N(d)$ corresponds to the cumulative probability function (CDF) defined as:

$$N(d) = \int_{-\infty}^{d} n(t)dt \tag{3.9}$$

From this, a look-up table can be generated which, for different driver types, yields the probability of lane changing for a certain distance to the lead car and velocity. For the 3 driver profiles mentioned, considering $d \in \{1, ..., 80\}$ ($\max_d = 80$) and $v \in \{15, ..., 34\}$, the table generated for the vehicle in the right lane has $4,800$ rows [1]. Under the same conditions, the look-up table for the vehicle in the left obtained has 240 rows (the difference is explained by the fact that the velocity does not influence this table). Thus, the unified decision making table has $5,040$ rows.

The code presented in Appendix A.3 corresponds to the methods for obtain the unified look-up table used for the decision making and monitoring module.

[1] In the data obtained for the results presented in this dissertation, $\delta = 1$.

3.2.3 Probabilistic Control Module

In light of the uncertainty introduced in the decision making and monitoring module regarding the visual perception of distance, it becomes only natural that this would be extended to the control module as well. In order to cope with this, a probabilistic version of the control module was designed, where the noise n presented in the previous section is taken into consideration.

In order to obtain the equivalent look-up tables for the probabilisitic control module, the noise can be simulated using repeated trials, and an average can be obtained for the crashing probability, the estimated final difference in position, the estimated difference in time and estimated final velocity of the vehicle. In the case of this dissertation, the tables generated (for linear and lane changing control) were the result of 100 trials.

The code presented in Appendix A.2 corresponds to the simulation of this probabilistic version of the control, which generates the linear and steering control tables.

3.2.4 Unified Two-Module Model

Using the three tables generated (two for the probabilisitic control - linear and steering control - and one for the decision making), the final DTMC model unifies both using a sequential variable $actr\_state \in \{1, 2\}$, where $actr\_state = 1$ corresponds to the control and $actr\_state = 2$ corresponds to decision making. In the version of the unified model built for this thesis, only two vehicles are present: the first is the one controlled by the driver and the second is a lead vehicle starting at a distance $x = x_{1,0}$ (in meters) from the main vehicle (which starts at $x = 0$) and moves with a constant speed of v_1. Its movement is, therefore, completely determined by the formula:

$$x_1(t) = x_{1,0} + v_1 \cdot t \tag{3.10}$$

The distance between the two vehicles can be obtained as:

$$d(t) = |x(t) - x_1(t)| \tag{3.11}$$

And the boolean function *posDist* can be defined as:

$$posDist(t) = \begin{cases} \text{true}, & \text{if } x(t) \geq x_1(t) \\ \text{false}, & \text{otherwise.} \end{cases} \tag{3.12}$$

Due to the large size of the tables generated, the models should be generated automatically for a given tuple of initial conditions $(d_{type}, v, v_1, x_{1,0})$ (where d_{type} is the driver class according to the ones defined in the decision making and monitoring subsection - 1 is aggressive, 2 is average and 3 is cautious), following the assumptions described below:

General assumptions

1. A transition between from state s to state s' is possible if, and only if, the variable $actr\_state$ takes different values in s and s'.
2. A transition between from state s to state s' is possible if, and only if, $t_{s'} \geq t_s$ (where t_a is the value of variable t in state a).
3. The model should have no states with self transitions, as there is always a continuous evolution of the state of the vehicles.
4. A deadlock state should only be entered if, and only if, the vehicle either crashes ($crashed = true$) or reaches the end of the road ($x = length$).

Control ($actr\_state = 1$)

1. If no lane change has been decided, the model reasons over linear acceleration using the corresponding look-up table for $d(t)$ and $v(t)$, and updates the state variables t, x, v, a and $crashed$ accordingly using the discrete laws of motion previously presented.
2. If a lane change has been decided, the model reasons over the lane change using the corresponding look-up table for o_{lane}, $d(t)$, $v(t)$, v_1, and updates the variables x and t incrementally using the values Δx and ΔT of the table, as well as sets the value of v and $crashed$.

Decision Making and Monitoring ($actr\_state = 2$)

1. If the vehicle is on the left lane and behind the other vehicle ($lane = left$ and $posDist = false$), the model will not attempt a lane change.
2. If the vehicle is on the left lane and in front of the other vehicle ($lane = left$ and $posDist = true$), the model will reason over lane changes using the decision making and monitoring look-up table for d_{type}.
3. If the vehicle is on the right lane and behind the other vehicle ($lane = right$ and $posDist = false$), the model will reason over lane changes using the decision making and monitoring look-up table for d_{type}.
4. If the vehicle is on the right lane and in front of the other vehicle ($lane = left$ and $posDist = true$), the model will not attempt a lane change.

From these rules and the tables obtained in the abstraction of the individual modules, it is possible to build a model generator. The code for an example of such a generator written in *Python* (used throughout the rest of the dissertation) is presented in Appendix A.4.

By running the model generator using the conditions $(d_{type}, v, v_1, x_{1,0}) = (1, 30, 22, 51)$, that is, an aggressive driver, starting at $30m/s$ with another vehicle going at $22m/s$ and starting at $51m$, the resulting model description is the one presented in Listing 3.1 (shortened for the sake of space saving; the full model is $13,435$ lines long).

```
//Model automatically built using model_generator.py for v1 = 22 and
    driver_type = 1 (to alter these values, run the script again).
//Generated on 31-07-2018 at 18:47.

dtmc

const int length = 500; // road length
const int driver_type = 1; // 1 = aggressive, 2 = average, 3 = cautious
    drivers - do not alter this manually!
const int max_time = 35; // maximum time of experiment

// Other vehicle
const int v1 = 22; // do not alter this manually!
const int x1_0 = 51;
```

```
// Environment variables
global t : [0..max_time] init 0; // time
global crashed : bool init false;

// Vehicle controlled
global actrState : [1..2] init 1; // active module: 1 = control (both cars
   ), 2 = decision making + monitoring
global lC : bool init false; // lane changing occuring?
global x : [0..length] init 0;
global v : [15..34] init 30;
global a : [-2..3] init 0;
global lane : [1..2] init 1;

formula x1 = x1_0 + v1*t;
formula dist = x1>x?(x1 - x):(x - x1);
formula positiveDist = (x < length)?x > x1:true;

module Decision_Making_Monitoring

  // If a crash occurs, then nothing else can happen
  //[] actrState = 2 & crashed -> 1:(crashed' = true);

  // If we are in lane 2, but behind the other vehicle, don't try to pass
  [] actrState = 2 & !crashed & lane = 2 & positiveDist = false -> 1:(
     actrState' = 1);

  // If we are in lane 1, and no vehicle is in front, don't change lanes
  [] actrState = 2 & !crashed & lane = 1 & positiveDist = true -> 1:(
     actrState' = 1);

  [] actrState = 2 & !crashed & lane = 1 & positiveDist = false & dist = 1
      & v = 15 -> 0.8:(actrState' = 1) & (lC' = true) + 0.2:(actrState' =
      1) & (lC' = false);

  ...

  [] actrState = 2 & !crashed & lane = 2 & positiveDist = true & dist >=
     80 -> 1:(actrState' = 1) & (lC' = true) + 0:(actrState' = 1) & (lC'
     = false);
endmodule

module Control

  // If we are in lane 1, and no lane change was decided, continue forward
      (which might result in crash)
  // The vehicle is behind the other driver (positiveDist = false, x < x1)
  [] actrState = 1 & !crashed & !lC & lane = 1 & x <= length - v & t < max
```

```
    _time & positiveDist = false & (x1 + v1 - x - v) >= 6 & v + a < 34 &
     v + a > 15 & dist = 1 & v = 15 -> 1:(x' = x + v) & (t' = t + 1) & (
    v' = v + a) & (a' = -2) & (actrState' = 2);

 ...

 [] actrState = 1 & !crashed & lC & lane = 2 & dist >= 43 & v = 34 & x >
    length - 136 & t > max_time - 6 -> 1:(crashed' = false) & (x' =
    length) & (v' = 34) & (t' = max_time) & (a' = 0) & (lane' = 1) & (
    actrState' = 2) & (lC' = false);

endmodule
```

Listing 3.1 Example of the model generated for the tuple $(d_{type}, v, v_1, x_{1,0}) = (1, 30, 22, 51)$ (shortened)

By loading and building the model in either PRISM or Storm, it is observable that it has 257 states and 295 transitions, a significant reduction from the possibilities presented in the straightforward control abstraction. Due to the fact that there are $3 \times 20 \times 20 \times length$ different possibilities in terms of initial conditions, it would be intractable, in the timeline of the dissertation, to obtain the number of states for all the combinations. Table 3.1 presents the number of states and transitions for some arbitrarily generated conditions (for a road of length $500m$).

d_{type}	v	v_1	$x_{1,0}$	**# States**	**# Transitions**
3	21	30	20	305	333
1	27	22	66	396	471
1	28	17	43	167	183
2	33	15	35	6	7
3	28	21	38	199	222
1	19	16	81	391	443
2	25	23	28	390	481
3	15	17	36	487	569
1	29	18	74	201	220
2	31	29	52	234	265

Table 3.1 Results of the number of states and transitions for arbitrary initial conditions.

3.3 Model Evaluation Metrics

With the model efficiently represented as a DTMC through abstraction, it becomes possible to perform model checking of certain properties on it in order to evaluate the relative performance of different profiles of drivers. In the following subsections, different categories of evaluation are presented and properties representing metrics within that category are proposed and tested in a small population of scenarios (the same test cases used to generate Table 3.1). In the rest of the dissertation it is assumed (unless explicitly mentioned otherwise) that the test road segment has a length of $500m$.

3.3.1 Completeness Properties

From the rules used to build the model, it is explicit that a correct model should only enter a deadlock if it crashes or the vehicle reaches the end of road. As such, it is expected that every path from the initial state leads to states where the vehicle is in one of those two conditions. This property can be explicitly written as:

```
P>=1 [F (crashed | x=length)]
```

and it tests whether a model is complete by verifying that every possible outcome satisfies the restrictions imposed (regardless of the value of the other variables). Additionally, both PRISM and Storm allow the following property to be verified:

```
P>=1 [F "deadlock"]
```

which guarantees that a deadlock will eventually be reached and no infinite loop is generated by mistake within the DTMC.

Using PRISM or Storm, the results of the model checking of these properties in the test cases are presented in Table 3.2. As expected, for all the scenarios the properties are satisfied, confirming that these models are complete in terms of the expected outcome. While these properties does not need to be extensively analysed, they constitute a guarantee that the generated model is complete and its verification is performed in all tests performed in the analysis of Chapter 5.

d_{type}	v	v_1	$x_{1,0}$	`P>=1 [F (crashed | x=length)]`	`P>=1 [F "deadlock"]`
3	21	30	20	`true`	`true`
1	27	22	66	`true`	`true`
1	28	17	43	`true`	`true`
2	33	15	35	`true`	`true`
3	28	21	38	`true`	`true`
1	19	16	81	`true`	`true`
2	25	23	28	`true`	`true`
3	15	17	36	`true`	`true`
1	29	18	74	`true`	`true`
2	31	29	52	`true`	`true`

Table 3.2 Results of the verification of the completeness properties.

3.3.2 Safety Property

One essential evaluation metric of the human driver is its capability of driving safely, i.e. without crashing. To that effect, the following safety property is devised:

```
P=? [F crashed]
```

It should be noted that, for a given set of initial conditions, the lower the quantitative value of the property, the safer the driver is. From the assumptions made in the abstraction process, it is expected that Aggressive drivers will perform worse than Average ones in this metric, which in turn will perform worse than Cautious drivers. Extensive experimental results regarding this property are presented in Chapter 5. The results of the verification of the safety property in the test cases are presented in Table 3.3. It can observed that these vary quite significantly according to the situation, with the quantitative values ranging from 0.0193 (unlikely to crash) to 1 (will crash with certainty).

d_{type}	v	v_1	$x_{1,0}$	P=? [F crashed]
3	21	30	20	0.0232
1	27	22	66	0.3017
1	28	17	43	0.7119
2	33	15	35	1
3	28	21	38	0.1604
1	19	16	81	0.6074
2	25	23	28	0.0562
3	15	17	36	0.0276
1	29	18	74	0.5123
2	31	29	52	0.0193

Table 3.3 Results of the verification of the safety property.

3.3.3 Liveness Properties

Liveness can be defined in terms of the efficiency of the drivers, i.e. how quickly do they reach the end of the road. One of the variables of the model is the time, t, in seconds since the beginning of the execution. For a given constant T, the property:

```
P=? [F (x=length & t < T)]
```

captures the probability of reaching the end within less than T seconds. It should be noted that this constant T needs to be adjusted to different values in order to test specific properties, so this property should actually be defined as the list of properties:

```
P=? [F (x=length & t < 1)]
P=? [F (x=length & t < 2)]
...
P=? [F (x=length & t < max_time-1)]
P=? [F (x=length & t < max_time)]
```

However, the problem with these properties relies on the fact that they intrinsically rely on the safety property, since a higher probability of crashing naturally implies a lower probability of reaching the end, and therefore an even lower probability of reaching the end under T seconds. To mitigate this issue, the following conditional

properties are introduced:

```
P=? [F (x=length & t<T) || F (x=length)]
```

which reads as *"what is the probability that the model eventually reaches a state where* `x = length` *and* `t < T` *given that it reaches one where* `x = length`*"*. This allows for direct comparisons between driver profiles and different scenarios. It should be noted that these are actually *max_time* different properties, as with the case of the unconditional ones.

