Real-Time Decision Making by Driverless City Vehicles: A Discrete Event Driven Approach

Thesis · January 2010

DOI: 10.13140/RG.2.1.1952.4321

1 author:

Andrei Furda

17 PUBLICATIONS **198** CITATIONS

SEE PROFILE

Real-Time Decision Making by Driverless City Vehicles

A Discrete Event Driven Approach

Andrei Edmond Furda
Diplom-Informatiker

Griffith School of Engineering
Institute of Integrated and Intelligent Systems
Griffith University

Submitted in fulfilment of the requirements
of the degree of Doctor of Philosophy

July 2010

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.
Andrei Furda

To my parents.

Abstract

This thesis addresses the topic of real-time decision making by driverless (autonomous) city vehicles, i.e. their ability to make appropriate driving decisions in non-simplified urban traffic conditions. After addressing the state of research, and explaining the research question, the thesis presents solutions for the subcomponents which are relevant for decision making with respect to information input (World Model), information output (Driving Maneuvers), and the real-time decision making process.

The World Model is a software component developed to fulfill the purpose of collecting information from perception and communication subsystems, maintaining an up-to-date view of the vehicle's environment, and providing the required input information to the Real-Time Decision Making subsystem in a well-defined, and structured way.

The real-time decision making process consists of two consecutive stages. While the first decision making stage uses a Petri net to model the safety-critical selection of feasible driving maneuvers, the second stage uses Multiple Criteria Decision Making (MCDM) methods to select the most appropriate driving maneuver, focusing on fulfilling objectives related to efficiency and comfort.

The complex task of autonomous driving is subdivided into subtasks, called driving maneuvers, which represent the output (i.e. decision alternatives) of the real-time decision making process. Driving maneuvers are considered as implementations of closed-loop control algorithms, each capable of maneuvering the autonomous vehicle in a specific traffic situation.

Experimental tests in both a 3D simulation and real-world experiments attest that the developed approach is suitable to deal with the complexity of real-world urban traffic situations.

Acknowledgements

First, I would like to thank Professor Ljubo Vlacic for his support, encouragement, and valuable advice during my time at ICSL.

I would also like to thank INRIA's team IMARA for the financial support provided towards conducting the experimental work at their test track in Rocquencourt, and I am particularly grateful to Dr. Michel Parent, Laurent Bouraoui and Francois Charlot for their effort and assistance in performing the experiments.

Special thanks go to the students who supported me in implementing various components of the autonomous vehicle control software: Sebastian Boisse, Evan Baxter, Dan Boers, David Cook, and Brett Wood.

Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations and Symbols

API	Application Programming Interface
CAN	Controller Area Network
CPU	Central Processing Unit
DARPA	Defense Advanced Research Projects Agency (US)
DGPS	Differential GPS
DMU	Decision Making Unit
GPRS	General Packet Radio Service (mobile data service)
GPS	Global Positioning System
GUI	Graphical User Interface
ICSL	Intelligent Control Systems Laboratory (Griffith University)
IMU	Inertial Measurement Unit
INRIA	Intitut National de Recherche en Informatique et en Automatique (France)
INS	Inertial Navigation System
LIDAR	Light Detection and Ranging (also LADAR)
MCDM	Multiple Criteria Decision Making
MDF	Mission Data File (DARPA)
PID	Proportional Integral Derivative (control)
RDDF	Route Definition Data File (DARPA)
RNDF	Route Network Definition File (DARPA)
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
UGV	Unmanned Ground Vehicle
UML	Unified Modeling Language
WAAS	Wide Area Augmentation System
WGS	Wideband Global SATCOM
XML	Extensible Markup Language

Mathematical Symbols

$\mathbb{N}$	The set of natural numbers
$\mathbb{R}$	The set of real numbers
$\subset$	Subset
$\cup$	Union (set)
$\in$	Element (set)
$\sum$	Sum
max	Maximum
{..}	Set
$\times$	Cartesian product
$\vec{y}$	Vector y
$dim()$	Dimension

Chapter 1

Introduction

1.1 Motivation

Autonomous city vehicles[1] operating safely on public urban roads, coexisting with human-driven vehicles and pedestrians, have been a research topic and vision for many years [1]. The objective is to develop autonomous vehicles, which, in an ideal case, do not require a major change of road infrastructure. Such vehicles should be able to drive autonomously like human-driven vehicles, however offering significant benefits.

With respect to civilian applications[2], the vision of autonomous city vehicles is an attempt to face today's increasing traffic problems in cities worldwide, such as traffic congestion, a very high number of human-caused traffic accidents, inefficient consumption of energy, and pollution [2, 3]. In 2003, the costs caused by traffic congestion alone in the United States were 3.7 billion hours of travel delay, 2.3 billion gallons of wasted fuel, resulting in an estimated total cost of over USD 63 billion [4]. In 2007, the travel delay

[1] See definition 1.2.1 on page 3.

[2] The development of autonomous city vehicles is driven today not only by civilian objectives, but also by military interests. Since many research projects have been funded and supported by military institutions, such as the US Defense Advanced Research Projects Agency (DARPA), the interest and demand for such vehicles for military applications is obvious. However, this work focuses only on the civilian use of such vehicles, and does not further elaborate on their potential benefits for the defense industry sector. Nevertheless, a limited overview on relevant military related projects is included in the state of the art overview.

increased to 4.2 billion hours of travel delay, 2.8 billion gallons of fuel, and a total estimated cost of USD 87.2 billion [5].

The United Nations (UN) estimate the number of fatal road traffic accidents worldwide at 1.2 million per year, a number which makes road traffic accidents the second leading cause of death for people aged between 5 and 29. It is further estimated that 20 to 50 million people worldwide are injured every year in road traffic accidents [6]. Consequently, these high numbers force many governments to assign high priorities to traffic related problems. Like many governments worldwide, the Australian Government planned to reduce the road fatality rate to less than 5.6 per cent per 100,000 people by 2010 (from 9.3 in 1999) [6].

Besides the huge costs of wasted time and energy, traffic problems have a major impact on the quality of life. The United Nations Human Settlements Programme focuses on the Sustainable City, and addresses the transportation problem in urban areas. As part of this program, the UN World Urban Forum 2006 noted that the problems caused by conventional cars were a major challenge, especially in developing cities, and that urban transportation was a major defining factor for the quality of life in large cities. The objective for the near future is to provide equal access to safe and pollution-reduced intelligent transport systems, and make it accessible to all citizens, including the elderly [2].

Autonomous city vehicles have the potential to solve or at least reduce these transportation problems caused by conventional human-driven vehicles. Cooperative autonomous city vehicles[3] are most likely to operate in a more efficient way, for example by avoiding traffic congestions, by crossing intersections faster and safer, and by avoiding unnecessary stops. They will be able to autonomously drive to parking areas, and pick up passengers on request, eliminating the need to search for parkings. Another vision is, that in the near future, fleets of autonomous city vehicles, so called Cybernetic Transport Systems (CTS) will couple the benefits of public transport with those of today's cars [7, 8].

By replacing the human driver with an intelligent decision making &

[3] See definition 1.2.4 on page 5.

control system, the road accidents caused by human errors, for instance caused due to driver fatigue, distractions, or speeding, are most likely to be eliminated. Communicating autonomous vehicles will exchange information about their traffic environment and about their driving intentions, in order to guarantee accident-free traveling. Furthermore, autonomous city vehicles will provide door-to-door transportation services for all citizens, which is especially important for the elderly and for people with disabilities.

Consequently, autonomous city vehicles will have the potential to reduce travel times, pollution and energy consumption. Furthermore, such vehicles are likely to have a major contribution towards reducing the number of traffic accidents, and by providing access to all citizens, they are likely to significantly improve the quality of life for many people.

This thesis addresses the topic of real-time decision making by autonomous city vehicles, i.e. the vehicle's ability to make appropriate driving decisions according to the road traffic conditions. This research topic has already been recognized as one of the highest future challenges in the late 1990s, in a time when research efforts were still focusing on coping with much simpler traffic scenarios, such as highway traffic [9]. Nevertheless, the literature review reveals that this topic is yet to be addressed, especially from the viewpoint on how to ensure the safe operation in public, non-simplified urban traffic conditions.

1.2 Terms and Definitions

Although a large amount of literature has been published in various research areas of autonomous vehicles, the used terminology is still not standardized, often expressing similar or even identical ideas using different terms. This section defines and explains the terminology used throughout the remainder of this thesis.

Definition 1.2.1. *Autonomous/Driverless Vehicles*
Both terms *autonomous* and *driverless* denote a vehicle's ability to drive along urban roads and reach a specified destination without human driving

intervention.

With respect to its load, a driverless vehicle can be either *unmanned*, or *manned* if it is carrying passengers. Due to conceptual similarities, the term *robotic vehicle* is also often used to refer to autonomous vehicles. In principle, autonomous vehicles can be considered as road scale robotic vehicles which are able to transport passengers or loads.

According to their operation environment, autonomous vehicles are categorized into *aerial vehicles*, *ground vehicles*, and *underwater vehicles*. Unless otherwise specified, this thesis always refers to autonomous *ground* vehicles. Ground vehicles are further subdivided into *off-road (or terrain) vehicles*, and *road vehicles*. While off-road vehicles are designed to operate in any terrain, autonomous road vehicles, including both autonomous highway vehicles and autonomous city vehicles, require a road traffic environment. Driverless city vehicles are designed to operate on roads in urban areas, while autonomous highway vehicles are designed to operate only on highways.

Definition 1.2.2. *Driving Maneuver*
In this thesis, a driving maneuver is considered as a feedback control algorithm which is able to maneuver the vehicle in a specific traffic situation. Examples for driving maneuvers are stop-and-go, overtaking, or road/lane following. Details about driving maneuvers, as implemented in this thesis, are later addressed in Chapter 6.

In the literature, especially for this concept of decomposing the complex task of autonomous driving into a number of subtasks (i.e. driving maneuvers) of manageable complexity, many different terms have been used. Examples are: “behaviors” [10, 11], “autonomous behaviors” [12], “contextual behaviors” [13], “high-level behavior” [14], “planners” [15], “navigation modules” [16], “special maneuvers” [17], “context” [18], “driving behaviors” [19], or “navigation algorithm” [20].

Definition 1.2.3. *Real-Time Decision Making by Autonomous City Vehicles*
This term relates to the vehicle’s ability to make real-time (i.e. within the

specified time limits) decisions regarding the activation and execution of the most appropriate driving maneuver for any given traffic situation.

Definition 1.2.4. *Cooperative Autonomous Vehicles*
Autonomous vehicles which communicate and perform driving maneuvers in cooperation with each other are referred to as cooperative autonomous vehicles. For example, in a cooperative overtaking maneuver, the slower vehicle further slows down, in order to enable the overtaking vehicle to safely finish the driving maneuver [1].

1.3 Scope of this Thesis

This section elaborates on the research objective of this thesis and explains its scope within the framework of research topics in the area of autonomous city vehicles. After presenting an overview of research topics in subsection 1.3.1, subsection 1.3.2 explains the research objective of this thesis.

1.3.1 Overview of Research Topics for Autonomous City Vehicles

The research and development of autonomous city vehicles comprises a variety of related research areas, and each area comprises a large number of research topics, such as:

- Sensor Technology: autonomous vehicles require on-board sensor systems which are able to operate in any light, weather, and road conditions, while always delivering accurate and reliable data.

- Communication Technology: reliable wireless communication between autonomous vehicles (vehicle-to-vehicle (V2V)), as well as communication between the road infrastructure and autonomous vehicles (vehicles-to-infrastructure (V2I)) enable cooperation between vehicles, contribute to safe driving, and also improves the efficiency.

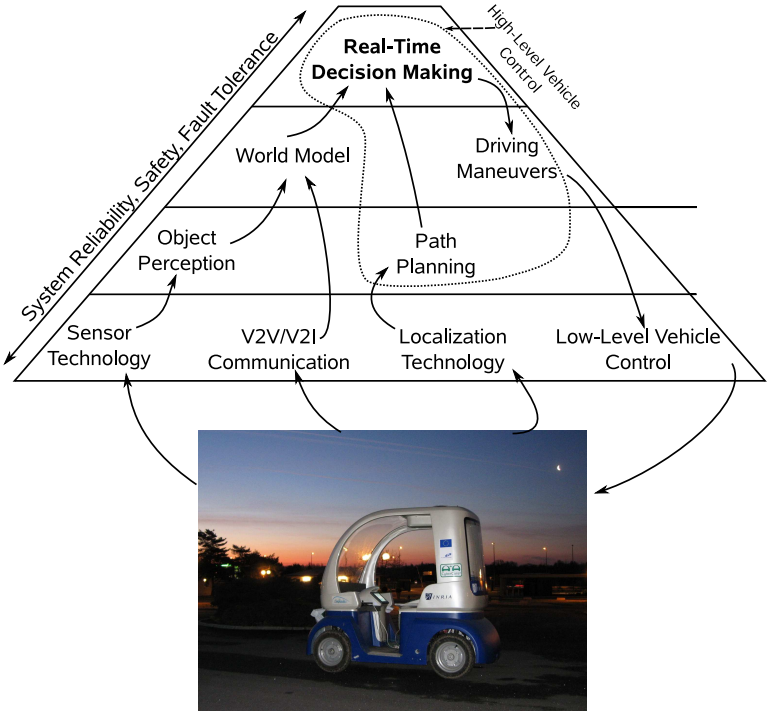


Figure 1.1: Research areas for autonomous city vehicles.

- Localization Technology: the autonomous vehicle's ability to know its road position is crucial for path planning and decision making. The currently used localization methods are based on the Global Positioning System (GPS) and Differential Global Positioning System (DGPS), sometimes in combination with Inertial Navigation Systems (INS). However, this technology still does not offer the required level of reliability and accuracy. Especially in urban areas, for instance between high buildings, or in tunnels, GPS receivers fail.

- Low-Level Vehicle Control: this includes the control of actuators for steering angle, accelerator, brakes, gearbox, etc.

- Perception Algorithms: perception algorithms operating on data obtained from one or multiple sensors are essential for autonous driving. Without the ability to reliably recognize the traffic features which are relevant for driving, such as traffic signs, road markings, obstacles,

pedestrians, vehicles, etc., driving is impossible. Furthermore, this research area addresses issues related to noisy and uncertain sensor data. The perceived information about the vehicle's traffic environment is collected, stored, and kept up-to-date in the World Model, which is a software component with the purpose to provide the Real-Time Decision Making subsystem with the required input information in a well-defined, structured way. The developed World Model is addressed in detail in Chapter 4.

- High-Level Vehicle Control: this research area includes tasks which are vehicle independent, and therefore on a higher abstraction level. Such tasks are for instance path planning, real-time decision making, and the execution of driving maneuvers.

- Overall System Reliability and Safety: since human safety will depend on such vehicles, autonomous vehicles will not be accepted unless they are safer than today's human driven cars. Furthermore, in order to be market competitive with conventional vehicles, autonomous vehicles need to reach at least a similar level of reliability. Consequently, each safety-related component needs to be developed focusing on road safety and reliability, for instance by integrating redundant components, fault detection, and fault tolerance.

1.3.2 The Research Objective of this Thesis

The main focus of this thesis is the problem of real-time decision making by autonomous city vehicles. Similar to a human driver, the real-time decision making & control subsystem of an autonomous vehicle needs to constantly make appropriate driving decisions with respect to the current traffic situation, in order to safely maneuver the vehicle in the complex urban traffic environment.

However, in addition to addressing this main problem, the following prerequisite aspects need to be addressed:

1. The autonomous vehicle's overall decision making & control system

architecture.

2. The required input information for decision making, the World Model.

3. The output information resulting from decision making.

Therefore, these aspects are addressed as well in detail in this thesis, and are included in its structure as elaborated in the following section.

1.4 Chapter Overview

The remainder of this thesis is structured as follows (Figure 1.2).

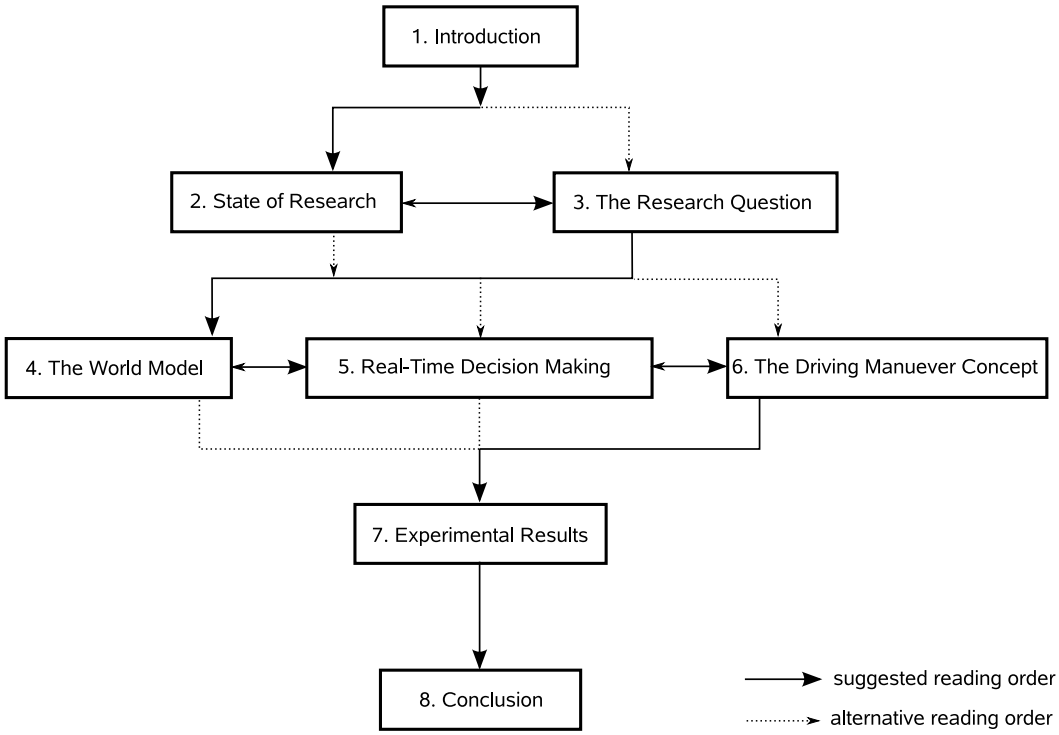


Figure 1.2: Chapter overview and possible reading order.

- Chapter 2 elaborates on the current state of the research of autonomous city vehicles, especially focusing on existing decision making algorithms and their limitations, as addressed by research groups worldwide. This

chapter reveals that, while numerous autonomous research projects have already produced advanced and impressive results for *simplified* road traffic conditions, none of the existing real-time decision making solutions for autonomous city vehicles is sufficient to cope with non-simplified, real-world urban traffic conditions. Therefore, this chapter explains and justifies the need for the developed real-time decision making solution presented in this thesis.

- Chapter 3 addresses and explains the research problem of real-time decision making for autonomous city vehicles and its scope within the decision making & control system of autonomous city vehicles. Furthermore, this chapter elaborates on the research objectives of this thesis, which are further addressed in the subsequent chapters.

- Chapters 4 presents the World Model, which provides the required information input about the vehicle's road traffic environment. The World Model is a software component developed to fulfill the purpose of collecting information from perception and communication subsystems, maintaining an up-to-date view of the vehicle's environment, and providing the required input information to the Real-Time Decision Making subsystem in a well-defined, structured way. This chapter explains the requirements which led to the development of the World Model, its object-oriented software architecture, its operational dynamics, and elaborates on the applied software design patterns.

- Chapter 5 addresses and presents the developed real-time decision making solution, i.e. the process of identifying the most appropriate driving maneuver to be performed under the given road traffic circumstances. Beginning with the system specification, this chapter explains the requirements and the problem definition which led to the developed two stage solution. While the first, Petri net based decision making stage is safety-critical[4] and focuses on selecting the feasible[5] driving maneuvers,

[4]Ensuring road safety.

[5]A driving maneuver is defined as feasible if it can be safely performed in a specific traffic situation, and is conforming to the road traffic rules.

the second, Multiple Criteria based decision making stage focuses on non safety-critical driving objectives, such as improving comfort and efficiency. Furthermore, this chapter elaborates on ensuring the real-time performance, error recovery from wrong driving decisions, and addresses relevant implementation details.

- Chapter 6 presents and explains the developed driving maneuver modeling concept. Driving maneuvers are feedback control algorithms, which are able to maneuver the vehicle in a specific traffic situation. From the viewpoint of Decision Theory, driving maneuvers are viewed as driving decision alternatives, and are the output of the real-time decision making subsystem. This chapter explains the requirements which led to the development of the driving maneuver modeling concept, the modeling method based on finite automata, which enables the implementation of driving maneuver algorithms with varying complexity using feedback control techniques. Furthermore, the developed modeling concept also addresses the implementation of driving maneuvers for error recovery, which are performed in order to avoid accidents by recovering from wrong driving decisions in unforeseen or suddenly changing traffic situations.

- Chapter 7 presents the experimental tests of the entire developed concept, including World Model, Real-Time Decision Making, and Driving Maneuvers. Using a 3D simulation environment, which models the real-world traffic environment at the Griffith University Nathan campus, the developed autonomous vehicle decision making & control system has been tested in various traffic scenarios, which included moving pedestrians, moving vehicles, and other static obstacles. Furthermore, this chapter presents the real-world experimental tests, which have been carried out in cooperation with the French research institute INRIA, using three vehicles interconnected over a wireless network. The results of both 3D simulation and real-world experimental tests show that the developed approach is applicable for real-world urban traffic conditions.

- Chapter 8 concludes this thesis by elaborating on its contributions, and by addressing future challenges in the development of autonomous city vehicles for real-world urban traffic conditions.

Chapter 2

State of Research

2.1 Introduction

Although autonomous ground, underwater, aerial, and planetary exploration robots of various types and purposes have been researched and developed since the late 1970s [21], and many projects focused on autonomous vehicles for highway [9, 22] and offroad [23–26] applications since the late 1990s, the efforts to develop autonomous vehicles for urban traffic is relatively new[1]. Furthermore, the so far undertaken autonomous city vehicles projects focused on very specific applications, and the developed solutions were limited to simplified traffic environments. The high complexity of non-simplified, real-world urban road traffic conditions has not been mastered by any autonomous city vehicle so far.

The following sections give an overview of autonomous city vehicle projects classified into two categories: civilian applications, and the DARPA Urban Challenge 2007. The French research institute INRIA[2], team IMARA, and the ICSL[3] research team at Griffith University, Australia, are among active research groups, which have been focusing on the development of autonomous city vehicles for civilian applications since the late 1990s.

On the other side, the major recent event driven by United States military

[1] Appendix A gives an overview of projects for highway and off-road applications.
[2] Intitut National de Recherche en Informatique et en Automatique
[3] Intelligent Control Systems Laboratory

objectives[4] was the DARPA Urban Challenge 2007, an autonomous vehicle race in a simulated urban environment.

2.2 Autonomous City Vehicle Projects for Civilian Applications

Following a series of research efforts focusing on the development of autonomous vehicles for European highways in the 1990s [9, 22, 28, 29], the European research teams' objectives evolved towards the development of autonomous city vehicles.

Griffith University and INRIA

The French research institute INRIA (Intitut National de Recherche en Informatique et en Automatique) developed the Cycab vehicle in 1997 (Figure 2.1). The golf cart sized electric vehicle was designed as a research platform for autonomous city traffic. It is still widely used today in many research programs worldwide, but especially at INRIA. The vehicle has been further developed, including different hardware and software configurations (e.g. dual steering).

The Cycab vehicle can only be controlled electronically, either automatically by a PC, or manually using a joystick. In its original configuration, it was equipped with a Motorola MC68332/MPC555 microcontroller, for controlling the vehicle's velocity and steering angle. In different configurations, INRIA used two software development approaches for the control software running on the microcontroller: Syndex and ORCCAD [31, 32].

A CAN bus vehicle interface allowed to control the vehicle using a PC. The originally presented prototypes did not integrate any sensors. However, the vehicle could be easily equipped with different sensors via the CAN bus interface or by connecting them directly to the PC. Later, the Cycab vehicle was manufactured by the French company Robosoft.

[4] The US Congress decided in 2001 to have one third of the ground combat vehicles operating unmanned by 2015 [27].

Figure 2.1: Cycabs autonomously traversing an unsignalled intersection [30].

In 2002, Griffith University's Intelligent Control Systems Laboratory (ICSL) and INRIA's IMARA Laboratory demonstrated a solution for on-road cooperative driving, a demonstration which was believed to be the first of its kind in the world [1, 30].

Figure 2.2: Mobile robots developed at ICSL.

The vehicle control algorithms were first tested with mobile robots, which were equipped with the same sensors as the Cycab vehicles during the demon-

stration (Figures 2.2, 2.1). The ICSL robots were able to follow a lane, to avoid static obstacles, to keep a safe distance to the front robot, detect dynamic obstacles, and traverse unsignalled intersections. Furthermore, the robots were able to communicate with each other or with a road infrastructure over a wireless network. This enabled them to perform driving maneuvers which require cooperation, such as traversing unsignalled intersections or predicting the driving maneuvers of other autonomous vehicles.

The same control software and sensors were used for experiments and a demonstration with Cycab vehicles. In the demonstration with the Cycab, the Cycab's low-level vehicle control, which controlled the vehicle's speed and steering angle using a dedicated microcontroller, received and executed commands which were sent over the CAN Bus interface by the ICSL control hardware and software (Figure 2.3).

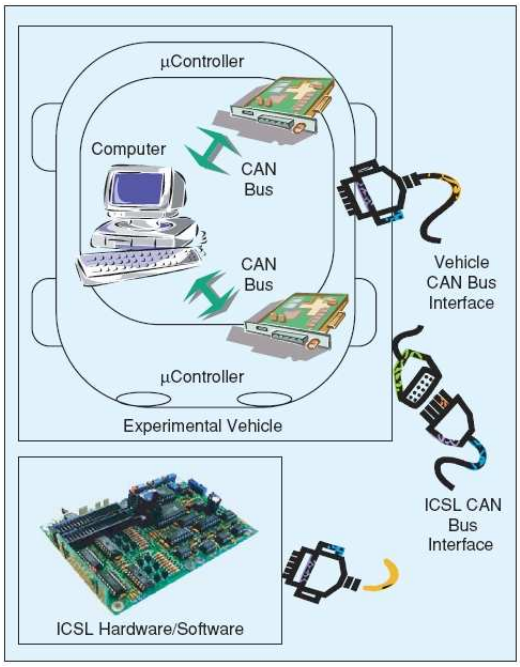


Figure 2.3: Block scheme of interfacing ICSL hardware with an experimental vehicle [1].

The vision was the development of cooperative autonomous vehicles able to coexist on the road with conventional human-driven vehicles. Using wireless communication, the vehicles performed driving maneuvers which

required inter-vehicle cooperation or synchronization, such as cooperative overtaking or traversing unsignalled intersections. In a cooperative overtaking maneuver, the slower vehicle further slowed down while being overtaken. At an unsignalled intersection, the vehicles were able to determine an order of traversal and to avoid possible deadlocking while waiting for each other to pass. Other driving maneuvers were stop-and-go, and obstacle avoidance.

However, at that time, the driving maneuvers were activated manually, on the spot, since a real-time decision making module, which should perform this task automatically with respect to the specific traffic situation, had not been developed yet [30]. The continuation of the project resulted in the development of the real-time decision making solution presented in this thesis.