Using Storm, both the unconditional and conditional properties can be verified directly. While PRISM supports the unconditional ones, conditional properties are not yet supported by the tool. However, it is possible to verify separately the properties `P=? [F (x=length & t<T)]` and `P=? [F (x=length)]`, and use Bayes' theorem:

$$\texttt{P=? [F (x=length \& t<T) || F (x=length)]} = \frac{\texttt{P=? [F (x=length \& t<T)]}}{\texttt{P=? [F (x=length)]}} \tag{3.13}$$

Table 3.4 presents the results of the verification of the liveness properties in the test cases. In the interest of brevity, only two of each (unconditional and conditional) properties verified are presented, for $T = 19$ and $T = 24$ (arbitrarily selected), and they will be identified using the following number system (for presentation purposes):

```
1. P=? [F (x=length & t<19)]
2. P=? [F (x=length & t<24)]
3. P=? [F (x=length & t<19) || F (x=length)]
4. P=? [F (x=length & t<24) || F (x=length)]
```

From Table 3.4, it is possible to observe that different scenarios, despite having distinct values of the unconditional properties, have a similar value of the conditional ones. For example, for $T < 24$, the scenario $(d_{type}, v, v_1, x_{1,0}) = (1, 28, 17, 43)$ has a a significantly lower unconditional probability than the $(d_{type}, v, v_1, x_{1,0}) = (3, 28, 21, 38)$, yet they have exactly the same conditional one. While the unconditional properties can be interpreted in these scenarios as *"the vehicle will reach the end of the road and be under 24s with probability 0.2881"* and *"the vehicle will reach the end of the road and be under 24s with probability 0.8396"*, in both these cases

d_{type}	v	v_1	$x_{1,0}$	1	2	3	4
3	21	30	20	0.0085	1	0.0085	1
1	27	22	66	0.5908	0.6983	0.8460	0.9999
1	28	17	43	0	0.2881	0	1
2	33	15	35	0	0	0	0
3	28	21	38	0	0.8396	0	1
1	19	16	81	0	0.3926	0	0.9999
2	25	23	28	0.1555	0.9438	0.1648	0.9999
3	15	17	36	0	0.5849	0	0.6015
1	29	18	74	0.4662	0.4877	0.9559	1
2	31	29	52	0.9992	0.9992	1	1

Table 3.4 Results of the verification of the liveness properties.

the conditional probability should be interpreted as *"if the vehicle reaches the end of the road, then it will do so before 24s with certainty"*. The conditional property can be understood as an elimination of the safety bias of inherent to each scenario, leading to more meaningful intersituational comparisons. Extensive experimental results regarding these property are presented in Chapter 5

3.4 Simulation of Paths in the Model

In order to visualise possible executions of the trajectory and decisions taken in the model, a visual simulator was designed and implemented in *Python*, with the code presented in Appendix A.7.

The simulator essentially uses the option `simpath` in the PRISM command line tool which allows it to output a simulated path in the model without having to actually build it. Using this outputted path in the model, the script reads it and plays it back to the user using a GUI built in *pygame*.

In order to faithfully represent the lane change operations the driver performs, an additional table is obtained using the steering control abstraction, which corresponds to the interpolation of the x and y positions of the vehicle as a function of time (i.e.

$x(t)$ and $y(t)$). Given the complexity of the y movement compared to the x movement, the positions are represented by a 6^{th} and 2^{nd} degree polynomial, respectively.

Due to the size of the example, only one test case is presented of a simulation of a path in a model. In Figure 3.7, an example of several frames of the simulation of a path in the scenario $(d_{type}, v, v_1, x_{1,0}) = (3, 28, 21, 38)$ is presented. In this case, the driver initiates a lane change to perform an overtake at the 2s mark and returns to the original lane at the 8s mark, with no crashing occurring. In the simulation, the vehicle took $18.3s$ to reach the end of the road. It should be noted that, as verified by the conditional liveness property, the vehicle took less than $24s$ to reach the end of the road segment. While it would appear the vehicle took less than $19s$ to reach the end (and thus would violate the probability of 0 obtained for this liveness property), the model considers steps of $1s$, so in the model, the vehicle actually took $19s$ to finish instead of $18.3s$.

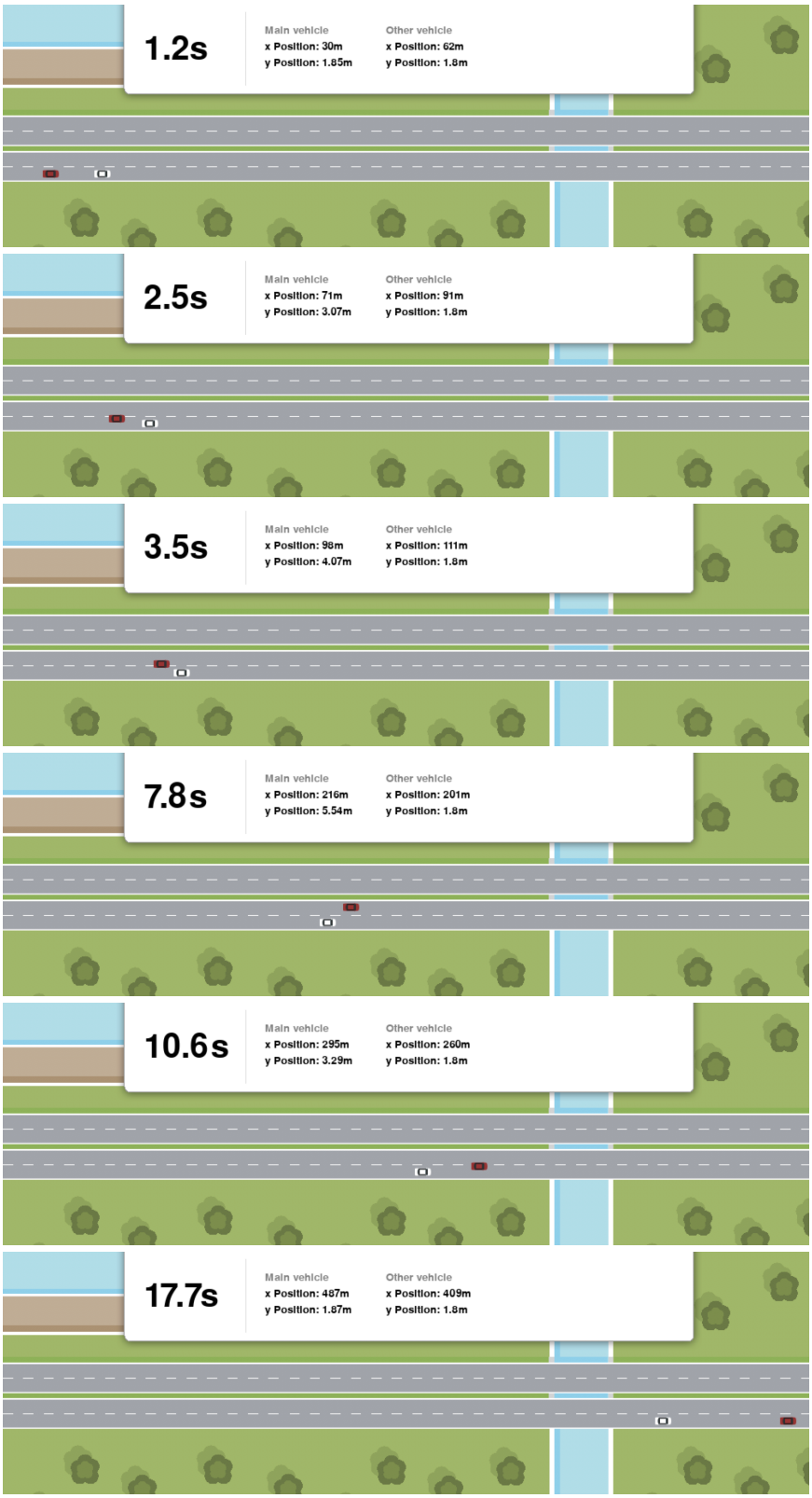


Figure 3.7 Snapshots of the simulator for one of the paths in the scenario $(d_{type}, v, v_1, x_{1,0}) = (3, 28, 21, 38)$.

Chapter 4

Advanced Driver Assistance Systems

This chapter explores and compares several ideas for assistance systems which can be implemented on top of the human driver model generated in the previous one. It starts by tackling the design of the system in terms of the placement in the existing module based model and the capabilities within it, following an incremental approach (considering the limitations of the vehicle and drivers). It then performs a comparison of the different approaches using two distinct test cases and it concludes which ADAS performs the best (within reasonable assumptions). This solution is then evaluated and compared to the human driver in Chapter 5.

4.1 Driver Assistance System Design

One of the aims of this dissertation is to obtain correct-by-construction driver assistance systems. As such, the design of the assistance system in this context corresponds mostly to determining which actions are available to the system at each state (the model becomes an MDP), and then performing synthesis using adequate properties. It should be noted that these actions must be realistic in nature, otherwise the obtained assistance system would prove to be useless in a real-world scenario - and usefulness is the end goal in terms of deployment. Figure 4.1 presents the underlying assumptions of where the system would lie in the environment, which influences the possibilities in terms of the design.

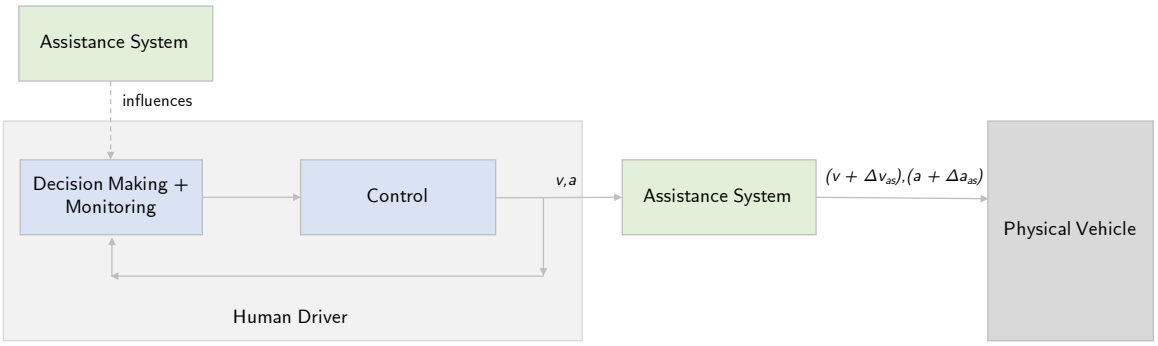


Figure 4.1 Overview of the system with the possibilities of the ADAS intervention.

It is assumed that the assistance system can not change the decision making (as it is a human cognitive process), but it can influence it to a certain degree through suggestion. This would be the ideal way of implementing the ADAS, as it would not require any intervention in the physical systems of the vehicle. Therefore, the first possibilities explored are based on this option. However, in the interest of safety and efficiency, incremental control based options are also developed, both at the level of linear acceleration (influencing a through Δa_{as}), as well as at the level of steering control (by influencing v and a through Δv_{as} and Δa_{as}, respectively).

4.1.1 Decision Making based ADAS

4.1.1.1 Decision Making with Fully Compliant Drivers

At the decision making level, the human driver model considered has two options: it either changes lanes, or it does not. These options can be influenced using suggestions (e.g. through visual or auditive cues) which can lead to either one or the other being taken. Salvucci in [22] does not consider accelerating and decelerating as part of the decision making because he argues they happen instinctively and therefore they should be exclusively part of the control. However, if a driver can be influenced to make a conscious decision to decelerate, for example, through the driver assistance system, then there is an argument for including this action in the decision making. Thus, a 3-option ADAS was designed, where an action α is defined as:

$$\alpha \in A := \{(lc) \mid (\neg lc \wedge a = a_p) \mid (\neg lc \wedge a = a_d)\} \tag{4.1}$$

where lc corresponds to lane changing, a is the acceleration, a_p is the acceleration the vehicle is currently holding and a_d is a constant value for deceleration that a driver applies when suggested to decelerate. In this dissertation, it was assumed that $a_d = -1$ (minimum decelerating value).

Considering that the drivers are fully compliant with the suggestions given by the system, the decision making at each step can be replaced by the all the possible actions, obtaining an MDP with three choices at this level. The code presented in Appendix B.1.1 corresponds to the implementation of a generator for this ADAS.

4.1.1.2 Decision Making with Partially Compliant Drivers

In the case of the ADAS previously designed, it is assumed that humans will not only follow all the suggestions, but they will comply with them fully and immediately after they are received (within the same ACT-R cycle in the decision making). However, this is not a realistic assumption by any means. In fact, people are more prone to following suggestions when these align with their original interests, than otherwise. With this in mind, a new solution is presented which more accurately represents the suggestive decision making considered in this section.

In the human driver model, a single action was available for a set of conditions, such that at any decision making state ($actr\_state = 2$) s there would be a value p such that the next state s' could be written as:

$$s' = p : (lC) + (1 - p) : (\neg lC \,\&\, a = a_p) \tag{4.2}$$

Consider a factor $\gamma \in [0, 1]$ which corresponds to how responsive a driver is to the suggestions given. The following rule can be written for each of the actions $\alpha_i \in A$ of the MDP according to the states s'_i which they lead to:

$$s'_i = \gamma : \alpha_i + (1 - \gamma) \cdot p : (lC) + (1 - \gamma) \cdot (1 - p) : (\neg lC \,\&\, a = a_p) \tag{4.3}$$

Given that $\gamma, p \in [0, 1]$, the transitions are guaranteed to sum up to one for every case. The decision making with fully compliant drivers corresponds to case where

$\gamma = 1$. The code presented in Appendix B.1.2 corresponds to the implementation of a generator for this particular ADAS.

4.1.2 Control based ADAS

While the decision making under the assumption of fully compliant drivers appears to be a powerful option in terms of safety and liveness, the weakening of the assumption to partially compliant drivers is expected to reduce performance substantially, particularly for lower values of γ. As such, control based assistance is added on top of the decision making assistance considered for partially compliant drivers, so as to improve performance. It should be noted that this type of assistance does not require human intervention, as per the assumptions noted in Figure 4.1.

4.1.2.1 Active Linear Acceleration Control

Considering the assumptions previously described, active linear acceleration control consists in an incremental addition to the acceleration value proposed by the human in the control module. Assume the acceleration of the vehicle imposed by the human is given in the model by $a \in \{a^{\min}, ..., a^{\max}\}$. In this module, a value Δa_{as} is considered such that:

$$\Delta a_{as} \in \{\Delta a_{as}^{\min}, ..., \Delta a_{as}^{\min}\} : \Delta a_{as}^{\min} > a^{\min} \wedge \Delta a_{as}^{\max} < a^{\max} \tag{4.4}$$

and the final acceleration applied to the vehicle becomes (considering as well that $a' \in \{a^{\min}, ..., a^{\max}\}$):

$$a'(t) = \max(\min(a(t) + \Delta a_{as}, a^{\max}), a^{\min}) \tag{4.5}$$

The restriction to the values of Δa_{as} presented in Equation 4.4 allows the system to be incremental (i.e. corrective) instead of enforcing the specific values chosen by the control assistance system. This is important to avoid strategies for the control assistance system which sharply contrast in terms of the values chosen, e.g. a strategy which at a certain time chooses an acceleration of 3 and in the next time step chooses one of -2 (not allowed for a small enough range of Δa_{as} and according to the linear

control abstraction obtained in Chapter 3). In this dissertation, it is assumed that $\Delta a_{as} \in \{-1, 0, 1\}$.

The implementation of such system consists in replacing the existing linear control at each step of the control module by the resulting accelerations of applying all the possible values of Δa_{as} to the acceleration decided by the human control module, obtaining an MDP with at most three actions at the linear acceleration control level (and at least two). The code presented in Appendix B.1.3 corresponds to the implementation of a generator for this ADAS (with the decision making assistance for partially compliant drivers).

4.1.2.2 Active Steering Control

While linear acceleration control assistance improves safety and liveness, steering assistance can also be improved using incremental velocity and acceleration.

In [22], Salvucci introduces a control law for the steering angle φ, as presented in Section 2.2, for a given k_{far}, k_{near} and k_I. By changing the values of these constants, different control laws are obtained, which introduce different accelerations and velocities at each time step, mimicking the behaviour of the incremental control previously assumed (i.e. within such a strategy, the difference in acceleration and velocity introduced is the difference between these values for the two control laws at each time step). The value of $\theta_{\max}$ (as defined in Section 2.2), guarantees the feasibility of the movement in terms of the acceleration induced. Thus, actions in these assistance system at the model level correspond to different sets of (k_{far}, k_{near}, k_I) available to the ADAS.

In this dissertation, three distinct sets of parameters were considered as possible actions, $(k_{far}, k_{near}, k_I) \in \{(15, 3, 5), (17, 3, 6), (14.5, 3, 7)\}$. An example of the simulation for the situation where $o_{lane} = 1$, $d = 20m$, $v = 15m/s$ and $v_1 = 15m/s$ (with the non-probabilistic control) is presented in Figure 4.2.

Using these values, new look-up tables were obtained with the probability of crashing, the Δx and ΔT incurred and the final velocity of the vehicle in question (following the probabilistic control module presented in Section 3.2.3) for each origin lane, distance to the other vehicle, velocities of both the vehicle in question and the other vehicle and the parameters of the control law (out of the three possibilities presented). Following

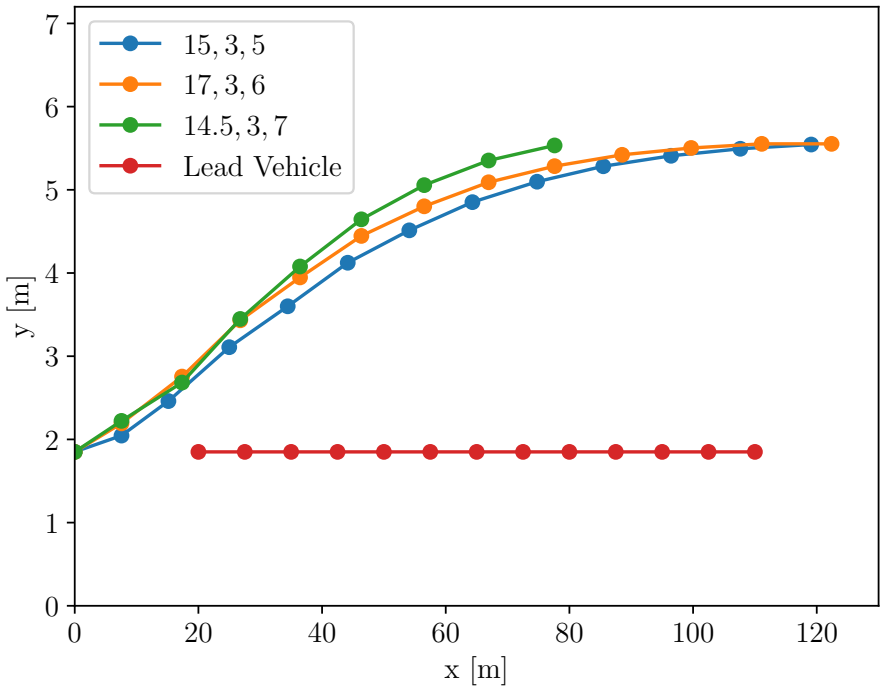


Figure 4.2 Example of the simulation of the lane change for $o_{lane} = 1$ (right lane), $d = 20m$, $v = 15m/s$ and $v_1 = 15m/s$ (the legend of each path corresponds to the situation with parameters k_{far}, k_{near}, k_I, respectively).

the same calculations as shown in Section 3.2.1, the obtained look-up table contains $103,200$ rows.

In terms of implementation, the MDP is obtained through simply reading the three possible actions for lane changing directly from the look-up table, similarly to the human driver model (the difference being the latter only has one option). The code presented in Appendix B.1.4 corresponds to the implementation of a generator for this ADAS (with the decision making assistance for partially compliant drivers and active linear acceleration control).

4.2 Design Evaluation

While it would appear to be the case that an ADAS with both decision making and control assistance at the linear acceleration and steering level would be the optimal choice for the driver assistance system, there are no guarantees that this is true. As such, an evaluation and comparison of all the possible designs must be performed in order to determine the best option which can then be compared to the human driver in Chapter 5. To do so, it is necessary to establish multi-objective metrics and some meaningful test cases (as testing all the options would be infeasible).

4.2.1 Multi-Objective Metrics

As established in Section 3.3, safety and liveness are two important metrics which will be used in Chapter 5 to evaluate the human driver model. Therefore, it makes sense that such properties should be the focus of the synthesis for the driver assistance system. The goal is to minimise the probability of crashing and to maximise the liveness property. This can be obtained using a multi-objective query such as the one below:

```
multi(Pmin=? [F crashed], Pmax=? [F (x=length) & (t<T)])
```

for a given constant T. Due to the trade-offs between both properties, the model checking of the multi-objective property of the format of the above generates a Pareto curve, where each point on the curve corresponds to optimal achievable values in terms of `P=? [F crashed]` and `P=? [F (x=500) & (t<T)]` under a given strategy. As such, it is possible to compare strategies through the comparison of the Pareto curves obtained for the different ADAS for some test cases.

However, the problem of the safety bias discussed in Section 3.3.3 persists through this comparison, and the ideal metric would take into account the conditional probability instead of the unconditional one. Since PRISM does not allow for the model checking of conditional properties, the Pareto curve using the conditional properties needs to be obtained in a different way. At this point, it is important to notice the following equality for a given strategy σ in the MDP:

$$P_s^\sigma(\texttt{F crashed}) = 1 - P_s^\sigma(\texttt{F x=length}) \tag{4.6}$$

as long as $p_{\max,s}(\texttt{F (crashed | x = length)}) = p_{\min,s}(\texttt{F (crashed | x = length)}) = 1$ (completeness property for the MDP). This is true from the fact that the events crashing and reaching the end are mutually exclusive. Thus, it is possible to write:

$$\begin{aligned} P_s^\sigma(\texttt{F (x=length) \& (t<T) | F x=length}) &= \frac{P_s^\sigma(\texttt{F (x=length) \& (t<T)})}{P_s^\sigma(\texttt{F x=length})} \\ &= \frac{P_s^\sigma(\texttt{F (x=length) \& (t<T)})}{1 - P_s^\sigma(\texttt{F crashed})} \end{aligned} \tag{4.7}$$

It is therefore possible to obtain the Pareto curve with the conditional properties by using Equation 4.7 on each of the points in the unconditional curve (generating the achievable set through this).

4.2.2 Test Cases

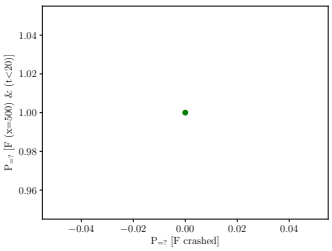
Due to the infeasibility of generating all the possible combinations of test cases, two test cases were considered when comparing the four different ADAS designed in the previous section. For each of these test cases, the unconditional and conditional Pareto curves for each driver type are presented for a T representative of the scenario and which allows for comparison between the ADAS considered. In terms of parameters of the model, the same values were used as in the human driver model part of the implementation, and γ was set to 0.1.

For the sake of brevity, the following numbering will be assumed in the captions and discussion of the Pareto curves presented in the test cases:

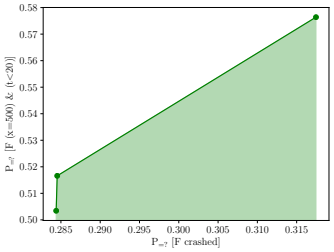
1. Decision making based ADAS with fully compliant drivers
2. Decision making based ADAS with partially compliant drivers (varies with classes of drivers)
3. Decision making with partially compliant drivers + linear acceleration control ADAS (varies with classes of drivers)

4. Decision making with partially compliant drivers + linear acceleration control + steering control ADAS (varies with classes of drivers)

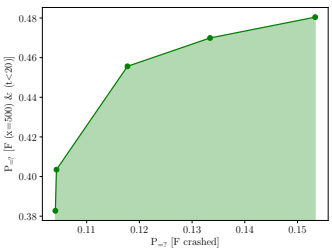
4.2.2.1 $v = 21$ **m/s,** $v_1 = 19$ **m/s,** $x_{1,0} = 70$ **m**



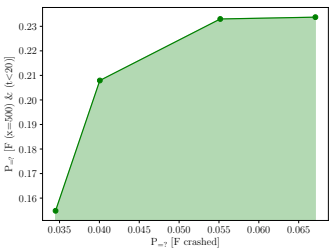
a. 1



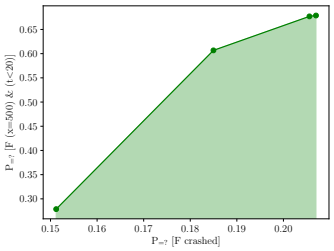
b. 2 (Aggressive drivers)



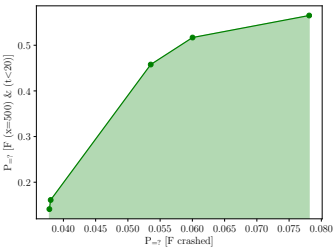
c. 2 (Average drivers)



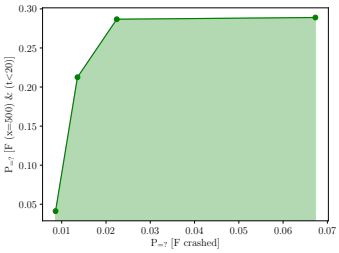
d. 2 (Cautious drivers)



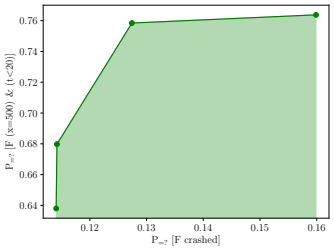
e. 3 (Aggressive drivers)



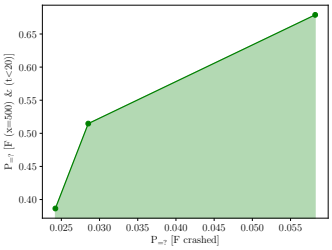
f. 3 (Average drivers)



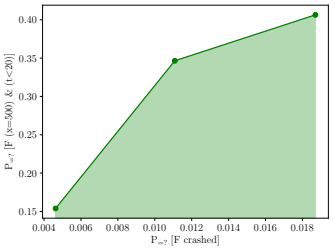
g. 3 (Cautious drivers)



h. 4 (Aggressive drivers)

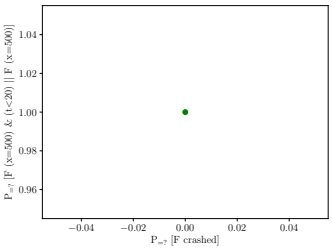


i. 4 (Average drivers)

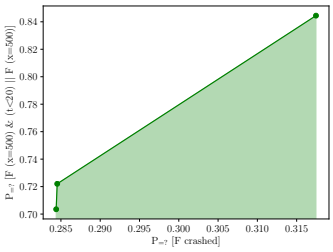


j. 4 (Cautious drivers)

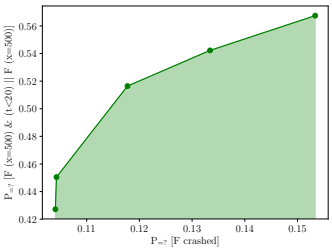
Figure 4.3 Pareto curves for the unconditional properties, for $T = 20$.