Cybernetic Transport Systems - CyberCars / CyberMove and CyberC3

The Cybernetic Transport System (CTS) project was initiated in 1999 by a consortium of 15 European research institutes and companies. The project's objective was the development of an Intelligent Transport System based on road vehicles with fully automated capabilities [7, 8].

CTS is based on the idea of car sharing. Autonomous cars (so-called Cybercars) should be available to the public for short-range inner city transport as "a new form of public transport which bridges the gap between private cars and public transport" [7]. The expected benefits of CTS systems were improved mobility, a cleaner environment, less energy consumption, and improved road safety.

The first experimental CTS system started in 1997 at the airport in Amsterdam [33], the first publicly accessible CTS in the world was developed in Rotterdam (Rivium project) as part of the European CyberCars/CyberMove initiative. In order to avoid legal difficulties, the vehicles were operated only on private property, on a former bicycle path without public traffic. Although the system was widely accepted by the public, it was stopped after the testing phase of one year. Reasons for not continuing it were the lack of transport

capacity and insufficient reliability. However, other similar projects were planned in other European cities [33]. The CyberCars/CyberMove project was completed in 2004, and then continued with a follow-up project Cybercars2 (2006-2009).

Another closely related project, CyberC3 (Cybernetic Technologies for Cars in Chinese Cities), focuses on the development of new markets for the European CyberCars in Chinese cities. The project started in 2004 and was funded by "EC Asia IT&C Programme". Its objective was the transfer of European CyberCar technologies to China [34]. Using technologies and know-how from the CyberCar/CyberMove project, the objective was to develop a transport system of autonomous electric vehicles for the Shanghai Century Park and other sites in China.

The architecture of the CyberC3 system consisted of three wirelessly communicating subsystems (Figure 2.4):

- Central Control System,
- Vehicle System,
- Station System.

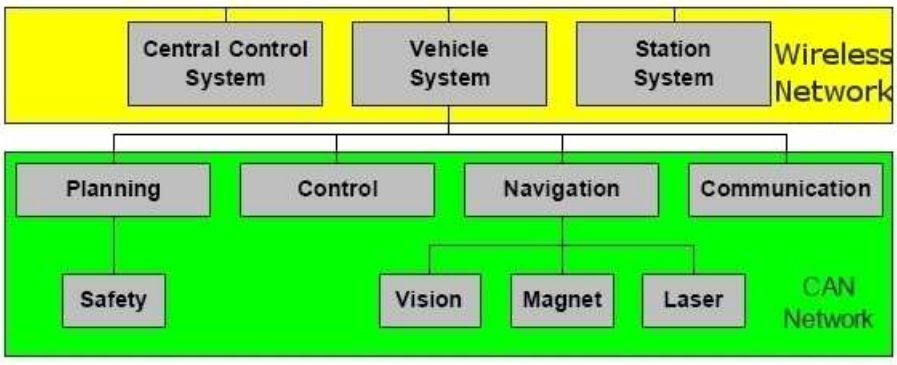


Figure 2.4: CyberC3 system architecture [34].

The vehicle system contained a navigation module, a planning module, a control module, and a communication module. The vehicle's position was determined by the guidance module using three different methods: magnets

in the road, vision and laser sensors. The magnetic guidance system was claimed to be the most reliable for outdoor use, and outperforming the vision system. A Sick LMS 291 LIDAR[5] sensor was used for obstacle detection.

Figure 2.5: CyberC3 vehicles [36].

Each vehicle's planning module determined the position by merging information from the magnetic guidance system, the vision system and the LIDAR sensor. The LIDAR sensor and the ultrasonic sensors were also used for the obstacle avoidance. As the vehicles were not able to leave the magnetic guidance path, they slowed down and eventually stopped whenever an obstacle was detected in front. The path following maneuver was implemented as a fuzzy PID controller, which controlled the vehicle's speed and steering angle over the CAN bus interface.

The Central Control System was responsible for monitoring the vehicles, for fleet management and logging tasks. Passengers were able to use the vehicle's user interface to select the destination station, to switch to manual mode or to press the emergency button. The Station System allowed the users to request a vehicle.

Since the vehicles were not able to leave the magnetic path, real-time decision making was not needed in the CyberC3 project.

[5]Light Detection and Ranging. Laser sensor for obstacle detection. A pulsed laser beam, reflected by a rotating mirror, is used to measure distances and angles to obstacles [35].

The Southwest Safe Transport Initiative

The Southwest Research Institute (SwRI) initiated the Southwest Safe Transport Initiative (SSTI) in 2006, with the objective of improving the performance and safety of vehicles, and developing autonomous vehicle technologies [37]. Today, SSTI is an ongoing research project.

The SSTI vehicle (Figure 2.6) is based on a 2006 Ford Explorer, and is equipped with an Ibeo Alaska XT LIDAR, an Oxford RT3052 GPS/INS, high resolution cameras, a drive-by-wire system, and an Intel Core 2 Quad and Duo Blade computer system.

Figure 2.6: The Southwest Research Institute SSTI Vehicle [37].

The SSTI vehicle's control software is based on the 4D/RCS architecture (addressed in Section A.1.2, [26]), and contains the following modules, called Navigation System Nodes (Figure 2.7):

- Route Management: provides a user interface through either a map, or for loading an RNDF[6] file.
- Advanced Navigation: responsible for crossing intersections, passing other vehicles, and reacting to detected pedestrians.
- Basic Mobility: generates low-level driving commands (called behaviors) for following a traffic lane, changing traffic lanes, performing turns, avoiding obstacles, following other vehicles, and stopping at stop signs.

[6]DARPA Route Network Definition File.

- Vehicle Path: generates controller commands to control the steering angle, vehicle heading, and lateral position on a path.
- Vehicle Controller: generates speed and steering angle commands.

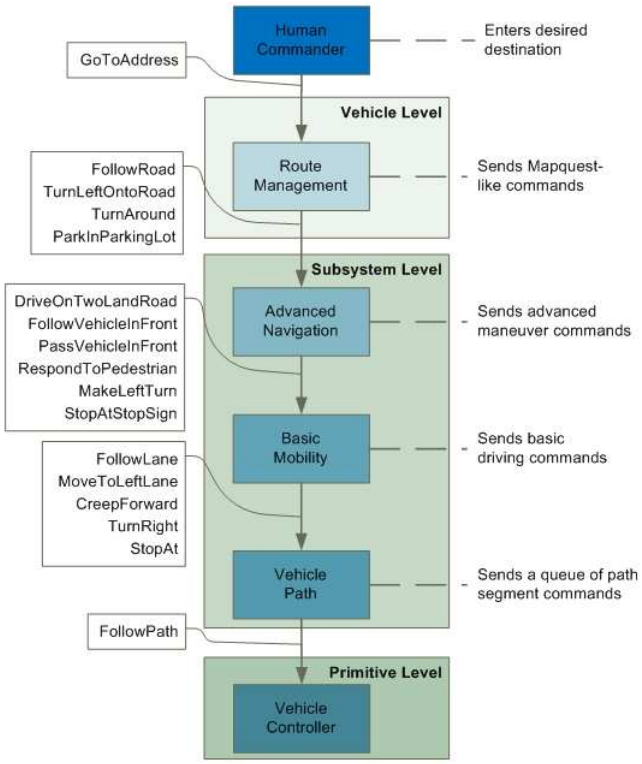


Figure 2.7: SSTI control software architecture [37].

The SSTI vehicle is able to follow vehicles, pass other vehicles, react to pedestrians and bicyclists, merge into traffic, and cross intersections. Additionally, the vehicle is able to change lanes, overtake, perform 3-point turns, and to replan the route to the destination, if necessary [37]. The vehicle's real-time decision making is based on checking of conditions, and reacting according to defined decision rules (Figure 2.8).

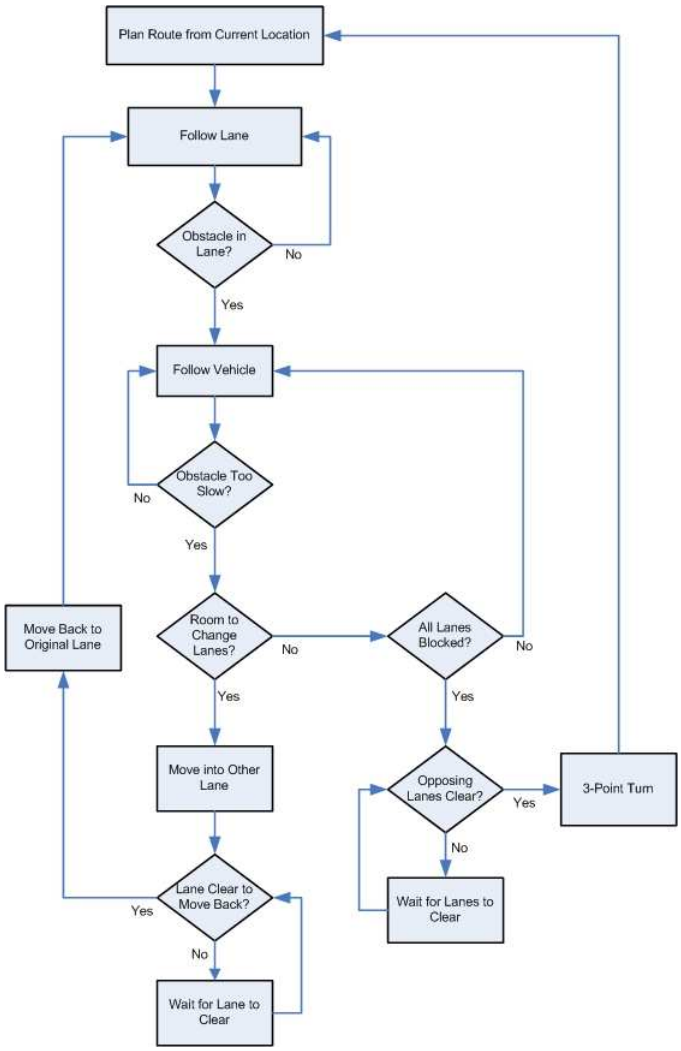
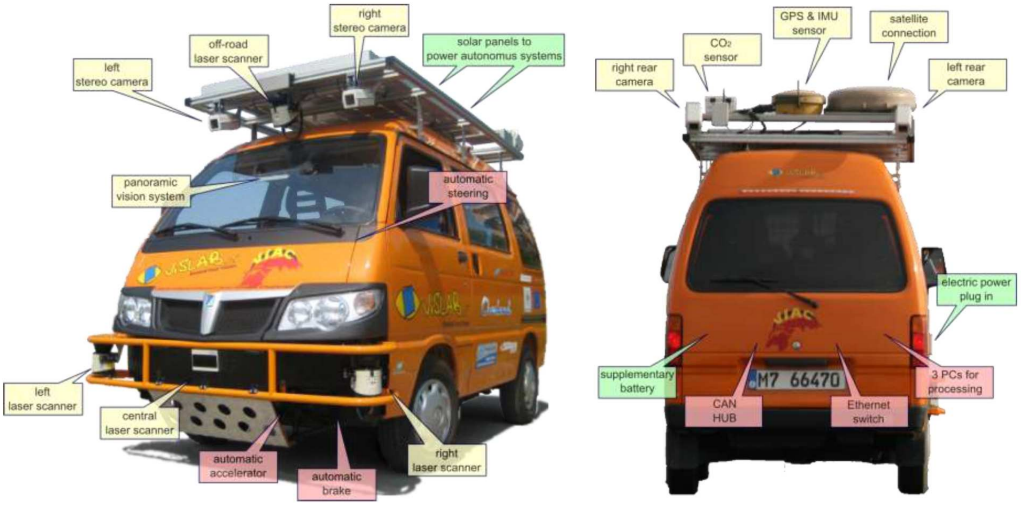


Figure 2.8: SSTI decision making for lane change due to obstacle [37].

VisLab's Autonomous Vehicle Journey from Parma to Shanghai

In 2009, the University of Parma's VisLab laboratory, directed by Prof. Broggi, announced the intention to develop autonomous vehicles able to drive autonomously 13,000km from Parma, Italy to Shanghai, China [38]. On October 28, 2010, VisLab's autonomous vehicles (Figure 2.9) reached their destination at the 2010 World Expo in Shanghai, China, after traveling over 15,000km in more than 3 months [39].



(a) (b)

Figure 2.9: VisLab's autonomous vehicle setup [39].

VisLab's autonomous vehicles were able to perform the following maneuvers and tasks [39]:

- following another vehicle,
- stop-and-go,
- path following,
- vehicle detection,
- road detection,
- pedestrian detection,
- obstacle detection,
- ditch and berm detection for off-road driving,

- panoramic vision system for 180 degrees monitoring,
- terrain slope estimation and mapping.

The VisLab team admits that *"during the trip some situations were identified in which the systems did not produce satisfactory results, sometimes making human interventions necessary."* [39].

Since this demonstration took place only recently, no details about the vehicle's control software architecture and decision making approach have been published yet.

Stadtpilot (Braunschweig, Germany)

The project "Stadtpilot"[7] aims to develop an autonomous vehicle for real-world urban traffic, able to drive autonomously on the ring road around the inner city of Braunschweig, Germany. The project is led by the University of Braunschweig, which uses the experience gained previously during the development of the DARPA Urban Challenge vehicle "Caroline" [40].

The Stadtpilot vehicle "Leonie" is based on a 2007 Volkswagen Passat, and is equipped with LIDAR, RADAR, and a drive-by-wire system (Figure 2.10).

Figure 2.10: The vehicles "Caroline" (left) and the Stadtpilot vehicle "Leonie" (right) [40].

[7] Translation: city pilot.

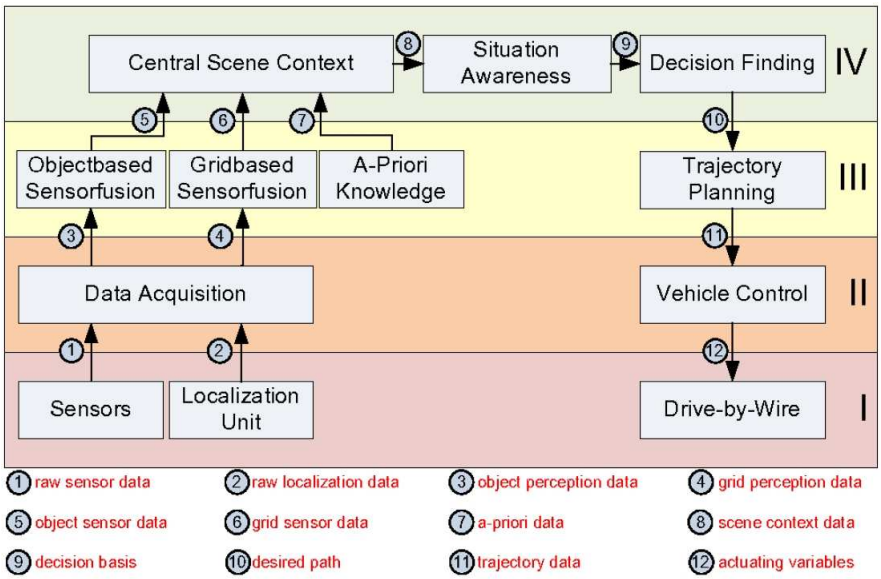


Figure 2.11: Stadtpilot control software architecture [40].

Leonie's control software architecture consists of four layers, with the decision making module in Layer IV (Figure 2.11). The decision making unit makes driving decisions based on a priori information in the form of a map, and based on information about the vehicle's surroundings, which is obtained from the on-board sensors.

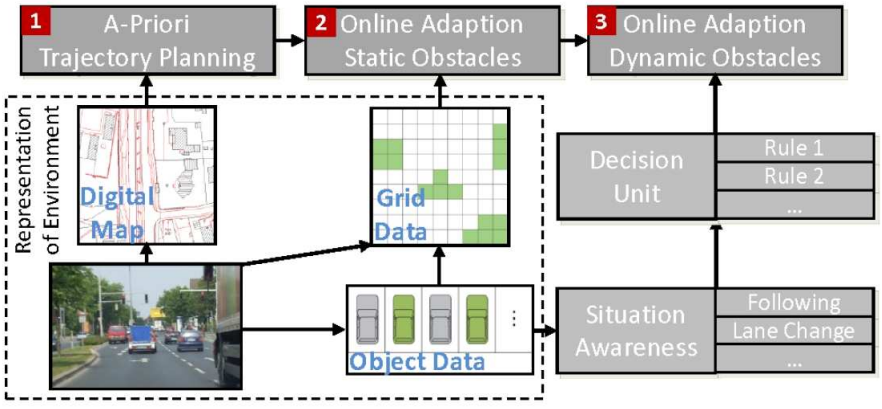


Figure 2.12: Stadtpilot's Decision Making Architecture [40].

A single data structure is used to store all required information, such as

positions of traffic lights, information about the number of traffic lanes, as well as the desired vehicle speed. The output of the decision making unit is provided in the form of drivable corridors, which are then optimized by smoothing splines using a so-called elastic band model (Figure 2.12) [40].

Google's Autonomous Vehicle Project

On 09. October 2010, Prof. Sebastian Thrun announced on Google's official blog that a team in cooperation with Google has *"...developed technology for cars that can drive themselves. Our automated cars, manned by trained operators, just drove from our Mountain View campus to our Santa Monica office and on to Hollywood Boulevard. They've driven down Lombard Street, crossed the Golden Gate bridge, navigated the Pacific Coast Highway, and even made it all the way around Lake Tahoe. All in all, our self-driving cars have logged over 140,000 miles. We think this is a first in robotics research. Our automated cars use video cameras, radar sensors and a laser range finder to see other traffic, as well as detailed maps (which we collect using manually driven vehicles) to navigate the road ahead. This is all made possible by Google's data centers, which can process the enormous amounts of information gathered by our cars when mapping their terrain. "* [41].

However, besides this news announcement, no further details about this project have been published so far.

2.3 The DARPA Urban Challenge 2007

The major recent event demonstrating the state of the art autonomous city vehicle technology was the DARPA Urban Challenge 2007 [27], an autonomous vehicle race in a simulated urban environment. However, although all vehicles competed in the same traffic environment, and had therefore identical decision making requirements, a direct comparison between their approaches is difficult due to varying terminology for similar ideas in the original publications. For instance, the vehicle "Boss" had a "behavioral system" consisting of "subcomponents" [20], while the vehicle "Junior" had

"navigation modules" executing "behaviors" or "actions" [16]. Nevertheless, the following is an attempt to give an overview based on a unified terminology.

The DARPA Urban Challenge took place on November 3, 2007, on a closed Air Force base in Victorville, California, USA. The prize was set to $2 million for the first autonomous vehicle able to drive 96km (60miles) in less than 6 hours, $1 million for the second place and $0.5 million for the third prize [42, 43]. The objective was to have *"autonomous ground vehicles executing simulated military supply missions safely and effectively in a mock urban area"*.

The race organizers attempted to create traffic conditions similar to a real urban scenario by including human-driven moving cars. However, compared to real urban traffic, the conditions were still simplified, as there were no pedestrians, bicycles, motorbikes, etc. Furthermore, the autonomous vehicles were not required to detect any traffic signs or signals [42]. These were provided as a priori information before the beginning of the race, along with navigation points (checkpoints) in a so-called Mission Data File (MDF). The a priori information was provided in the so-called Route Network Definition file (RNDF) [27].

Out of the eleven vehicles admitted for the finals of the race[8], six were able to finish the DARPA Urban Challenge 2007 (Figure 2.13). The winner was the vehicle "Boss" (Team Tartan Racing and Carnegie Mellon University), "Junior" (Team Standford Racing) achieved second place, and "Odin" (Team Victor Tango) achieved third place [42].

Boss (1st place)

The vehicle "Boss" (Figure 2.13.a) was developed by the "Tartan Racing Team", a cooperation of Carnegie Mellon University, General Motors, Caterpillar, and Intel [20].

The vehicle was based on a Chevrolet Tahoe, and was equipped with a GPS/IMU receiver, eleven LIDAR sensors of various types (six Sick LMS,

[8] 86 teams were initially accepted to participate, 35 were selected for the National Qualifying Event, out of which 11 were admitted to the finals [42].

(a) Boss (1st) [20]

(b) Junior (2nd) [16]

(c) Odin (3rd) [19]

(d) Talos (4th) [14]

(e) Little Ben (5th) [15]

(f) Skynet (6th) [11]

Figure 2.13: The six vehicles which were able to complete the DARPA Urban Challenge 2007, listed in the order in which they completed the race. Official ranking was given to places 1, 2, and 3.

one Velodyne HDL-64, two Continental ISF 172, two IBEO Alasca XT), five Continental ARS 300 Radar sensors, and two cameras. The computing unit consisted of ten 2.16-GHz Core2Duo processors.

Boss was able to detect static and dynamic obstacles, however it did not perform accurate obstacle classification. For instance, all moving obstacles were classified as vehicles if they were in a lane or parking space [20].

Boss' decision making approach was based on so-called "behavioral reasoning". Depending on where the vehicle was driving, three different driving maneuvers (called "behaviors") could be executed:

- Lane Driving,
- Intersection Handling,
- Achieving a Zone Pose (parking areas, crowded intersections).

Each of these driving maneuvers were composed of several so-called "sub-behaviors" (Figure 2.14). The "state estimator" determined the vehicle's position. Based on this information, the "goal selector" calculated the next destination point, which could be either on a traffic lane or zone (i.e. parking or intersection).

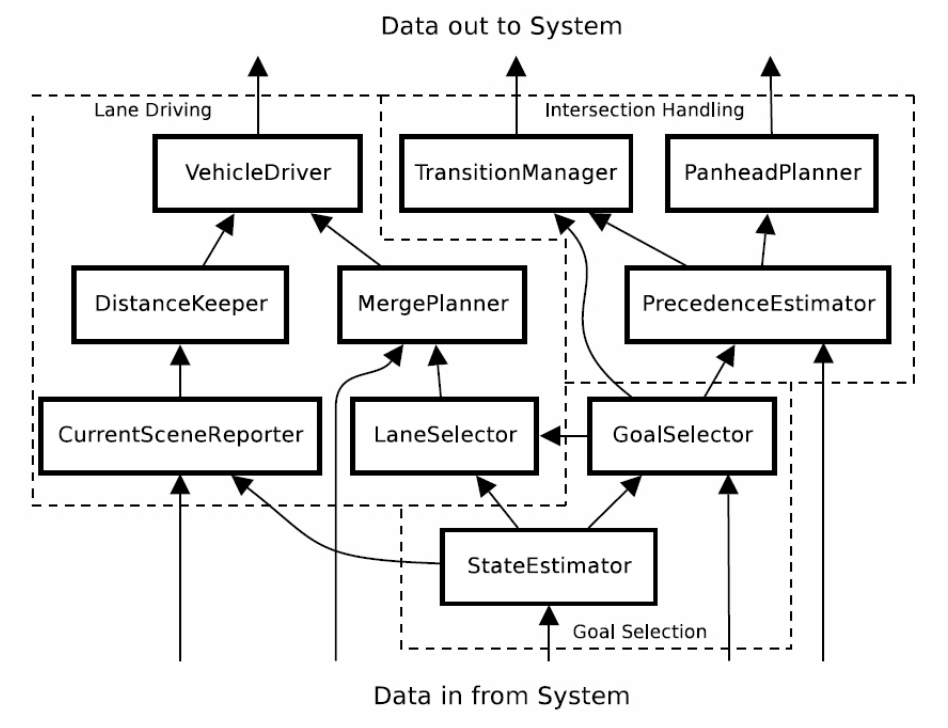


Figure 2.14: Boss' decision making architecture [20].

The "Lane Driving" maneuver integrated a traffic "lane selector", a "merge planner", a "current scene reporter", and a "vehicle driver". The "merge

planner" determined the feasibility of changing into the lane proposed by the "lane selector". The "current scene reporter" calculated the distance and velocity of the nearest obstacle, the "distance keeper" calculated the required velocity in order to keep safety distances to obstacles, and the "vehicle driver" determined so-called "motion parameters" (i.e. speed, acceleration, desired lane).

The driving maneuver "Intersection Handling" integrated three components: a "precedence estimator", a "pan-head planner" and a "transition manager". The "precedence estimator" determined the driving precedence of vehicles at an intersection, the "pan-head planner" processed data from the pan-head sensors, and the "transition manager" sent goals to the motion planner.

Boss' control software integrated multiple "error recovery" levels. Despite their name, these were however not focusing on how to improve the road safety, but instead on how to avoid getting stuck in traffic. In each higher error recovery level, the vehicle would perform higher risk driving maneuvers, for instance by ignoring traffic lanes, and eventually driving outside intersections and roads.

Although their vehicles has won the Urban Challenge, Boss' developers acknowledge that their traffic representation was not sufficient to make intelligent driving decisions compared to human drivers [20]:

> *"A richer [traffic] representation including more semantic information will enable future autonomous vehicles to behave more intelligently.*

Another problem addressed by them is the problem of proof of software correctness [20]:

> *"Our approach of generating an ad hoc, but large, set of test scenarios performed relatively well for the Urban Challenge, but as the level of reliability and robustness approaches that needed for autonomous vehicles to reach the marketplace, this testing process will likely be insufficient. The real limitation of these tests is that it is too easy to teach to the test and develop systems that*

are able to reliably complete these tests but are not robust to a varied world. To reduce this problem, we incorporated free-for-all testing in our test process, which allowed traffic to engage Boss in a variety of normal, but unscripted, ways. Although this can increase robustness, it can in no way guarantee that the system is correct.“[9]

Junior (2nd place)

The vehicle ”Junior“ (Figure 2.13.b) was developed by Stanford University. The vehicle was based on a Volkswagen Passat, and was equipped with five LIDAR sensors, a GPS/INS receiver, five Radar sensors and two Intel quad core computers [16].

Junior's control software architecture has been originally developed for the vehicle ”Stanley“ (section A.3.1), the winner of the DARPA Challenge 2005 [16]. It consisted of several modules communicating asynchronously. The main module groups were (Figure 2.15):

- Sensor Interfaces,
- Perception Modules,
- Navigation Modules,
- Drive-by-Wire Interface,
- Global Services.

Junior's decision making was based on a finite automaton (finite state machine) with 13 states. Figure 2.16 shows 11 of the 13 states, as published in [16]. The Figure omits the states ”Escape“ and ”Traffic Jam“, as ”nearly all states have transitions to them“ [16].

The finite automaton states fulfilled the following purposes:

[9]This problem is addressed in this work in Chapter 5, by specifically focusing on the ability to formally verify the correctness of the decision making module with respect to its specification.

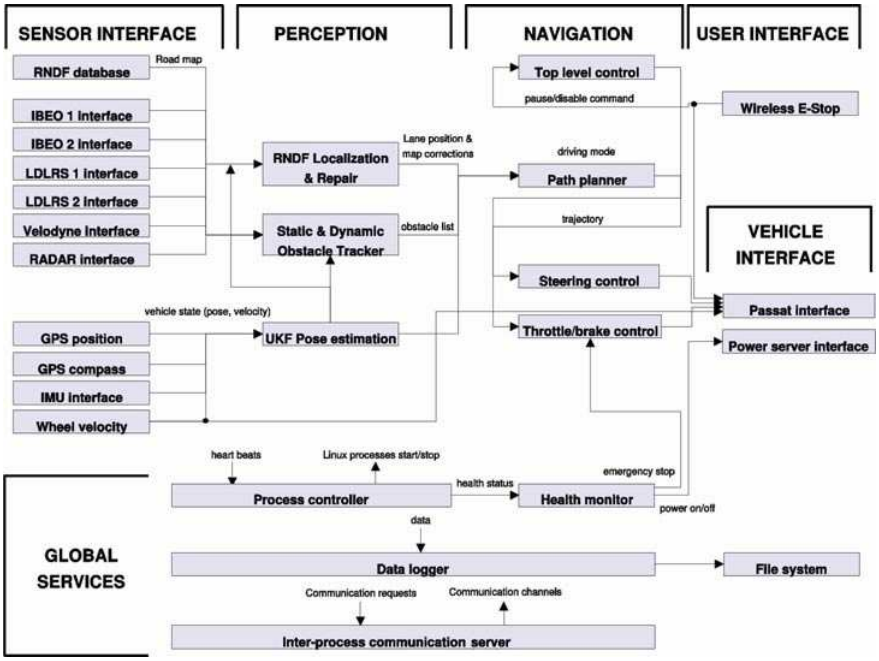


Figure 2.15: Junior's control software architecture [16].