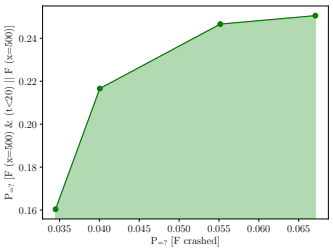
a. 1



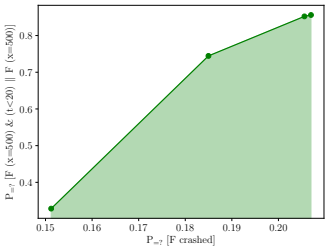
b. 2 (Aggressive drivers)



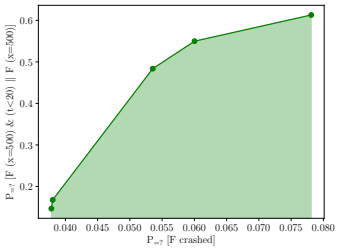
c. 2 (Average drivers)



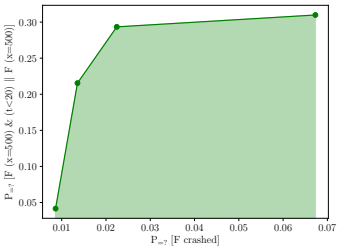
d. 2 (Cautious drivers)



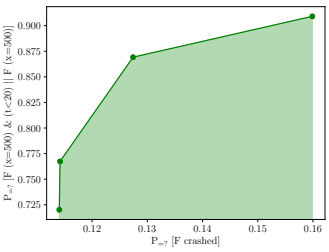
e. 3 (Aggressive drivers)



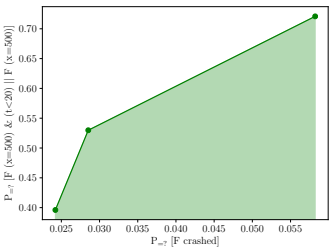
f. 3 (Average drivers)



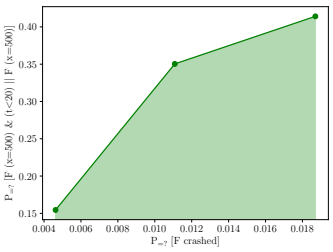
g. 3 (Cautious drivers)



h. 4 (Aggressive drivers)



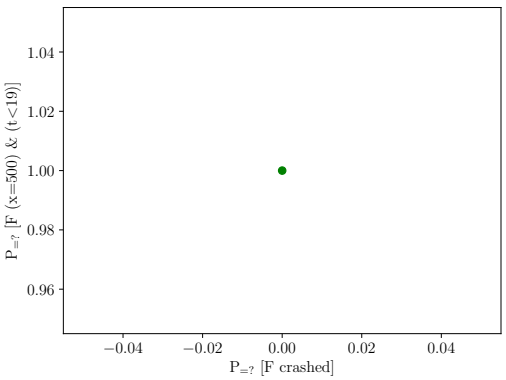
i. 4 (Average drivers)



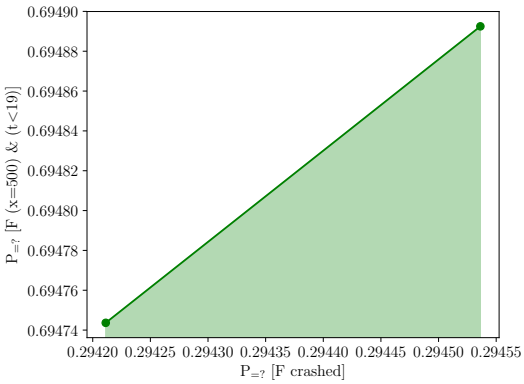
j. 4 (Cautious drivers)

Figure 4.4 Pareto curves for the conditional properties, for $T = 20$.

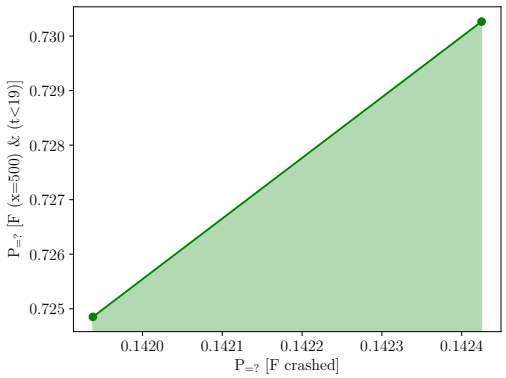
4.2.2.2 $v = 30$ **m/s,** $v_1 = 22$ **m/s,** $x_{1,0} = 50$ **m**



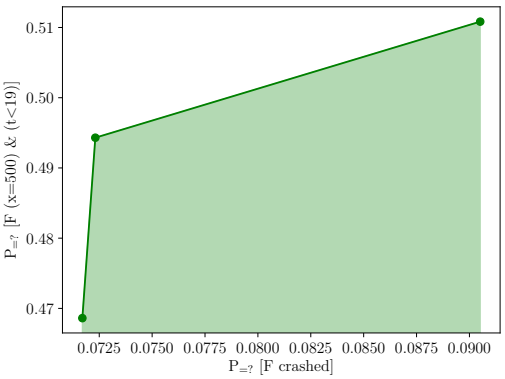
a. 1



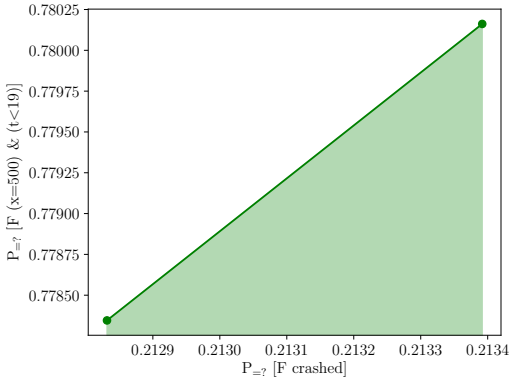
b. 2 (Aggressive drivers)



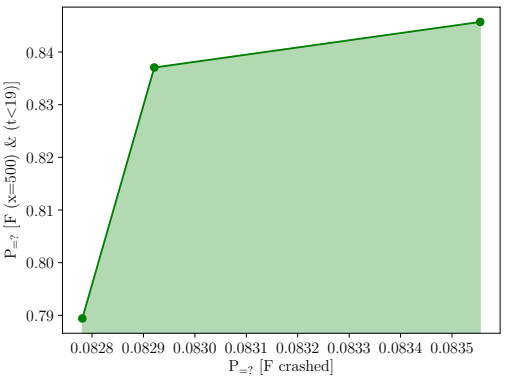
c. 2 (Average drivers)



d. 2 (Cautious drivers)

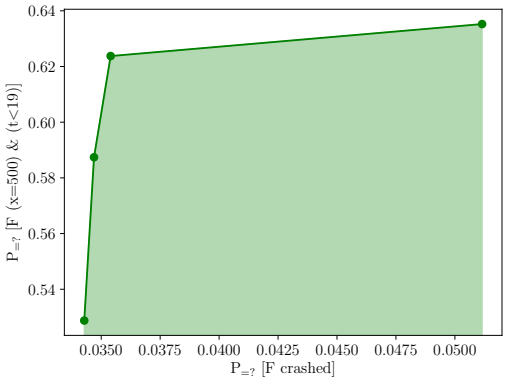


e. 3 (Aggressive drivers)

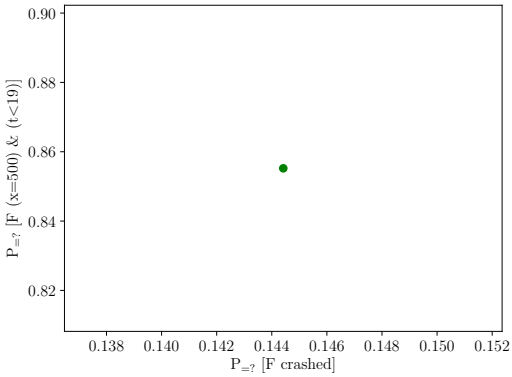


f. 3 (Average drivers)

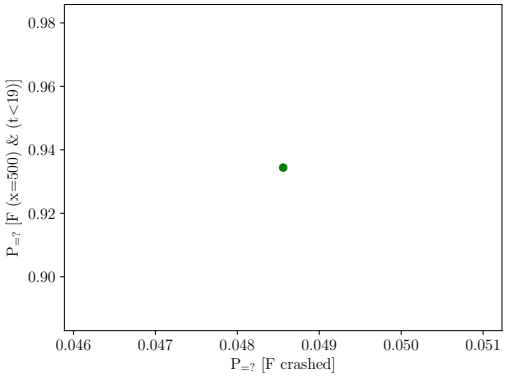
Figure 4.5 Pareto curves for the unconditional properties, for $T = 19$.



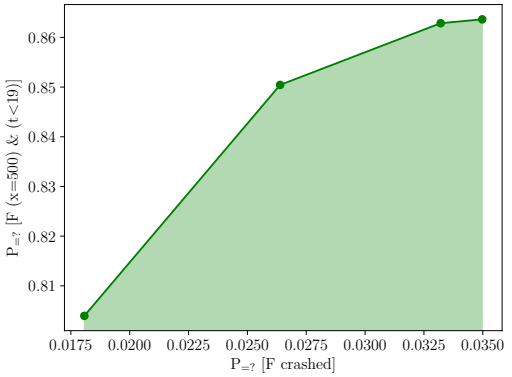
g. 3 (Cautious drivers)



h. 4 (Aggressive drivers)

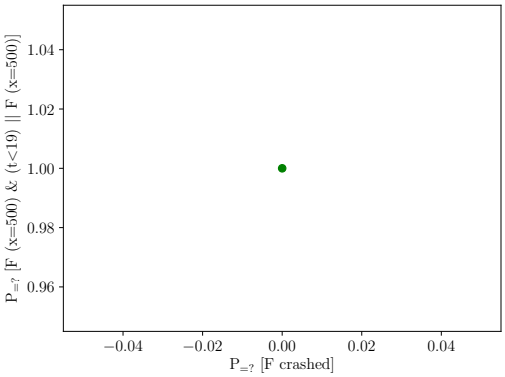


i. 4 (Average drivers)

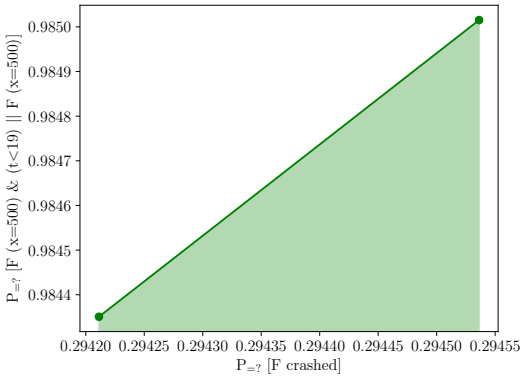


j. 4 (Cautious drivers)

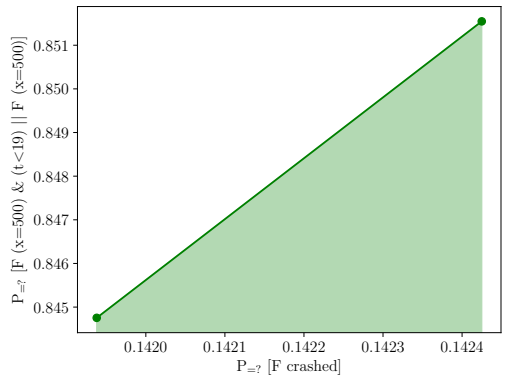
Figure 4.5 Pareto curves for the unconditional properties, for $T = 19$ (cont.).



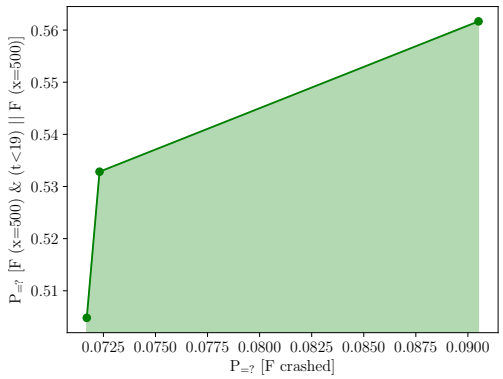
a. 1



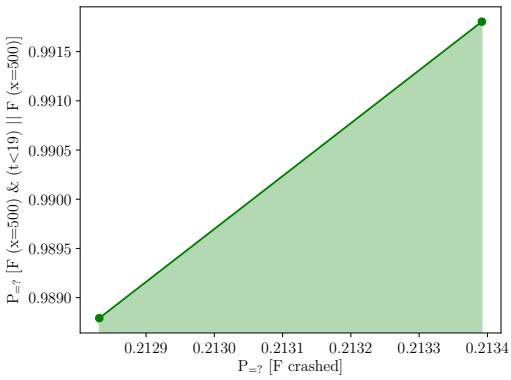
b. 2 (Aggressive drivers)



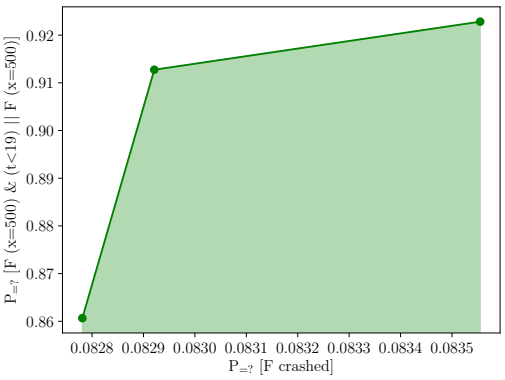
c. 2 (Average drivers)



d. 2 (Cautious drivers)

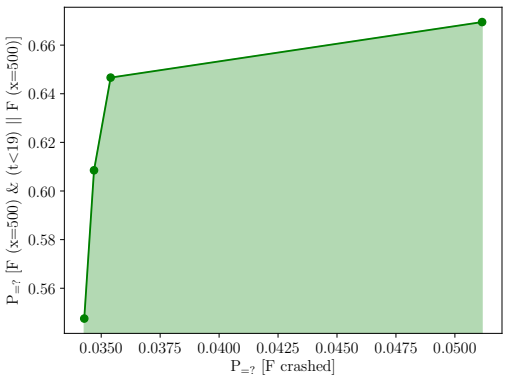


e. 3 (Aggressive drivers)

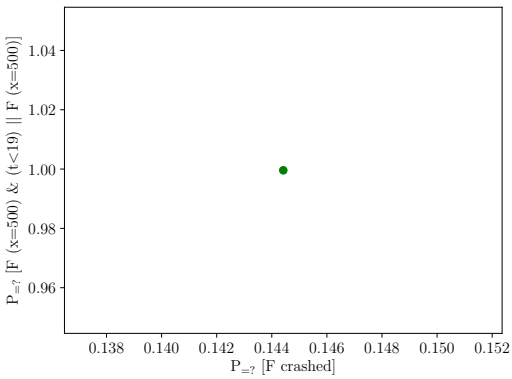


f. 3 (Average drivers)

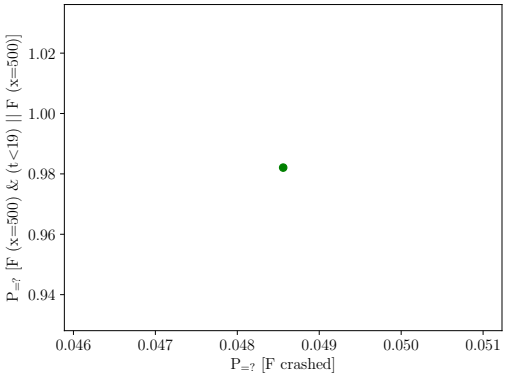
Figure 4.6 Pareto curves for the conditional properties, for $T = 19$.