LOCATE_VEHICLE:	initial state.
FORWARD_DRIVE:	forward driving, lane following, obstacle avoidance.
STOP_SIGN_WAIT:	waiting at stop signs.
CROSS_INTERSECTION:	intersection handling.
STOP_FOR_CHEATERS:	waiting for other cars to clear intersections.
UTURN_DRIVE:	U-turn.
UTURN_STOP:	The vehicle stops before a U-turn.
CROSS_DIVIDER:	avoid partial road blockage (crossing yellow line).
PARKING_NAVIGATE:	driving in parking areas.
TRAFFIC_JAM:	drives around a road blockage (ignoring traffic rules).
ESCAPE:	"same as TRAFFIC_JAM, more extreme" [16].
BAD_RNDF:	drives on roads without matching RNDF.
MISSION_COMPLETE:	end of race.

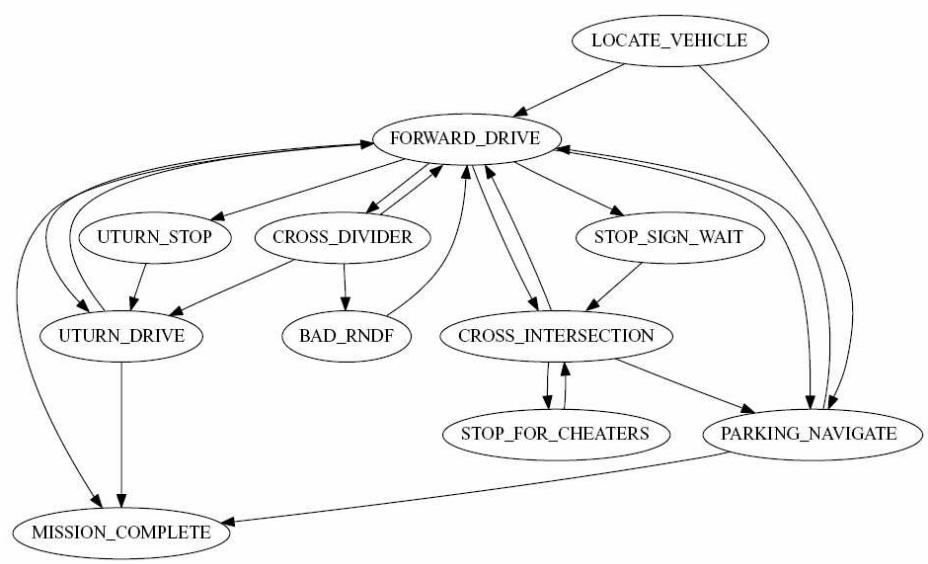


Figure 2.16: Junior's automaton for decision making [16].

Junior's developers conclude that, although their vehicle was able to complete the DARPA Urban Challenge without colliding with any obstacles while respecting traffic rules, their vehicle, and probably any other vehicle competing in this race, would probably not be able to cope with a realistic urban traffic environment. According to them, the reason is the simplified traffic environment, and the fact that DARPA officials *"frequently paused robots in the Urban Challenge to clear up traffic jams"* [16].

Odin (3rd place)

The vehicle "Odin" (Figure 2.13.c) was developed by Team "VictorTango", a cooperation between Virginia Tech, TORC, Ford, and Caterpillar. Odin was based on a Ford Hybrid Escape, which was equipped with two servers (each with two quad-core processors), a GPS/INS receiver, two IBEO Alasca XT LIDAR sensors, a Alasca A0 LIDAR sensor, two SICK LMS 291 LIDAR sensors, and two cameras. However, the cameras were not used during the race [19].

Odin's control software architecture claims to follow a "novel hybrid deliberative-reactive paradigm", where "perception, planning, and acting oc-

cur at several levels and in parallel tasks" [19]. Odin's architecture is shown in Figure 2.17.

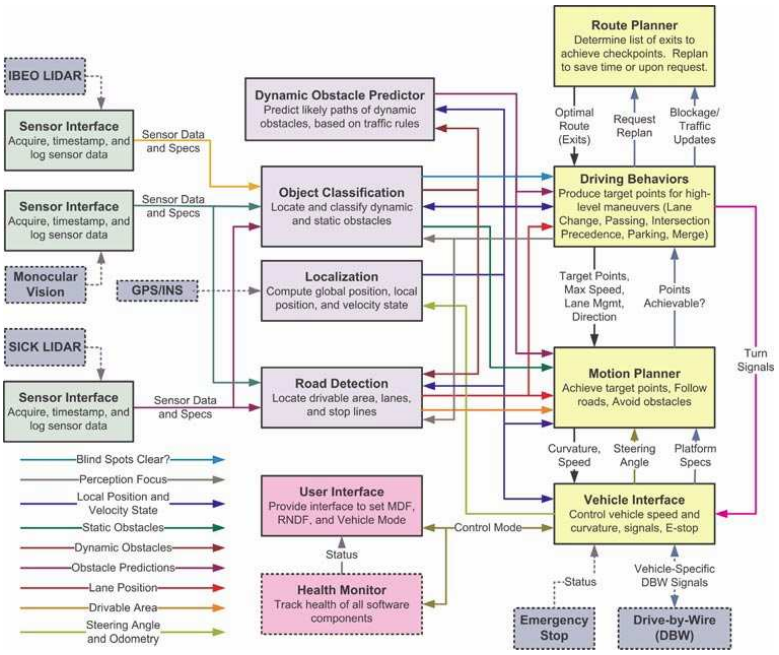


Figure 2.17: Odin's control software architecture [19].

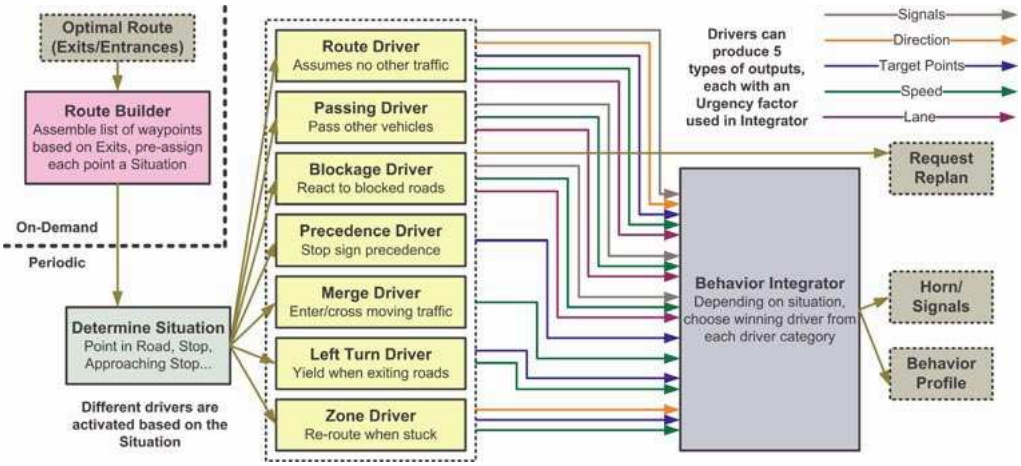


Figure 2.18: Odin's decision making [19].

Relying only on LIDAR sensor information, obstacles were classified as vehicles if they were moving, or as static obstacles otherwise. The avail-

able cameras were not used because computer vision was *"determined to be too computationally intensive to return accurate information about nearby objects"* [19].

Odin's control software was able to preform a variety of driving maneuvers, so-called "behaviors", or "drivers" (Figure 2.18). The driving maneuvers "route driver", "passing driver", and "blockage driver" were used to cope with obstacle-free driving, to pass slower or stopped vehicles, and to replan around a blocked road, respectively. The driving maneuvers "precedence driver", "merge driver", and "left turn driver" were used for crossing intersections, merging into traffic, and turning. The driving maneuver "zone driver" was used for driving on parking areas.

The decision making module (called "Behavior Integrator") was based on an "arbitration" method, using an "winner-takes-all" mechanism . A "system of hierarchical finite state machines" was used to allow driving maneuvers "distinguish between intersection, parking lot, and normal road scenario" [19]. No details have been published about the finite state machines or the arbitration mechanism.

During the final race event, the software modules for vision-based and laser-based lane detection, as well as vision-based obstacle detection, were disabled. The reason for not using them was the team's fear that the software was "not mature enough to handle all the possible cases within the scope of the Urban Challenge rules" [19]. Consequently, Odin relied only on LIDAR and GPS/INS sensors.

Talos

The vehicle "Talos" (Figure 2.13.d) was developed by Team MIT (Massachusetts Institute of Technology). The vehicle was based on a Land Rover LR3[10], and was equipped with a Quanta blade server computer system, an Applanix POS-LV 220 GPS/INS, a Velodyne HDL-64 LIDAR sensor, 12 SICK LIDAR sensors, 5 cameras, and 15 Delphi radars. The cameras were used to detect road markings, but not for obstacle detection [14].

[10] A Ford Escape was first used during the development.

Talos' control software consisted of the following 10 modules (Figure 2.19):

Road Paint Detector:	used cameras to detect road markings
Lane Tracker:	merged RNDF data with detected lanes and LIDAR data
Obstacle Detector:	used LIDARs to detect obstacles
Hazard Detector:	used LIDAR to "assess the drivability" and to detect curbs
Fast Vehicle Detector:	used Radar to detect fast approaching vehicles
Positioning:	used GPS/INS to estimate the position
Navigator:	calculated the next short-term goal
Drivability Map:	provided drivability information of planned paths
Motion Planner:	calculated path trajectories
Controller:	executed low-level control tasks.

The Navigator module integrated the following so-called "high-level behaviors":

- Shortest route to the next MDF checkpoint,
- Intersection precedence, crossing, and merging,
- Passing,
- Blockage Replanning,
- Goal generation for the motion planner,
- "Generation of fail-safe timers",
- Turn signaling.

The Blockage Replanning and so-called "Fail-Safe Modes" were mechanisms to ensure that the vehicle continues the race, although increasing the risk of collisions. Whenever the vehicle stopped, so-called "fail-safe timers" were started and increase the so-called "fail-safe modes" after predefined time intervals. In each higher mode, safety constraints were released, which in turn increased the probability that the vehicle would continue driving, however at the cost of higher risk of potential collisions. For instance, safety

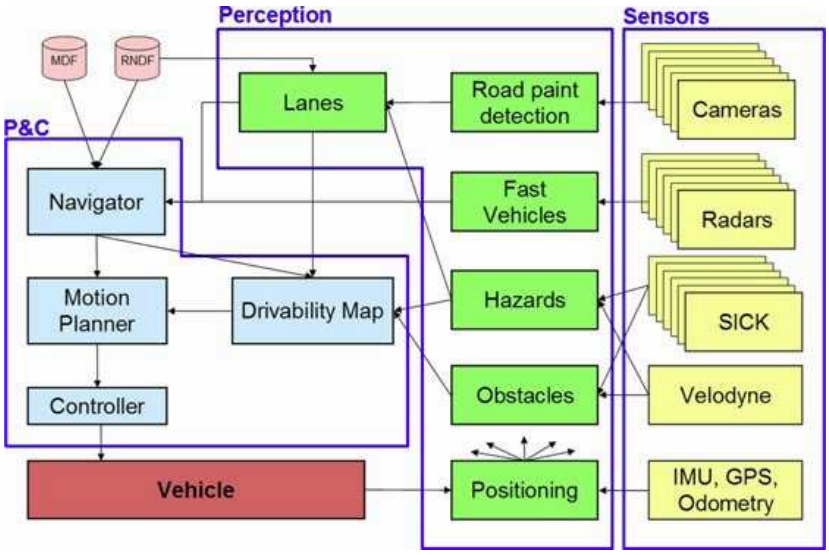


Figure 2.19: Talos' control software architecture [14].

distances around obstacles were decreased, traffic lanes were ignored, the part of the vehicle behind its rear axle was "neglected", or MDF checkpoints were skipped [14].

Talos decision making tasks were executed by the Navigator module, which continuously calculated the next destination point and moved it in front of the vehicle accordingly. Whenever Talos had to wait for other traffic (e.g. at intersections), the destination point remained unchanged. Passing was executed in a similar way, by notifying the motion planner whether the overtaking lane was free [14].

Little Ben

The vehicle "Little Ben" (Figure 2.13.e) was developed by the Team Ben Franklin, which was a cooperation between the University of Pennsylvania, Lehigh University, and Lockheed Martin Advanced Technology Laboratory. The vehicle was based on a Toyota Prius hybrid and equipped with 7 Mac Mini computers (Core 2 Duo), a GPS/INS sensor, 3 SICK LMS-291 LIDARs, 2 SICK LDLRS LIDARs, 1 Velodyne LIDAR, and 3 Hokuyo URG-04LX LIDARs [15].

Little Ben's controls software consisted of multiple modules, which communicated over messages (Figure 2.20). The modules had the following functionalities:

Driving Modules:	low-level vehicle control,
Mission Planner:	calculated a path based on the RNDF and MDF files,
Sensor Modules:	gathered sensor data,
Map Plan:	integrated sensor information,
Path Follow Module:	generated driving commands for the low-level vehicle control.

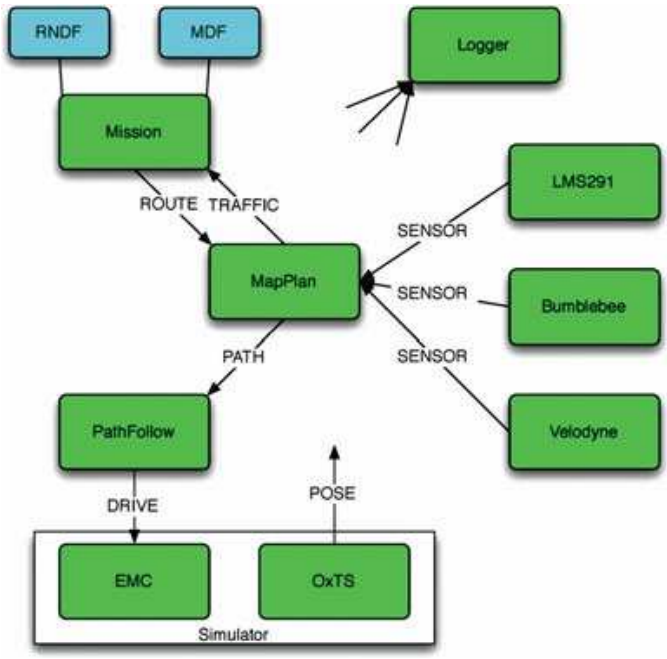


Figure 2.20: Little Ben's control software architecture [15].

Depending on the vehicle's position, Little Ben's decision making function selected between one of the following driving maneuvers, so-called specialized local planners:

- Lane Following,
- Intersection,
- U-Turn,
- Zone (parking area).

The decision about the activation of a specific driving maneuver was made based on the vehicle's current position with respect to the RNDF file and the next waypoint.

Skynet

The vehicle "Skynet" (Figure 2.13.f) was developed by the Team Skynet, which consisted of members of Cornell University. The vehicle was based on a Chevrolet Tahoe, and was equipped with 17 computers, each with a Pentium Dual-Core Mobile processor. However, only 7 of the 17 available computers were used during the race. Furthermore, Skynet was equipped with a GPS/INS receiver, 3 IBEO Alaska XT LIDARs, 2 SICK LMS-291 LIDARs, 1 SICK LMS-290 LIDAR, 1 Velodyne HDL-64E LIDAR, 8 Delphi FLR Radar sensors, and a camera [11].

Figure 2.21 shows Skynet's control software architecture. The Local Map module merged sensor information into a single map, relative to the vehicle coordinate system ("vehicle-centric"). All obstacles were treated in the same way, without distinguishing between static or dynamic obstacles. The Local Map updated and provided the list of obstacles at 10Hz. The Scene Estimator consisted of two algorithms: "posterior pose", and "track generator". The "posterior pose" combined the Local Map obstacle list with information from the RNDF file and from lane detection algorithms. The "track generator" combined obstacle information with the information from the "posterior pose", in order to track other vehicles.

Skynet's decision making module (called "Behavioral Layer") decided between four driving maneuvers (so-called "behavioral states):

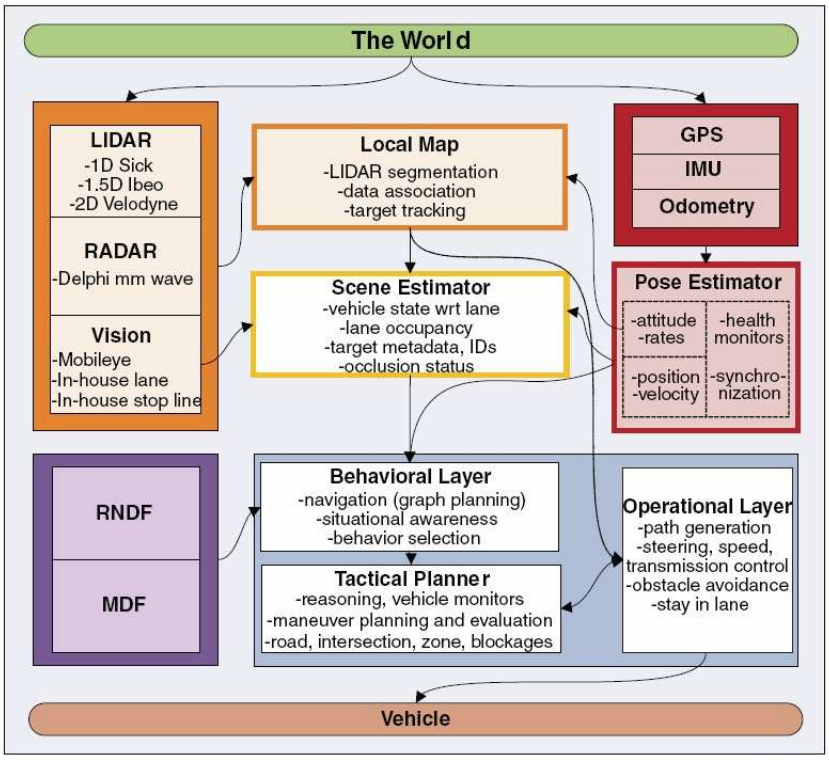


Figure 2.21: Skyent's control software architecture [11].

- Road,
- Intersection,
- Zone,
- Blockage.

The developers state that the only true decision Skynet was capable of (i.e. a choice exists between two options), was the decision regarding passing other vehicles. The passing maneuver was part of the "Road" driving maneuver, and the decision was made using a decision tree with heuristic rules. The decision tree and its heuristic rules have been developed in a simulation environment [11].

2.4 Discussion

The literature about autonomous city vehicles reveals that many questions with respect to the problem of real-time decision making, especially focusing on the high requirements of real-world, non-simplified urban traffic conditions, are yet to be addressed (Section 1.3.1). While numerous autonomous city vehicle projects have demonstrated solutions for simplified traffic conditions, according to their developers, none of the so far existing real-time decision making solutions is capable of safely coping with the complexity of non-simplified urban traffic conditions.

The demonstration of cooperative driving, which was made in a cooperation between Griffith University's Intelligent Control Systems Laboratory (ICSL) and INRIA's IMARA Laboratory in 2002, was based on a control software originally developed for autonomous mobile robots. While this demonstration showed the successful execution of cooperative driving maneuvers, which are relevant for coping with urban traffic, a crucial vehicle ability was missing, namely the ability to make driving decisions in real-time.

So far, the Southwest Research Institute did not publish any detailed information about the decision making solution developed for its currently ongoing project Southwest Safe Transport Initiative (SSTI).

VisLab's autonomous vehicle journey from Parma to Shanghai has been completed only recently, on October 28, 2010. Therefore, it is expected that more detailed information about this recent project is yet to be published.

The project "Stadtpilot" has been initiated recently, and is based on the technology initially developed for the DARPA Urban Challenge 2007. The so far published information about the vehicle's "Leonie" decision making solution has focused on the decision with respect to the vehicle's driving trajectory for a single driving maneuver, and not on the decision making with respect to the selection of the most appropriate driving maneuver, as addressed in the context of this thesis.

At the time of working on this thesis, besides a news announcement, no further details have been published about Google's autonomous vehicle project.

The DARPA Urban Challenge 2007 took place on a closed Air Force base. Although DARPA tried to create an environment similar to urban traffic by including human-driven vehicles, the traffic environment conditions were very simplified compared to real urban traffic. For example, the only traffic participants were autonomous vehicles and human-driven cars, while pedestrians, bicycles, or motorbikes were not considered. DARPA provided all traffic signs as a priori information, along with a detailed route information and navigation points.

As a result of these simplified application conditions, it was possible to successfully complete the race using LIDAR sensor information alone[11], and without recognizing and distinguishing between any obstacle types. Some teams even decided a short time before the race not to use their already fully developed computer vision systems for obstacle detection (e.g. Odin [19], Talos [14]).

Another consequence of the simplified conditions was that only three driving maneuvers, namely following a road, crossing intersections, and driving on parking areas, were sufficient to successfully complete, and even win the race.

The minimal perception abilities (i.e. LIDAR information alone), combined with the ability to perform very few driving maneuvers, resulted in very limited requirements for real-time decision making. The developed decision making solutions were focused on fulfilling the race requirements with the overall goal to win the race, and not on enabling autonomous vehicles to *safely* cope with real-world urban traffic.

Although they were successful in the Urban Challenge, some teams admitted that the solutions developed specifically for this race were not sufficient to enable safe autonomous driving in real-world urban traffic conditions. For example, Junior's development team concludes: *"Still, a number of advances are required for truly autonomous urban driving. The present robot is unable to handle traffic lights. No experiments have been performed with a more diverse set of traffic participants, such as bicycles and pedestrians. Finally, DARPA frequently paused robots in the Urban Challenge to clear up traffic*

[11] Based on 2D LIDAR data, only distances and the obstacle size is known.

jams. In real urban traffic, such interventions are not realistic. It is unclear whether the present robot (or other robots in this event!) would have acted sensibly in lasting traffic congestion." [16].

Despite winning the DARPA Urban Challenge 2007, the Tartan Racing team (vehicle "Boss") acknowledged that their traffic representation was not sufficient to make intelligent driving decisions compared to human drivers, and noted that *"A richer [traffic] representation including more semantic information will enable future autonomous vehicles to behave more intelligently* [20]. Furthermore, they conclude: *"Urban environments are considerably more complicated than what the vehicles faced in the Urban Challenge; pedestrians, traffic lights, varied weather, and dense traffic all contribute to this complexity. As the field advances to address these problems, we will be faced with secondary problems, such as, how do we test these systems and how will society accept them? Although defense needs may provide the momentum necessary to drive these promising technologies, we must work hard to ensure that our work is relevant and beneficial to a broader society. Whereas these challenges loom large, it is clear that there is a bright and non-too-distant future for autonomous vehicles."* [20].

2.4.1 Comparison of Existing Solutions for Real-Time Decision Making

Common Points

The main common point between all existing solutions for real-time decision making for autonomous city vehicles is that they were all designed and developed for very specific and narrowly defined application scenarios.

In order to be able to design, implement, and test them within the short development time frames with the limited resources, the most successful teams chose the approaches which were just sufficient for fulfilling the application requirements for driving in the simplified urban traffic conditions. Established modeling tools, such as finite automata [16, 18], hierarchically structured finite automata [17, 19], arbitration methods [19], weighted

votes [44], or simply heuristics [11] offered the benefits of easier implementation, analysis, debugging, and in-field testing.

Furthermore, since these solutions were developed for the specific, simplified traffic environments, with the objective of winning a race, they were tailored to cope with the very few required traffic situations, using the available technology. For example, the DARPA Urban Challenge vehicles had a similar number of driving maneuvers, and their decision making abilities were limited to dealing with road following, intersection crossing, and parking areas. Since pedestrian detection was not a race requirement, none of the vehicles had the ability to react to them accordingly. Nevertheless, the developers claim that their existing approaches are extendable to include new decision situations and driving maneuvers.

A more recent overview on the common points of the 2007 DARPA Urban Challenge vehicles can be found in [45].

Points of Difference

Since all competing DARPA Urban Challenge vehicles were developed for identical scenarios, under identical requirements, with similar resources and technology, the points of difference with respect to decision making are relatively few. They mostly consist in different modeling approaches and algorithms. While some teams chose finite automata [16], others chose heuristics [11], or even activated driving maneuvers according to the vehicle's road environment[12] (e.g. intersection, road following, parking area) [20].

Unaddressed Issues

Although the DARPA Urban Challenge vehicles were successful in coping with the simplified urban traffic conditions, there are many unaddressed issues with respect to decision making, before such vehicles will be able to cope with real-world, non-simplified urban traffic.

While the number of required driving maneuvers was relatively low for the

[12] Although not stated by the developers, this approach can also be regarded as a finite automaton.

simplified DARPA Urban Challenge traffic conditions, real-world conditions require a significantly higher number of driving maneuvers, with variable complexity. Therefore, since the decision making subsystem makes driving decisions by switching between the execution of driving maneuvers, one of the questions is how to design and model versatile driving maneuver algorithms with varying complexities in a common structure[13] in a consistent way.

Furthermore, the question of how to integrate a significantly larger number of information sources (e.g. on-board sensors, communication), each bringing important information regarding the vehicle's environment, has not been addressed[14].

Another relevant, but unaddressed issue is how to model and implement a highly complex operational behavior of the decision making subsystem, which needs to incorporate not only a large number of traffic rules, but also driving objectives, which lead to safe driving, and ideally reflect driving decisions of human drivers.

Autonomous city vehicles for civilian, non-military applications are not likely to gain public acceptance unless they prove to be safer than conventional human-driven vehicles. Therefore, the decision making subsystem plays a crucial role toward reaching this goal. However, the so far published material about autonomous city vehicles did not reveal any evidence that the problem of real-time decision making for such vehicles has already been addressed with respect to ensuring road safety, and focusing on a generic solution for real-world (non-simplified) urban traffic conditions.

The remainder of this thesis elaborates on, and presents a solution for this crucial and so far hardly addressed problem of real-time decision making for autonomous vehicles operating in real-world, non-simplified urban traffic conditions.

The following Chapter 3 explains the research question, its scope within the autonomous vehicle's decision making & control system, and the research objectives addressed in this thesis.

[13]These questions are addressed in Chapter 6.
[14]These questions are addressed in Chapter 4.

Chapter 3

The Research Question

3.1 Introduction

As already mentioned in the introduction chapter, the problem of decision making depends, in a technical way, on solutions and results from a variety of other research topics, such as sensor technology, perception, localization, path planning, and vehicle control (Figure 3.1). At the current time, the state of research and technical developments of many of these topics on which decision making depends, have not reached a sufficiently advanced level which would be required to build and test autonomous vehicles in real-world urban traffic conditions.