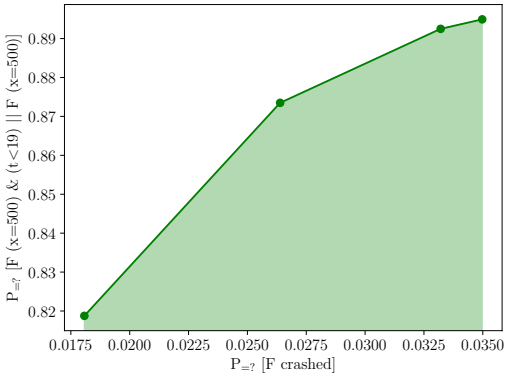
g. 3 (Cautious drivers)



h. 4 (Aggressive drivers)



i. 4 (Average drivers)



j. 4 (Cautious drivers)

Figure 4.6 Pareto curves for the conditional properties, for $T = 19$ (cont.).

4.2.3 Overall Comparison

From the Pareto curves presented, it is noticeable that the best performing ADAS is the first presented, that is, a decision making assistance system with the assumption of fully compliant drivers. This assistance system allows maximum safety (`P=? [F crashed]` $= 0$) and efficiency (`P=? [F (x=500) & (t<T)]` $= 1$) for both test cases and respective values of T considered. However, as discussed before, this situation is unrealistic in nature. By adding the partial compliance assumption, the performance drops drastically, leading to fairly high probabilities of crashing in aggressive drivers (minimum of 0.285 for the first test case and 0.294 for the second one) and lower liveness performance as well. By introducing linear acceleration control, safety increases significantly (the probability of crashing is lowered), but the increase of the probabil-

ity of the liveness property being satisfied is almost negligible in some situations (and it decreases in some cases; e.g. the aggressive drivers in the first test case). In all the cases tested, the introduction of steering control improves, in both cases and for all driver classes, both safety and liveness. Therefore, the assistance system number 4, that is, decision making with partially compliant drivers, active linear acceleration control and active steering control is the best performing system in the two test cases presented (other than number 1, which is based on unrealistic assumptions).

This analysis does not provide a statistical guarantee that this is the best performing ADAS out of the ones tested in the majority of the situations (as this would be too time consuming for the time frame of this dissertation). However, the systems considered are incremental in nature, in the sense that they were built through iteration and by adding more actions to the one immediately before. As such, number 4 was expected for perform better than numbers 2 and 3, simply due to the fact that there were more choices available to it. The two test cases presented corroborate this idea. Therefore, we conclude that the assistance system number 4 would be the most complete and better candidate for deployment, and, as such, the in-depth experimental results and discussion presented in Chapter 5 use this ADAS.

Chapter 5

Experimental Results

In this chapter, the performance of the driver assistance system is evaluated by comparing the probability of certain properties in the driver model alone to those same properties in the model of driver and the ADAS designed in Chapter 4. This is achieved using the various metrics defined throughout the dissertation. Some explicit test cases are compared both in terms of liveness and safety, and a randomly generated sample population of the initial conditions is used to compare the two to further extend. After a discussion of the main results, some additional experiments are presented which are outside the scope of the comparison.

5.1 Human driver and ADAS comparison

5.1.1 Safety Results

As presented in Chapter 3, the safety property for the human driver model can be written as:

```
P=? [F crashed]
```

A similar safety property is presented in Chapter 4, adapted for the fact that the model of the human driver and driver assistance system is an MDP instead of a DTMC:

```
Pmin=? [F crashed]
```

These properties are directly comparable, as they represent the same quantitative value (this is not the case with the liveness properties). It should be noted that the lower the value of this property, the safer the system is. This means that these properties can be used not only for comparisons between the human driver alone and the full system (human and driver assistance system) for a given scenario, but also for inter-situational comparisons (i.e. with different initial conditions).

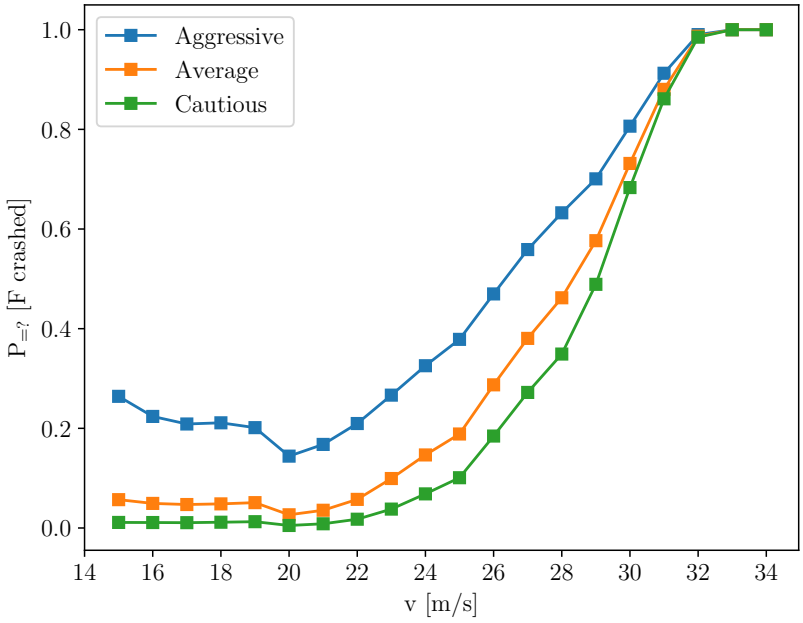
A scenario, in both cases, can be uniquely defined as a tuple $(d_t, v, v_1, x_{1,0})$, where $d_t \in \{1, 2, 3\}$ is the driver class considered (aggressive, average or cautious, respectively). In practice, this means that, for a road segment of $500m$ and considering $v, v_1 \in \{15, ..., 34\}$ and $x_{1,0} \in \{1, ..., 150\}$ (assuming that the driver will be safe for a distance greater than $150m$), there are $20 \times 20 \times 150 \times 3 = 180,000$ different scenarios to consider. Assuming that model checking each of those scenarios for both the human driver and the full system takes 5 minutes each (an assumption which will be challenged in Chaper 6), this task would take approximately 625 days to complete. Given the time frame of this dissertation, this would be infeasible. As such, three other methods where designed as a way to compare both systems:

- **Main vehicle initial velocity test cases**: for two randomly selected tuples $(v_1, x_{1,0})$, the variation of the value of the safety properties for each driver profile with the variation of v were obtained. This directly compares 20 different scenarios for each driver class. The two tuples were randomly generated as $(v_1, x_{1,0}) = (20, 35)$ and $(v_1, x_{1,0}) = (22, 40)$, and the results are presented in Figures 5.1 and 5.2, respectively.

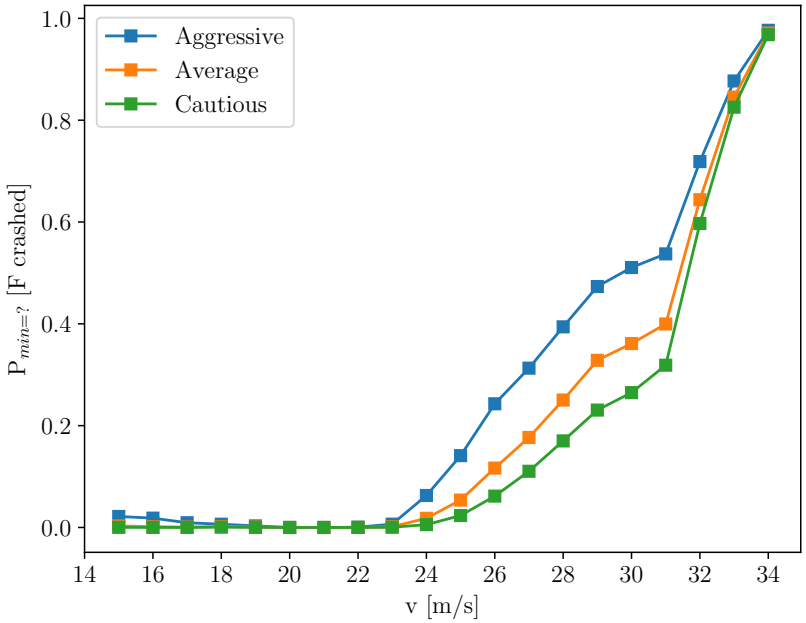
- **Bivariant initial velocity test cases**: for a randomly selected $x_{1,0}$, the variation of the values of the safety properties for each driver profile with the change of v and v_1 (within certain ranges; $v \in \{20, ..., 30\}$ and $v_1 \in \{15, ..., 25\}$) were generated. This directly compares 120 different scenarios for each driver class. The value $x_{1,0} = 50$ was randomly generated for this test case, and the results are presented in Figure 5.3.

- **Box plots**: for a randomly generated sample of 100 different initial conditions of $(v, v_1, x_{1,0})$, obtain a box plot which represents the distribution of the values

of the safety properties for this sample and for each driver profile. The results are presented in Figure 5.4.

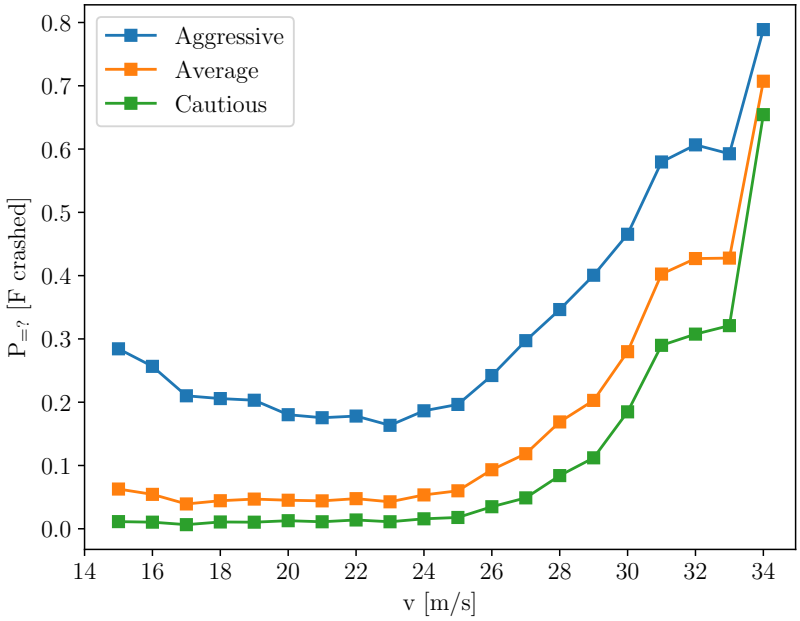


a. Human driver model

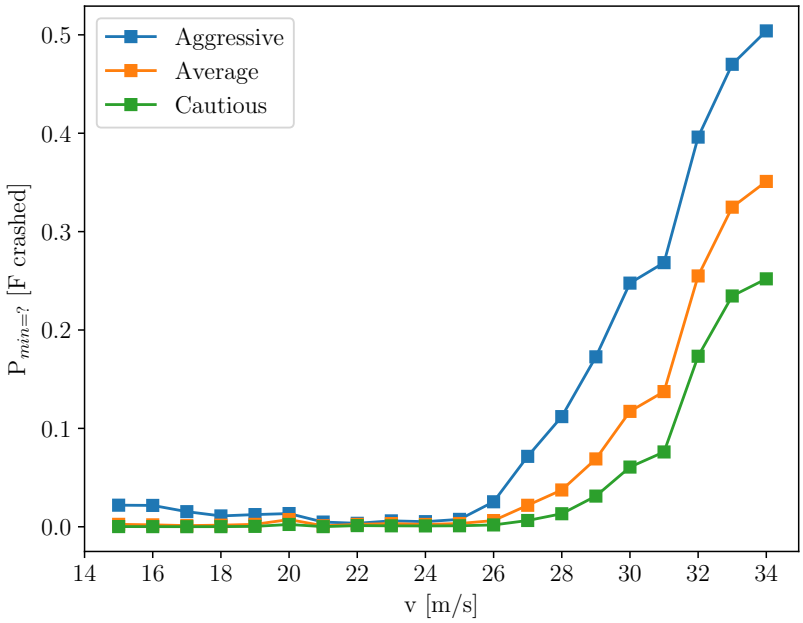


b. ADAS

Figure 5.1 Plots of the variation of the value of the safety property with the initial velocity of the main vehicle for the conditions $v_1 = 20m/s$ and $x_{1,0} = 35m$.

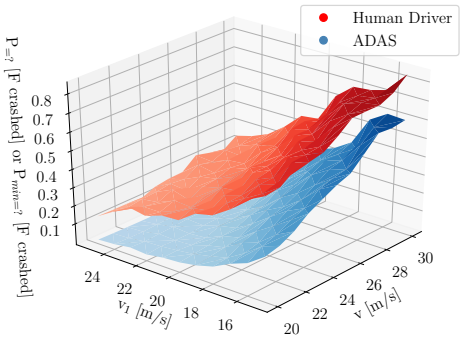


a. Human driver model

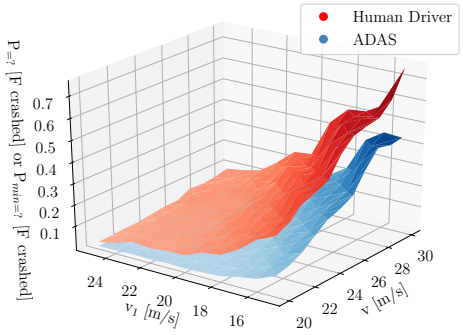


b. ADAS

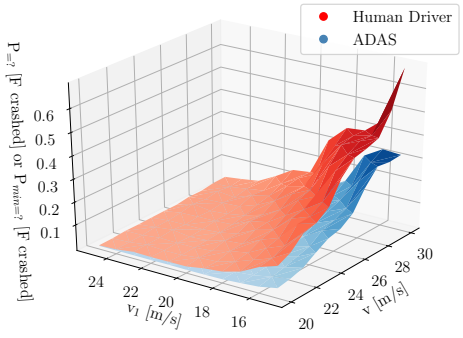
Figure 5.2 Plots of the variation of the value of the safety property with the initial velocity of the main vehicle for the conditions $v_1 = 22m/s$ and $x_{1,0} = 40m$.



a. Aggressive drivers

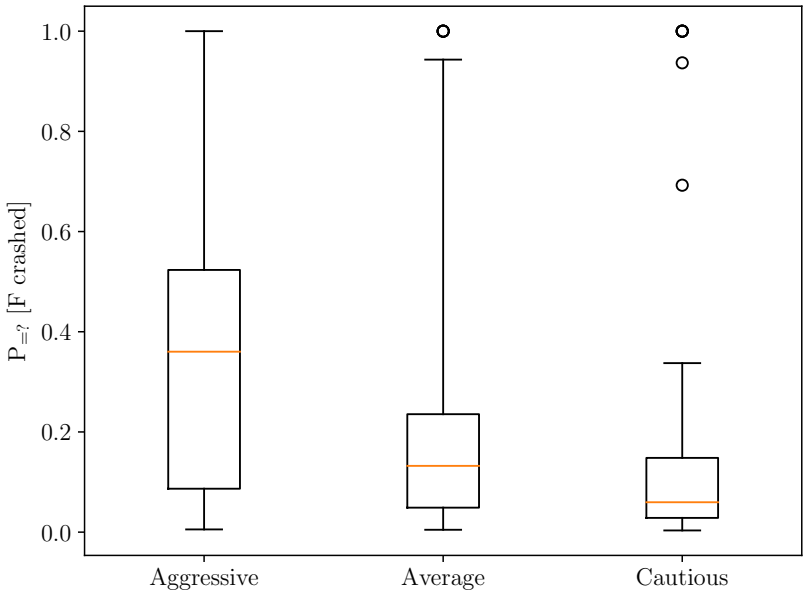


b. Average drivers

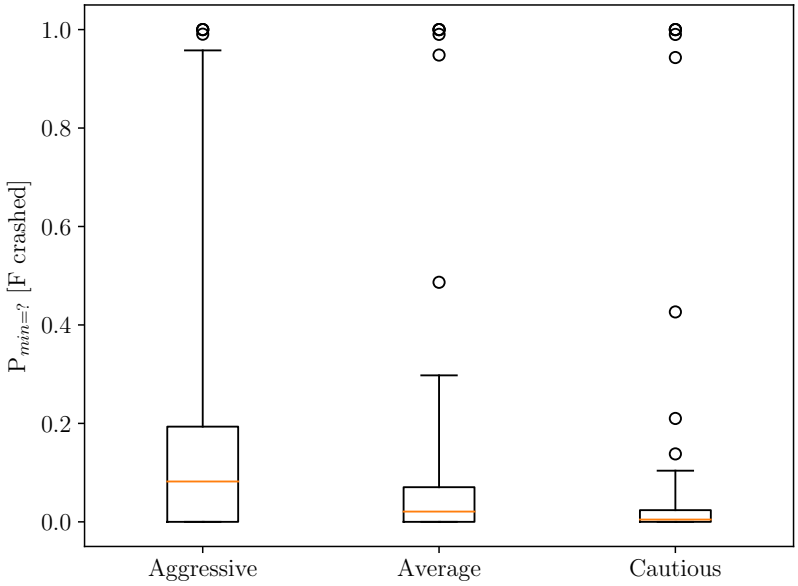


c. Cautious drivers

Figure 5.3 Variation of the value of the safety properties with the initial velocities of both vehicles for $x_{1,0} = 50m$.



a. Human driver model



b. ADAS

Figure 5.4 Box plot of the value of the safety properties for a randomly sampled population of 100 different initial conditions (equal in both cases).

5.1.2 Liveness Results

As introduced in Chapter 3, there are multiple liveness properties that can be considered when evaluating the human driver model. However, the most appropriate ones for inter-situational comparisons are the conditional properties of the type:

```
P=? [F (x=length) & (t<T) || F (x=length)]
```

for a given constant T.

In the case of the full system (human and ADAS), the model becomes an MDP and, as mentioned in Chapter 2, conditional properties are not so easily calculated in MDPs due to the existence of multiple adversaries. Despite this, it is possible to reason over lower bounds of these types of properties under the conditions of Proposition 2.3.1. For the propositions $\varphi_1 =$ `F (x=length) & (t<T)` and $\varphi_2 =$ `F (x=length)`, it is trivial that for any path $\pi \in Paths(s)$: $\pi \models \varphi_1 \Rightarrow \pi \models \varphi_2$. Furthermore, it can also be assumed that for:

$$\sigma' \in \arg \sup_{\sigma \in Adv} P_s^{\sigma}(\varphi_1 \mid \varphi_2) \tag{5.1}$$

it is true that:

$$P_s^{\sigma'}(\varphi_1) = p_{\max,s}(\varphi_1) \tag{5.2}$$

since intuitively the interest is in adversaries which maximise the conditional property through the maximisation of the probability of eventually reaching `(x=length) & (t<T)` (as this is the comparable case in the human driver model, and optimised in the). Thus, under the conditions of Proposition 2.3.1, it is possible to write:

$$p_{\max,s}(\texttt{F (x=length) \& (t<T) | F (x=length)}) \geq \frac{p_{\max,s}(\texttt{F (x=length) \& (t<T)})}{p_{\max,s}(\texttt{F (x=length)})} \tag{5.3}$$

Therefore, a lower bound on the conditional probability, ζ, can be obtained using the quantitative properties:

$$\zeta = \frac{\texttt{Pmax=? [F (x=length) \& (t<T)]}}{\texttt{Pmax=? [F (x=length)]}} \tag{5.4}$$

At this point it should also be noted that (as presented in Section 4.2.1):

$$p_{\max,s}(\texttt{F x=length}) = 1 - p_{\min,s}(\texttt{F crashed}) \tag{5.5}$$

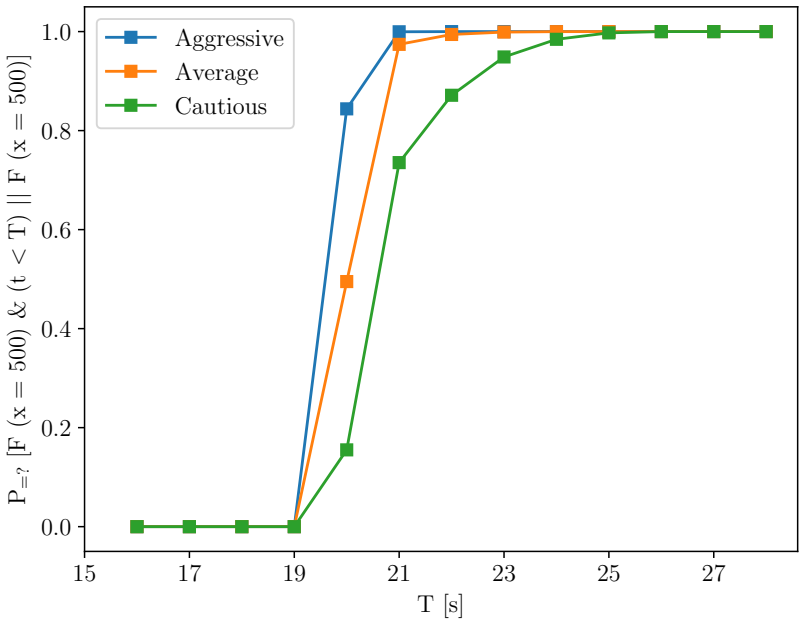
so long as $p_{\max,s}(\texttt{F (crashed | x = length)}) = p_{\min,s}(\texttt{F (crashed | x = length)}) = 1$ (completeness property for the MDP). Therefore, ζ can be re-written as:

$$\zeta = \frac{\texttt{Pmax=? [F (x=length) \& (t<T)]}}{1 - \texttt{Pmin=? [F crashed]}} \tag{5.6}$$

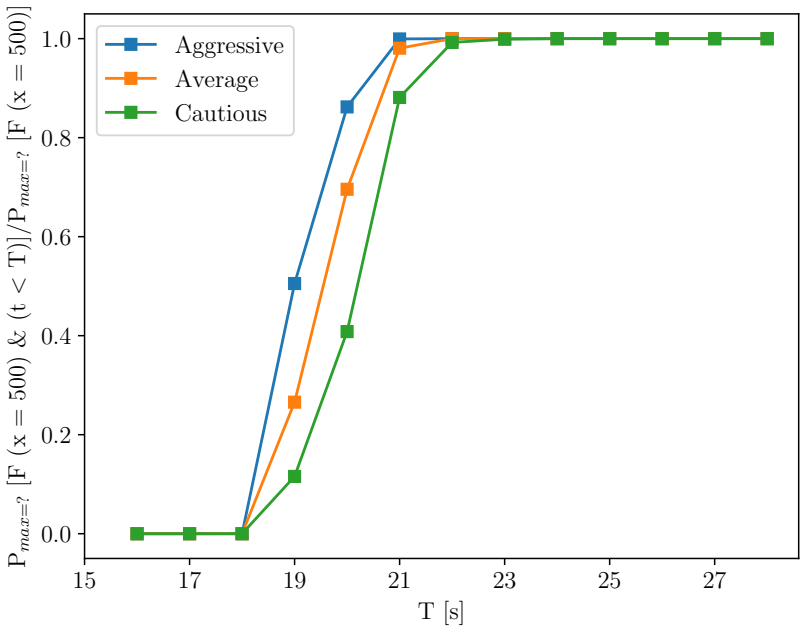
As was the case with the safety properties, model checking the entire space of possible initial conditions would be infeasible given the time frame of this dissertation. Similarly, two methods were designed to compare both systems at the liveness level:

- **Temporal variation of liveness**: for two randomly selected tuples $(v, v_1, x_{1,0})$, the value of all the pertinent liveness properties ($T \in \{16, ..., 28\}$) was obtained. The two tuples were randomly generated as $(v, v_1, x_{1,0}) = (21, 19, 70)$ and $(v, v_1, x_{1,0}) = (26, 22, 46)$, and the results are presented in Figures 5.5 and 5.6, respectively.

- **Box plots**: for a randomly generated sample of 100 different initial conditions of $(v, v_1, x_{1,0})$, obtain box plot which represents the distribution of the values of two liveness properties using appropriate values of T (given the sample) for this sample and for each driver profile. After the sample was generated, it was determined that $T = 21$ and $T = 22$ were representative values (in the sense that they are great enough to avoid all values of both properties being 0, and not so large that all values for all scenarios are 1; these situations would make it impossible to compare the human and the full system). The results are presented in Figures 5.7 and 5.8.

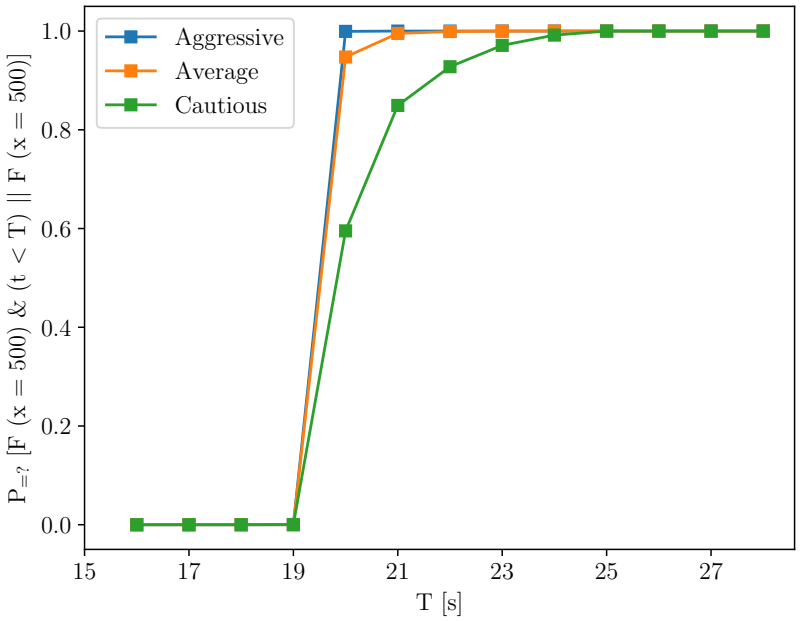


a. Human driver model

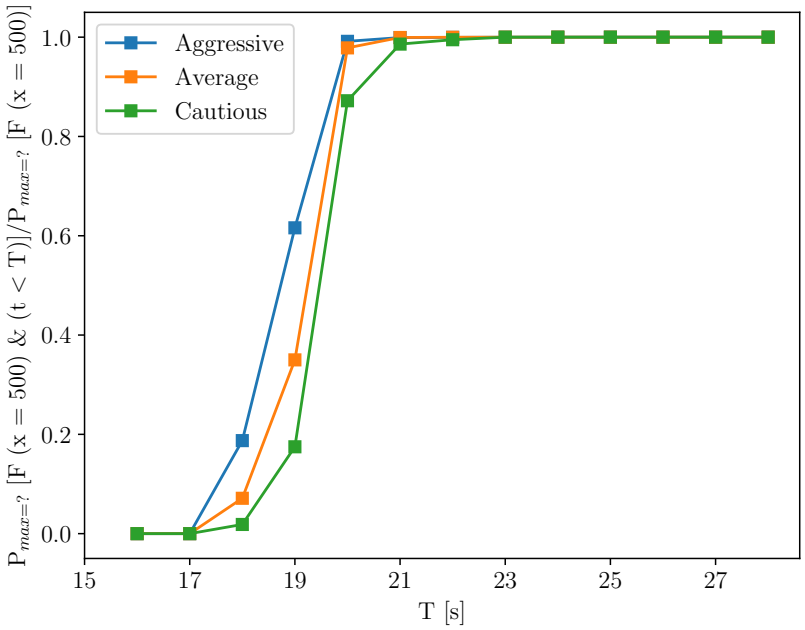


b. ADAS

Figure 5.5 Plots of the variation of the value of the liveness properties with the value of T (in s) for the initial conditions $v = 21m/s$, $v_1 = 19m/s$ and $x_{1,0} = 70m$.