Although aware of the limitations imposed by currently available technology, which however are likely to be overcome through research and technical advances in the near future, the undertaken research project and this resulting thesis are aimed at developing a generic solution from the viewpoint of design requirements for enabling autonomous vehicles to safely cope with real-world urban traffic conditions.

The solution presented in this thesis does not build upon the currently existing and available technology in order to develop a today technically fully implementable solution, which however might not entirely fulfill requirements. Instead, the main focus is on the more ambitious objective of coping with real-world, non-simplified urban traffic conditions, while taking

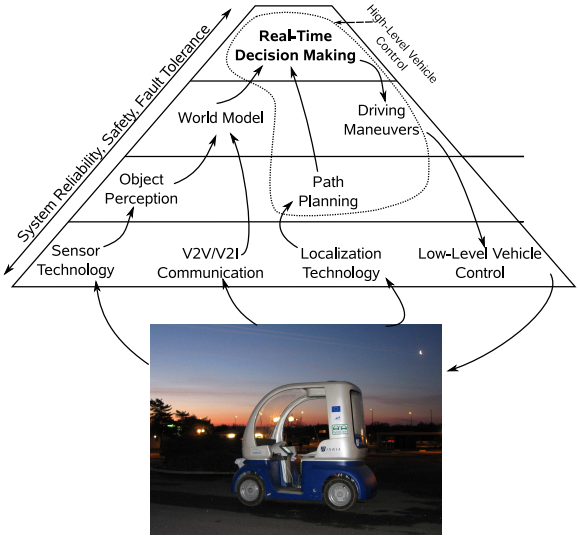


Figure 3.1: Research areas for autonomous city vehicles (repetition of Fig. 1.1).

into account the limitations of the current technology.

While this approach brings the apparent drawback that actually building such a system might not be technically possible at the current time, it leads to a solution which will fully fulfill its objectives, and which will be implementable in the near future.

For example, although the currently available perception sensors (e.g. pedestrian detection) are relatively advanced, they still do not offer the required accuracy and reliability for safe driving in urban traffic [12,20]. Nevertheless, since research efforts are constantly focusing on improving available, and developing new sensor technologies, it is only a matter of time until the current technical limitations will be overcome. Therefore, due to continuous technological advancements, it is important not to restrict the design on the currently available technology, but instead to focus on entirely fulfilling the requirements[1].

[1]This approach has been motivated by, and is in line with Stadler's remark: *"There are two overall limitations on every design process: the limitation imposed by natural law, reflected in the postulates of the relevant theories, and the limitations imposed by the available technology required to manufacture a given design. All too often the latter is taken as the critical limitation, when instead we should first find the best design possible within an axiomatic structure, and then strive to develop the technology capable of achieving it."*

The following subsections explain the decision making & control system architecture and the research objectives addressed by this thesis.

3.2 Autonomous Vehicle Decision Making & Control System Architecture

The autonomous vehicle decision making & control system consists of the following functional subsystems (Figure 3.2):

- Perception Subsystem,
- Real-Time Decision Making & Driving Maneuver Control,
- Driving Maneuvers,
- Low-Level Vehicle Control.

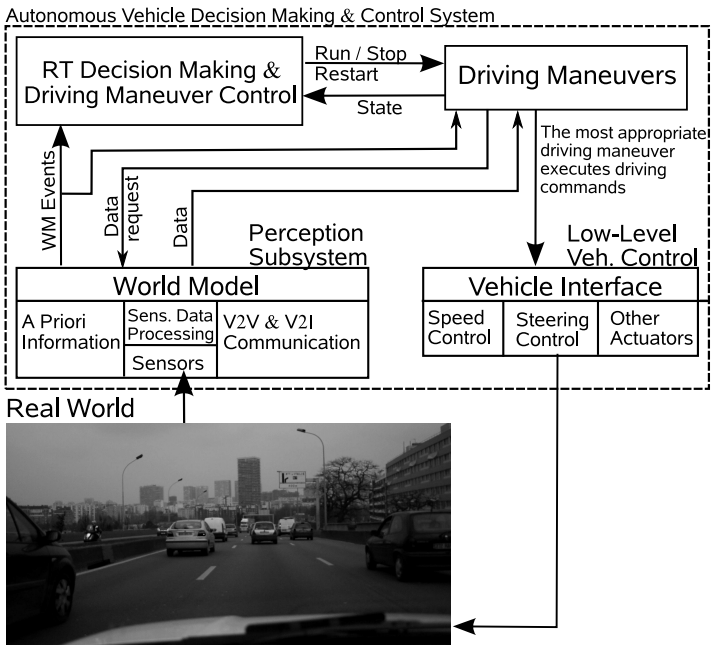


Figure 3.2: Simplified view of the autonomous vehicle decision making & control software architecture and the flow of data [47].

(Wolfram Stadler [46], p.355).

The purpose of the Perception Subsystem is to collect available information about the vehicle's road traffic environment, to manage and process it, and to provide it in a adequate form to the Real-Time Decision Making & Driving Maneuver

Based on the information provided by the Perception Subsystem, the Real-Time Decision Making & Driving Maneuver Control subsystem makes driving decisions. This software subsystem decides about the activation and the execution of the most appropriate driving maneuver for any given traffic situation.

The Driving Maneuvers subsystem contains a set of closed-loop control algorithms, each able to maneuver the vehicle in a specific traffic situation. The driving maneuvers direct their output to the Low-Level Vehicle Control subsystem.

The Low-Level Vehicle Control subsystem contains hardware and software components, which control the vehicle's speed, steering angle, and other actuators (e.g. transmission).

3.3 Research Objectives

While addressing the problem of real-time decision making, the questions regarding information input and information output emerges. Regarding the information input, the design and development of a World Model is required, which provides the available information about the vehicle's traffic environment in order to be able to make appropriate driving decisions.

Regarding the information input, i.e. the World Model, the following issues need to be addressed:

- A World Model data structure which is suitable to store both the information about relevant entities, but also their relationships to each other,

- The management of information stored in the World Model (i.e. inserting, deleting, maintaining up-to-date and consistent data),

- An application programming interface to the World Model, which enables the exchange of information between the World Model and the Real-Time Decision Making subsystem.

Regarding the information output of the decision making process, the following aspects need to be addressed:

- The decision making process needs to start and stop the execution, and retrieve information about the status of driving maneuvers. Therefore, driving maneuvers require a common structure.
- Driving maneuvers with different complexity need to be implemented. For example from very simple ones, such as Stop&Go to complex maneuvers, such as overtaking or intersection crossing.
- The execution of driving maneuvers needs to be in parallel (or concurrent) to other relevant processes.

Consequently, this thesis elaborates on, and presents the developed solutions, for the following problems:

1. Development of a World Model, which is the input for the Real-Time Decision Making subsystem.
2. Development of a Real-Time Decision Making algorithm.
3. Development of a generic Driving Maneuver concept, which can be used for the purpose of modeling and implementing the autonomous vehicle's maneuvers.
4. Integration and testing of the entire solution for real-time decision making by driverless city vehicles in both simulation and real world road traffic conditions.

The following Chapter 4 addresses and explains the World Model.

Chapter 4

The World Model

4.1 Introduction[1]

Prior to developing a solution to solve the problem of Real-Time Decision Making by driveless city vehicles, and thus enabling them to make appropriate driving decisions in urban traffic, the question emerges regarding the required input information about the vehicle's surrounding environment.

Similar to a human driver, the Real-Time Decision Making subsystem of an autonomous city vehicle requires a large variety of information about the surrounding traffic environment. In addition to information known before the vehicle begins its journey, sensor components perceive specific aspects of urban traffic, such as traffic lanes, traffic signs, obstacles, etc., while the vehicle is driving. Additionally, information may be provided through communication with other vehicles (vehicle-to-vehicle, V2V), and/or with the infrastructure (vehicle-to-infrastructure, V2I).

Although numerous solutions have been developed for modeling the environment of autonomous robots [49] and/or off-road autonomous vehicles [26, 50], no solution or proposal has yet been published, which is capable of modeling all relevant aspects of urban roads, which are required for autonomous city vehicles safely operating in non-simplified urban traffic conditions. Such aspects are for instance perceived and classified objects, such as

[1]The concept of this chapter has been accepted for publication in [48].

vehicles, static obstacles, pedestrians, roads, intersections, traffic lanes, and traffic signs, but also the relationships between these objects (e.g. obstacle on left lane, intersection connecting roads, traffic sign valid for one lane, etc.).

A closely related approach for autonomous city vehicles has been proposed by Benenson et al. [51]. However, it limits the modeled traffic features to static and dynamic obstacles, without including additional perceived traffic information such as traffic lanes, intersections, or traffic signs. Other related research areas with similar requirements for world model information are for instance driver assistance systems, such as the project Safespot [52].

The modeling solutions developed for the DARPA Urban Challenge 2007 were focused on simplified requirements, and were therefore not sufficient for enabling autonomous vehicles to operate in public urban traffic. For example, the developers of the DARPA Urban Challenge winning vehicle "Boss" observed that their traffic representation was not sufficient to make intelligent driving decisions compared to human drivers [20]. Furthermore, as already mentioned, Junior's developers (2nd place) noticed that their vehicle, and probably any other vehicle competing in this race would not be able to cope with a realistic city traffic environment [16].

Figure 3.2 shows that the World Model, a specifically developed software component, fulfills the purpose of:

a) collecting information from perception and communication subsystems;

b) maintaining an up-to-date view of the vehicle's environment, and;

c) providing the required input information to the Real-Time Decision Making subsystem in a well-defined, structured way.

While sensor and perception related issues, such as sensor noise, uncertainties, sensor reliability, etc., are important research topics, they are not further addressed in this thesis, as they should be addressed in the scope of the research and development of sensor and perception subsystems. Here, we assume that the perception information is correct and reliable.

4.2 Requirements

The overall objective of the autonomous vehicle decision making & control software is to obtain a control functionality which is in line with that of a human driver, while achieving superior road safety. For this purpose, different types of input information about the traffic environment are required (Figure 4.1):

- a priori information,
- information obtained from on-board sensors in real-time during the vehicle's movement,
- information obtained through communication.

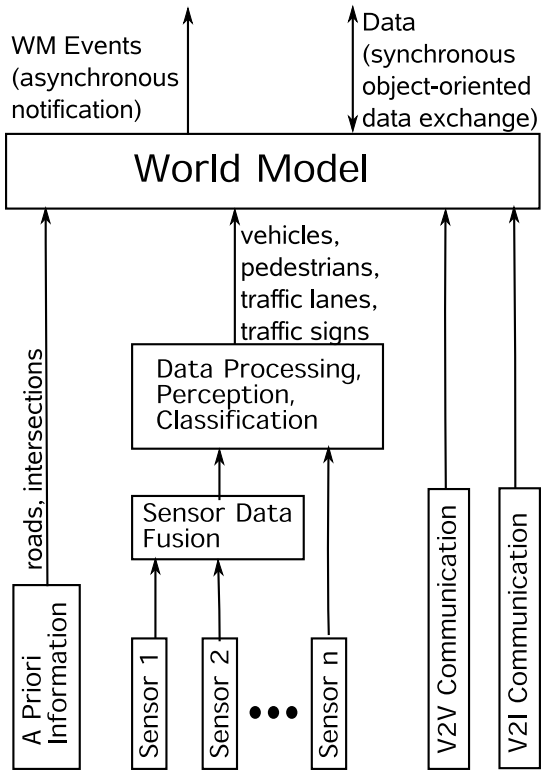


Figure 4.1: World Model information input and output.

The a priori information includes all information which is available in advance, before the autonomous vehicle starts its journey. This includes for

example a planned travel path, coordinates of intersections, and/or other relevant information about the road infrastructure, such as the number of traffic lanes. The minimum information which needs to be given a priori is the list of roads to follow, and the list of GPS waypoints.

The a priori information is specified for the planned route in a similar way as in the DARPA Urban Challenge 2007 (i.e. Route Network Definition File RNDF) [27, 53], however using a structured XML file instead.

In addition to the a priori information, the autonomous vehicle continuously obtains real-time information about its traffic environment. This information is obtained either from the vehicle's own on-board sensors (e.g. cameras, LIDAR, RADAR), through Vehicle-to-Vehicle (V2V) communication, or through Vehicle-to-Infrastructure (V2I) communication (e.g. a traffic management centre), or both [54]. Components for sensor data fusion, processing, perception and classification deal with sensor-related problems such as sensor noise, and uncertainties. Therefore, it is assumed that the information coming from these components is reliable, correct, and consistent with the real-world environment.

The purpose of the World Model is to merge this information and to provide at any given time an accurate and up-to-date representation of the autonomous vehicle's traffic environment, which is used as input information by the real-time decision making subsystem.

4.2.1 Functional Requirements

The following are the World Model's most relevant functional requirements (Figure 4.1):

- Stores a priori information, such as
  - Roads:
    - List of coordinates
    - Speed limit
    - Type: one-way, two-way
    - Number of ongoing lanes
    - Number of oncoming lanes
  - Intersections:

        * Coordinates (global)
        * Crossing Roads
        * Priority Road
    - Traffic Signs:
        * Sign Type
        * Coordinates (global)

- Stores information provided by the Sensor Components, such as
    - Obstacles:
        * Type (e.g. vehicle, pedestrian, static)
        * coordinates (global and/or vehicle coordinates)
        * velocity
        * direction of travel
    - Traffic Lanes:
        * Coordinates of left/right boundary
        * Lane width
        * Direction (e.g. turns left/right)
        * Speed limit
    - Traffic Signs (perceived):
        * Sign Type
        * Coordinates (global)
- Stores information provided by the Communication component, such as:
    - Communicating Vehicles:
        * Vehicle ID
        * coordinates (global and/or vehicle coordinates)
        * velocity
        * direction of travel
        * driving intention (overtake, stop)
    - Traffic Management Centre:
        * Road speed limit
        * Blocked roads
- Cyclicly merges and updates the a priori information with the information obtained continuously from Sensor and Communication components.
- Calculates cyclicly the relationships between the stored entities. For example, if sufficient data is available, the World Model determines the positions of:
    - the current road,
    - the current traffic lane,

  - obstacles on the current road,
  - type of obstacles on the current lane/road,
  - distances to obstacles.