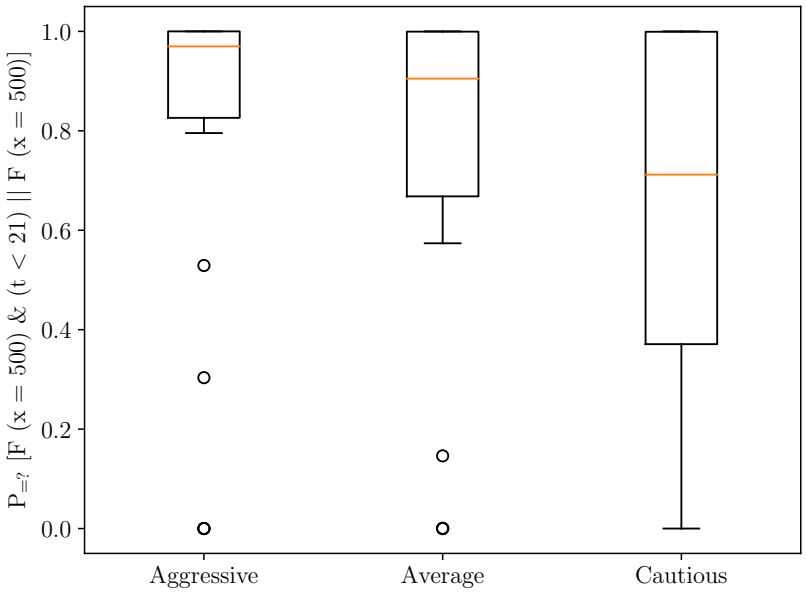


a. Human driver model

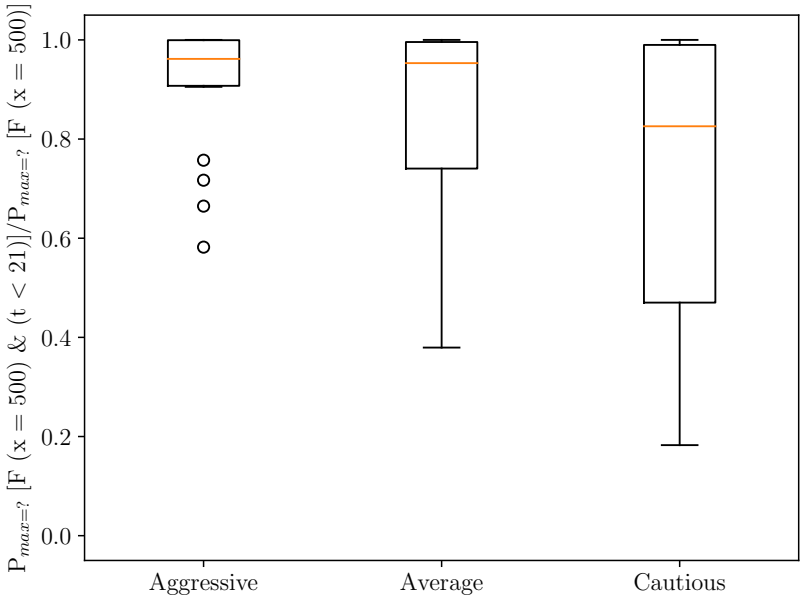


b. ADAS

Figure 5.6 Plots of the variation of the value of the liveness properties with the value of T (in s) for the initial conditions $v = 26m/s$, $v_1 = 22m/s$ and $x_{1,0} = 46m$.

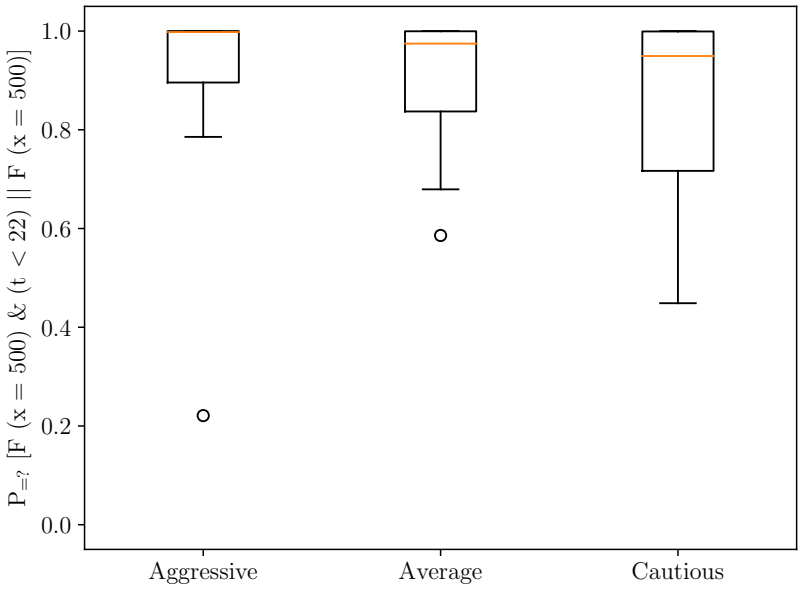


a. Human driver model

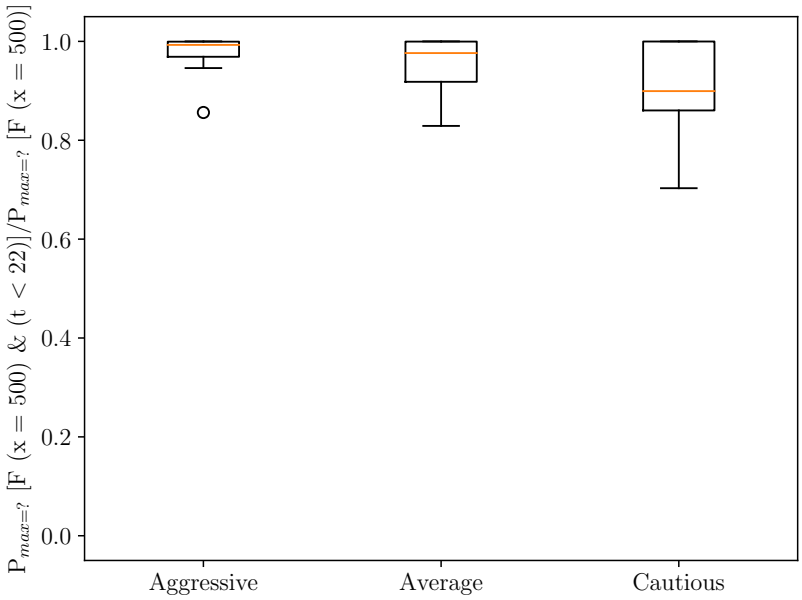


b. ADAS

Figure 5.7 Box plot of the value of the liveness properties for $T = 21s$ for a randomly sampled population of 50 different initial conditions (equal in both cases).



a. Human driver model



b. ADAS

Figure 5.8 Box plot of the value of the liveness properties for $T = 22s$ for a randomly sampled population of 50 different initial conditions (equal in both cases).

5.1.3 Discussion

The discussion of the results presented in Sections 5.1.1 and 5.1.2 is threefold in nature. Firstly, the comparison between the different driver classes within each of the individual systems (i.e. human driver model and the human and ADAS system) is presented. Secondly, the results of the experiments of the human driver alone to the full system in both safety and liveness are discussed (within the same driver profile). Finally, an overall discussion is presented on the extent of the validity of such comparisons through the drawbacks of the metrics used.

With respect to the individual driver classes presented in both the human driver model and the human and ADAS system, the results overwhelmingly support the initial idea that aggressive drivers perform the worse in terms of the safety metrics, followed by average drivers and finally cautious ones. While this is observable in the individual test cases (Figures 5.1 and 5.2), Figure 5.4 firmly supports this conclusion within both systems. In terms of liveness, the reverse is true, with aggressive drivers outperforming average drivers who in turn outperform cautious drivers. Once more, while individual test cases support this result (Figures 5.5 and 5.6), Figures 5.7 and 5.8 emphatically reinforce it. Intuitively, this is what was expected from the construction of the model.

In terms of the comparison between the human driver model and the human driver with the ADAS system, it becomes necessary to interpret the plots while considering the difference in the metrics used.

In the safety evaluation, as mentioned in Section 5.1.1, the results are directly comparable as the metrics are the equivalent of one another for the representation of the models of both systems (i.e. a DTMC for the human driver model and an MDP for the human and ADAS system). The most useful comparison in terms of the test cases in this metric is presented in Figure 5.3, as the 3D plots are divided into the three categories of drivers considered. It is observable that the introduction of the ADAS increases safety for all situations considered in this test case (i.e. reduces the value of the safety property). From the more general overview seen in Figures 5.7 and 5.8, the same conclusion is drawn for each driver class, with the 25%, 50% (median) and 75% quartiles being lower for the system with the ADAS than those for the human driver alone.

With respect to liveness, the comparison needs to be carefully drawn, as the conditional property in the human driver model alone (DTMC) is not the equivalent of the metric chosen for liveness in the human and ADAS system (MDP). However, ζ is proven to be the lower bound of the conditional property in the human and ADAS system under the conditions presented in Section 5.1.2. From Figures 5.5 and 5.6, it can be observed that, for each T, the values of ζ are always greater or equal than those of the conditional property of the human driver model. The same result can be seen in Figures 5.7 and 5.8 for $T = 21$ and $T = 22$, respectively, for the 25% and 75% quartiles. Given that ζ is the lower bound of the maximal conditional property, it can be concluded that the system with the ADAS outperforms the human driver alone in this metric.

Despite the results pointing to the improvements in terms of the safety and liveness properties the ADAS designed brings, they should be taken with care, as there are drawbacks in the metrics used.

The first drawback has to do with the fact that the comparison in the human and ADAS deals with minimal and maximal properties due to the existence of adversaries in MDPs. Thus, the fact that the safety property is lower in this case than in the human driver model and the liveness properties are higher, does not mean that there is a feasible strategy where both safety and liveness are optimal and outperform the human driver (there might be a trade-off). This is not the case with the human driver model, where it is known that both quantitative properties are satisfied simultaneously. In order to evaluate this feasibility properly, Pareto curves should be generated for each individual scenario and it should be seen whether there exists a Pareto point where both values outperform the human driver. This analysis is time consuming with the tools currently available (i.e. PRISM and Storm) and was therefore infeasible in the time frame of this dissertation.

An additional drawback in terms of the liveness properties in the full system (i.e. human driver and ADAS system) case is that these are defined as lower bounds which are not proven to be tight. In fact, in Figure 5.8, while the 25% and 75% quartiles appear to be higher in the case of the full system than the human driver model, this is not the case with the median, which is at similar values (or even lower for the class of cautious drivers). From the rest of the data, it would appear that this value might be the result of the fact that the lower bound is not tight and might be underestimating the value of the maximal conditional property for the sample

population used. However, this is not certain, and it might also be that the full system is outperformed, in terms of the liveness properties, by the human driver in certain conditions. The use of the unconditional liveness properties here would be meaningless, as it would incur in the safety bias first presented in Section 3.3.3. A solution to this problem would be to develop the tools to model check conditional properties in MDPs, however, this was outside the scope of this dissertation and would be highly time consuming.

5.2 Additional Experiments

While outside the scope of the main goal of the dissertation of evaluating safety and liveness, additional properties can be used to synthesise ADAS which enforce other types of behaviours.

Left Lane Penalising

When a driver attempts an overtake of the lead vehicle, it should do so by performing two lane changes, one from the current lane to the one immediately to the left, and another one to return to the original lane after it has passed the lead vehicle. The manoeuvre should be performed safely, yet as quickly as possible to avoid traffic jams [7]. With this in mind, properties can be specified to assert such a regulation to the ADAS to guarantee compliance.

In particular, the property should penalise the usage of the left lane ($lane = 2$). For this purpose, the following reward structure was designed:

```
rewards
  [] lane = 1: 0;
  [] lane = 2: 1;
endrewards
```

And the property:

```
Rmin=? [C]
```

minimises the value of this property, and, therefore, the time spent in the left lane. Thus, the following multi-objective property penalises the time spent on the left lane,

while maintaining safety:

```
multi(Pmin=? [F crashed], Rmin=? [C])
```

To exemplify the use of this reward structure and property, the scenario $(d_t, v, v_1, x_{1,0}) = (1, 30, 22, 50)$ (an aggressive driver with an initial speed of $30m/s$ and a lead vehicle at a distance of $50m$ going at $22m/s$) is considered. It should also be noted that the length of the road segment considered in this case was $400m$. By verifying this property, the Pareto curve presented in Figure 5.9

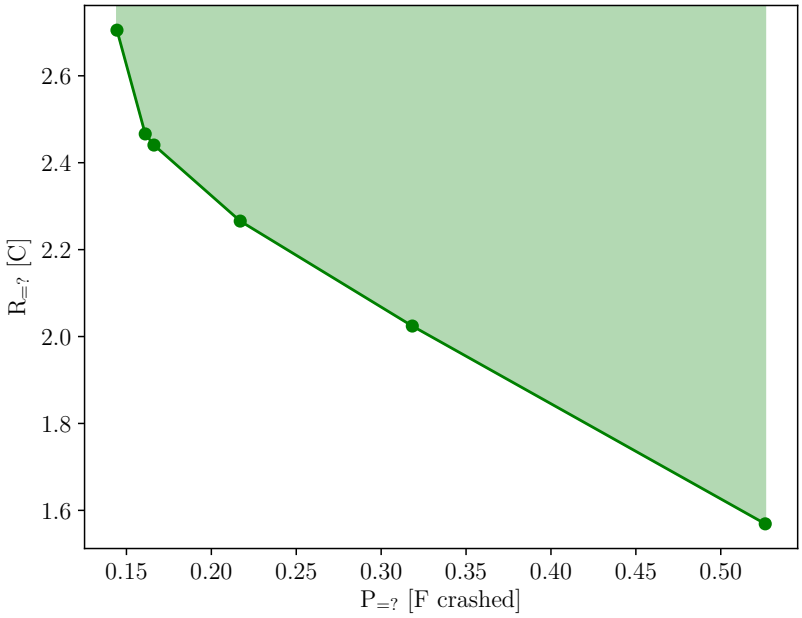


Figure 5.9 Pareto curve for the model checking of the lane penalising property.

Observing this curve, it is possible to conclude that it is feasible to have `R<=1.8 [C]`, and thus the following property:

```
multi(Pmin=? [F crashed], R<=1.8 [C])
```

yields an adversary which can be used to generate a sample path in the model. Similarly to what is presented in Section 3.4, a simulator was created to allow for the visualisation of a path in the model (the code for which is shown in Appendix B.3). The results of the simulation are presented in Figure 5.10.

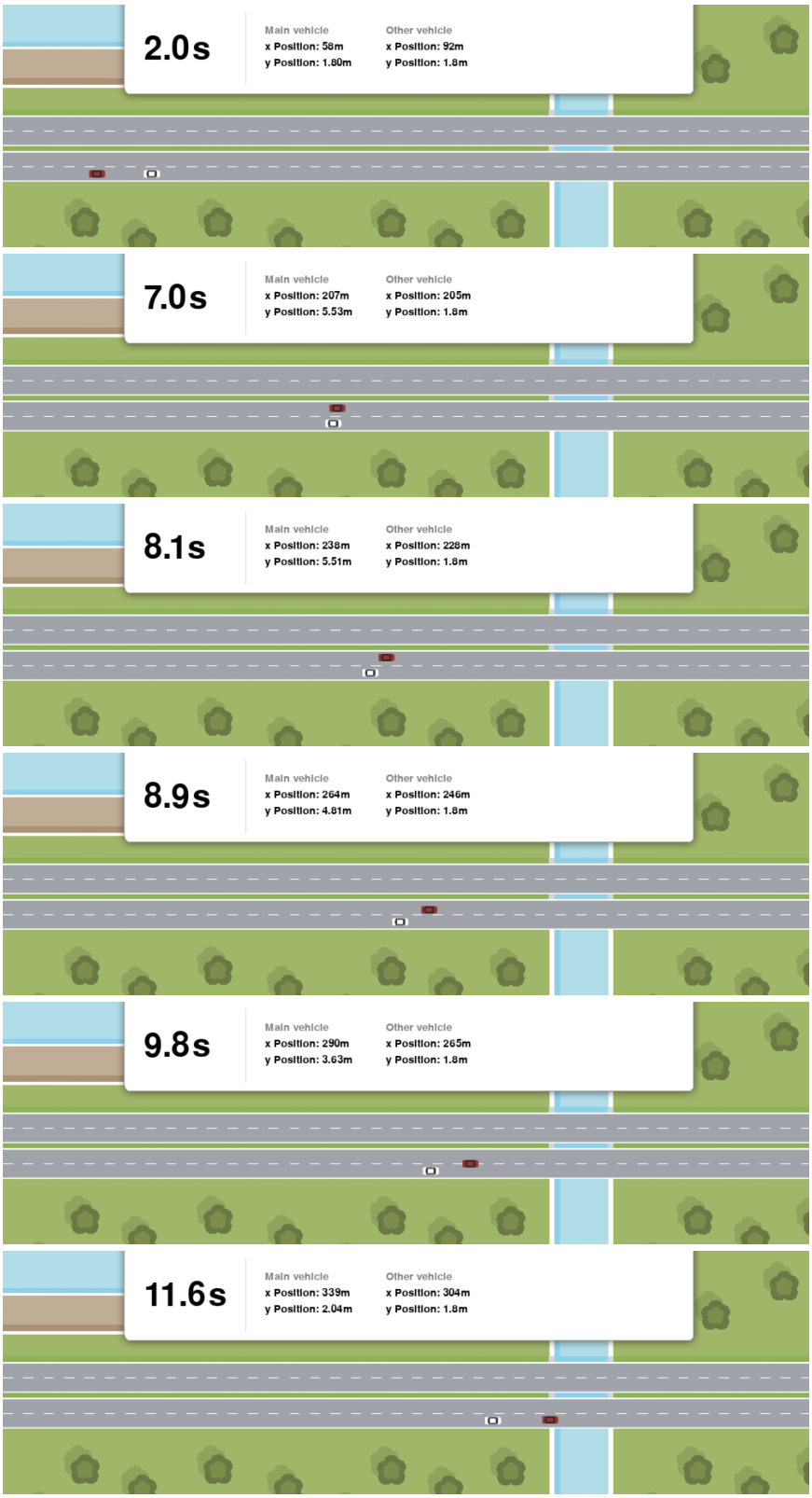


Figure 5.10 Snapshots of the simulator for one of the paths of the model.

Unsafe by Construction

Until this point, the assumption has been that the goal of the assistance system was to enforce safety and liveness. However, assume that a bad actor has access to the model and the deployment process used in the vehicle. In such a case, it becomes important to evaluate the damage that such an actor could do. Take the same scenario as above, $(d_t, v, v_1, x_{1,0}) = (1, 30, 22, 50)$, and consider the multi-objective property:

```
multi(Pmax=? [F crashed], P>=0.5 [F x>=100])
```

This property maximises the probability of crashing, while requiring the vehicle to drive for at least $100m$ with probability 0.5 before the crash happens. By model checking this property, the maximum value of `P=? [F crashed]` comes out to be 0.5326, and the adversary obtained permits the generation of paths such as the one presented in the simulator frames in Figure 5.11. It should be noted that the value of `P=? [F crashed]` in the human driver model is 0.3935, and thus the bad actor has successfully increased the probability of crashing.

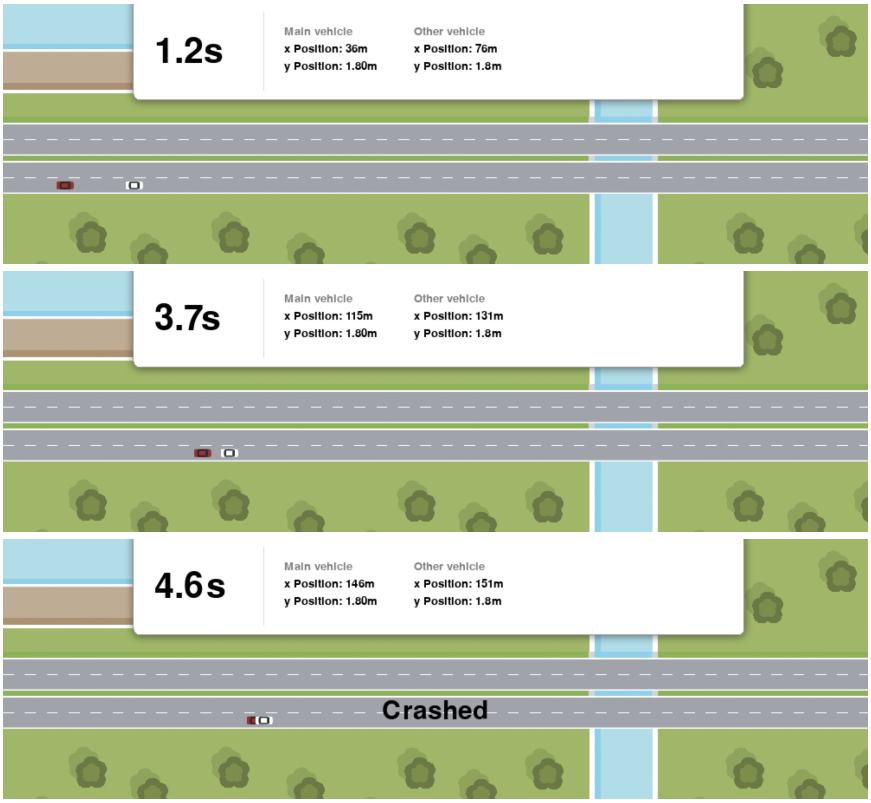


Figure 5.11 Snapshots of the simulator for one of the paths of the model.

Chapter 6

Conclusions and Future Work

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

Appendix A

Code for Human Driver Modelling

A.1 ACT-R Implementation (Matlab)

A.1.1 Main Loop

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.1.2 Control Module

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non

risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.1.3 Decision Making Module

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.1.4 Monitoring Module

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.2 Control Abstraction (Matlab)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.3 Decision Making and Monitoring Abstraction (Matlab)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.4 Probabilistic Two Module Model Generator (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.5 Prism Automatic Model Checker (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.6 Storm Automatic Model Checker (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur portti-

tor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

A.7 Simulator (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

Appendix B

Code for the Advanced Driver Assistance Systems

B.1 MDP Model Generators for Different ADAS (Python)

B.1.1 Fully Compliant Drivers in Decision Making

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

B.1.2 Partially Compliant Drivers in Decision Making

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

B.1.3 Additive Linear Control

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

B.1.4 Additive Steering Control

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula

purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

B.2 Pareto Curve Generator (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

B.3 Strategy Synthesis and Simulator (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

B.4 Prism Automatic Model Checker (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

Appendix C

Code for the Experimental Results and Evaluation

C.1 Plot Generator for Human Driver Model (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

C.2 Plot Generator for the Decision Making and Full Control ADAS (Python)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque accumsan magna eu metus congue, vitae tristique enim commodo. Aliquam consectetur porttitor nisl, eget pulvinar quam venenatis eget. Aliquam erat volutpat. Nullam at ipsum

at felis tincidunt consequat non sed mi. Morbi placerat iaculis dapibus. Aliquam non risus ac urna interdum pretium. Donec maximus, massa vitae suscipit congue, ligula purus sagittis magna, vitae interdum enim dui sed felis. Curabitur mollis eleifend diam a varius. Vivamus eu porttitor lorem. Mauris sagittis ultrices ligula, a euismod mi efficitur a. Aenean molestie sapien a nibh suscipit, nec venenatis urna varius. Sed eu ex id lacus tincidunt ultricies. Aenean quam massa, dignissim ac suscipit posuere, varius sit amet turpis. Nunc semper turpis ullamcorper elit venenatis, in molestie nulla fermentum.

Bibliography

[1] PRISM Manual | The PRISM Language | Example 1, Dec 2010.

[2] John R Anderson. *The Architecture of Cognition*. Psychology Press, 2013.

[3] John R Anderson, Michael Matessa, and Christian Lebiere. ACT-R: A theory of higher level cognition and its relation to visual attention. *Human-Computer Interaction*, 12(4):439–462, 1997.

[4] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[5] Tina Balke and Nigel Gilbert. How do agents make decisions? A survey. *Journal of Artificial Societies and Social Simulation*, 17(4):13, 2014.

[6] Ewin R Boer and Marika Hoedemaeker. Modeling driver behavior with different degrees of automation: A hierarchical decision framework of interacting mental models. In *Proceedings of the 17th European annual conference on human decision making and manual control*, pages 63–72, 1998.

[7] John Carr. State "keep right" laws.

[8] Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. On stochastic games with multiple objectives. In *International Symposium on Mathematical Foundations of Computer Science*, pages 266–277. Springer, 2013.

[9] Edmund M Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model checking and the state explosion problem. In *Tools for Practical Software Verification*, pages 1–30. Springer, 2012.

[10] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification*, pages 592–600. Springer, 2017.

[11] Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 53–113. Springer, 2011.

[12] Iuliia Kotseruba and John K Tsotsos. A review of 40 years of cognitive architecture research: Core cognitive abilities and practical applications. *arXiv preprint arXiv:1610.08602*, 2016.

[13] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 200–204. Springer, 2002.

[14] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 220–270. Springer, 2007.

[15] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *International conference on computer aided verification*, pages 585–591. Springer, 2011.

[16] Chak Lam. Driver assistance using cognitive modelling and strategy synthesis.

[17] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

[18] Andrew Liu and Dario Salvucci. Modeling and prediction of human driver behavior. In *Intl. Conference on HCI*, 2001.

[19] Highway Capacity Manual. Highway capacity manual. *Washington, DC*, 11, 2000.

[20] Austin Mohr. A survey of state abstraction techniques for markov decision processes.

[21] Dario Salvucci, Erwin Boer, and Andrew Liu. Toward an integrated model of driver behavior in cognitive architecture. *Transportation Research Record: Journal of the Transportation Research Board*, (1779):9–16, 2001.

[22] Dario D Salvucci. Modeling driver behavior in a cognitive architecture. *Human factors*, 48(2):362–380, 2006.

[23] Dario D Salvucci and Andrew Liu. The time course of a lane change: Driver control and eye-movement behavior. *Transportation research part F: traffic psychology and behaviour*, 5(2):123–132, 2002.

[24] Ron Sun. The importance of cognitive architectures: An analysis based on CLARION. *Journal of Experimental & Theoretical Artificial Intelligence*, 19(2):159–193, 2007.