- Notifies other subsystems of relevant events in the traffic environment through an asynchronous mechanism.
- Provides other subsystems complete access to all stored information by replying to synchronous data requests.

4.2.2 Non Functional Requirements

Besides the typical non functional requirements which are relevant for the whole autonomous vehicle decision making & control system, such as reliability[2] and robustness[3], specific non functional requirements for the World Model are:

- Real-Time performance: all World Model information needs to be stored, processed, and provided to other subsystems, more precisely to the Real-Time Decision Making subsystem, within defined time intervals. The real-time requirements however are not constant, but may vary depending on the autonomous vehicle's speed.

- Interoperability with various types of sensor components or perception subsystems: In order to be able to integrate a variety of perceiving sensor components and their interoperability, the World Model should not depend on specific hardware and/or software components, such as a specific camera system for the recognition of obstacles. Instead, such hardware specific components are integrated in lower component levels, and are not part of the World Model.

- Modularity: Sensor components provide information about specific perceived entities for which they were designed, such as a perceived vehicles, pedestrians, traffic lanes, or traffic signs. Consequently, a modu-

[2]Reliability with respect to software refers to the probability that the computer program operates error-free in a specified environment for a specified period of time [55].

[3]Software robustness refers to the ability of a computer program to perform well not only under ordinary, but also under unusual conditions.

lar decomposition into objects representing such entities, is suitable for storing this information in a structured way.

- Portability to other autonomous vehicle decision making & control system architectures: The information requirements about the vehicle's traffic environment are generic for all autonomous city vehicles. Consequently, due to a clearly defined API of the World Model, the whole Perception Subsystem can be integrated into other modular decision making & control system architectures, if necessary.

- Ease of integration through flexible information exchange: The World Model should actively notify other subsystems about events of relevance happening in the real world. This is realized using the World Model Events API.

- Reusability: Another application for Perception Subsystem and its World Model is for instance its integration into driver assistance systems. Such systems notify the driver of potential dangers, and require therefore the same information about the vehicle's traffic environment as the decision making subsystem of an autonomous vehicle.

The modular decomposition of the control system architecture into the five functional subsystems (Figure 3.2) is common in many current control system architectures for autonomous city vehicles, such as those developed for the DARPA Urban Challenge [14, 16, 18]. Therefore, the decoupling of the World Model from other control system functionalities as a monolithic subsystem enables its integration and reusability in a variety of autonomous vehicle control architectures. Nevertheless, in order to ensure its usability and real-time execution on dedicated hardware, the internal architecture of the World Model needs to be structured in a modular way, and provide a well-defined interface to its functionality.

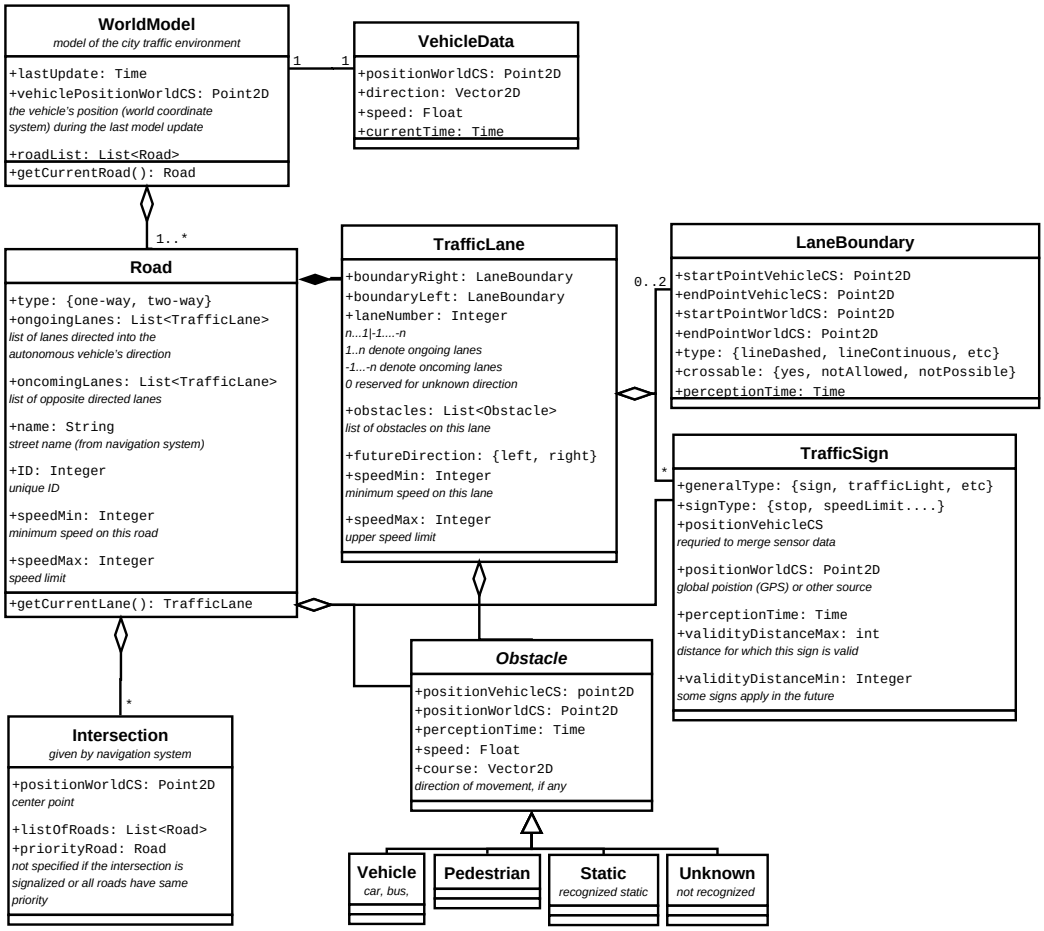


Figure 4.2: World Model UML [56] class diagram (simplified overview).

4.3 The World Model Software Architecture

Due to the high complexity, a highly detailed model of the urban traffic environment is not adequate, as it would exceed an autonomous vehicle's limited real-time computing capabilities. Therefore, the modeled entities are restricted to a minimum set considered to be necessary for autonomous driving, while maintaining an extendable object-oriented design for possible future needs.

The modeled entities are those which need to be stored as specified in the list of functional requirements (subsection 4.2.1): roads, traffic lanes (including one or two lane boundaries), intersections, obstacles (vehicles,

pedestrians, static obstacles, obstacles of unknown type), and traffic signs[4].

A generic and extendable software representation of the traffic environment is defined by decomposing the traffic features, such as roads, traffic lanes and traffic signs, into several entities and specifying their relationships to each other. In the World Model data structure, each entity is represented by a class.

Figure 4.2 shows a simplified[5] UML class diagram of the software component "World Model". The main class of this software component is called "WorldModel". This class is associated with the class "VehicleData", which contains all vehicle related data, such as the vehicle's current position, steering angle, speed and the current time.

The class "WorldModel" is an aggregation of roads (class "Road"); the "WorldModel" contains at least one instance of the class "Road". It can be assumed that at least the current road is always known during the vehicle's movement, using localization systems such as GPS and INS (Inertial Navigation System).

A road is specified by its type (e.g. one-way, two-way), name, ID, and speed limits. Each road contains two lists of references to traffic lanes: ongoing lanes and oncoming lanes.

A traffic lane (class "TrafficLane") is defined by its left and right lane boundaries (class "LaneBoundary"), a lane number which also encodes the travel direction relative to the calculated travel route, a list of known obstacles, the future direction of the lane and speed limits. A traffic lane is only defined as part of a road. Each traffic lane contains a maximum of two lane boundaries.

A lane boundary (class "LaneBoundary") is defined by a start point and an end point, which are coordinates of detected lane boundary segments. Furthermore, a lane boundary also contains a type and an attribute, which specifies whether the vehicle is able or allowed to cross the lane (*crossable*).

[4]The current concept only includes entities on the road, and neglects entities outside of the road environment (e.g. pedestrians on sidewalk). However, the modular architecture allows to extend the World Model to include such entities, if required.

[5]For the sake of readability, the UML diagram is simplified. Attributes are defined *public* and only a few of the required operations are specified.

Traffic signs (class "TrafficSign") can belong to a traffic lane, or, if they are valid for the whole road, to the road. A traffic sign is described by a general type (e.g. road marking, sign), the traffic sign type (e.g. stop, give way), its position and distance of validity.

The abstract class "Obstacle" encapsulates common properties of all obstacles, such as position, speed and moving course. Subclasses of the class "Obstacle" represent recognized obstacles, such as vehicles, pedestrians, etc. Similar to traffic signs, obstacles can belong to traffic lanes or to roads. The class "Static" describes static obstacles (e.g. tree), while the class "Unknown" is used for obstacles which were detected but could not be further classified.

Intersections (class "Intersection") are specified by their positions, a list of roads belonging to the intersection and, if known, the priority road. On the other hand, each road contains references to its intersections. The referencing between the classes "Intersection" and "Road" enables both the extraction of an intersection's roads and the calculation of the intersection sequence on a road.

The main objective of the World Model data structure is to include all information about the perceived traffic environment, and make it accessible in an intuitive way. This in turn simplifies the software development process and reduces the risk of programming errors. Another benefit is the possibility to log in real-time and later analyze what the vehicle perceived at any given time.

Example 4.3.1. In this simplified example it is assumed that the following information is available from various sensors or through communication, and is to be stored in the World Model, in order to enable safe overtaking (Figure 4.3):

- The autonomous vehicle's current (ongoing) traffic lane (e.g. using on-board cameras, GPS),
- Position, speed, and type of the detected (slower) vehicle (B) in front (e.g. using LIDAR, RADAR sensors, and cameras),

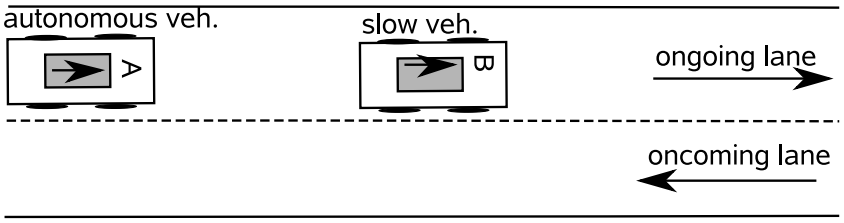


Figure 4.3: The overtaking maneuver and required World Model information.

- Oncoming traffic lane (e.g. using cameras, or communication),
- Information whether the oncoming lane is free of any other obstacles, such as oncoming vehicles (e.g. using cameras, LIDAR sensors, and communication).

The information about the autonomous vehicle's traffic environment shown in Figure 4.3 is stored in the World Model data structure as listed in pseudocode in Algorithm 4.1.

Example 4.3.2. For crossing an intersection, this example assumes that the following information is available either from a variety of on-board sensors, or is obtained through communication with other vehicles or the infrastructure (Figure 4.4):

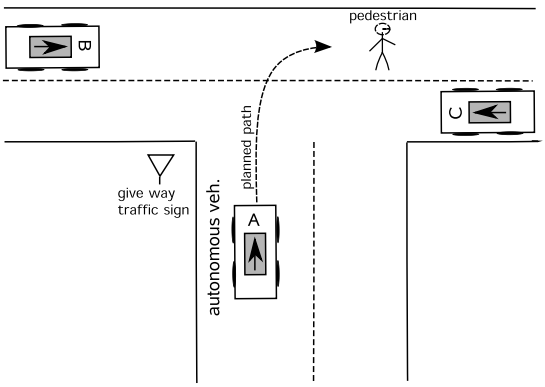


Figure 4.4: The intersection crossing maneuver and required World Model information.

- The autonomous vehicle's current road (e.g. a priori known, using GPS and cameras),

Algorithm 4.1 storeOvertakingInformation

Require: current traffic lane
Require: position, speed, type of detected vehicle in front
Require: position of oncoming lane markings
Require: list of obstacles on oncoming lane

```
1: {within World Model updating process}
2: current_road = new instance of class Road
3: current_lane = new instance of class TrafficLane
4: current_road.current_lane ← perceived current_lane
5:
6: vehicleB = new instance of class Vehicle
7: vehicleB.position ← position of vehicle B
8: vehicleB.speed ← speed of vehicle B
9: {insert vehicleB into current lane}
10: current_road.current_lane.insert(vehicleB)
11:
12: current_road.oncoming_lane ← perceived oncoming_lane
13:
14: {oncoming lane is obstacle free}
15: current_road.oncoming_lane.clearObstacles().
```

- Position of the intersection (e.g. a priori known, or through communication),
- Position of the Give Way traffic sign (e.g. a priori known, using cameras, or communication),
- Position of the connecting road along the planned travel path (e.g. a priori known),
- Position, speed, and type of all obstacles (i.e. vehicles B and C, and the pedestrian) (e.g. using LIDAR and RADAR sensors, and cameras).

The information about the autonomous vehicle's traffic environment shown in Figure 4.4, which is required to safely cross the intersection, is stored in the World Model data structure as listed in pseudocode in Algorithm 4.2.

Since the World Model is used as input for real-time decision making, its consistency and accuracy with the real world must be assured at any time during the vehicle's journey. The following subsection explains how these issues are addressed.

Algorithm 4.2 storeIntersectionCrossingInformation

Require: current road
Require: position of intersection
Require: position of Give Way sign
Require: road along planned travel path
Require: position of all obstacles within intersection area

```
 1: {within World Model updating process}
 2: current_road = new instance of class Road
 3: intersection = new instance of class Intersection
 4: traffic_sign = new instance of TrafficSign(GiveWay)
 5: intersection.insert(traffic_sign)
 6: current_road.insert(intersection)
 7:
 8: connecting_road = new instance of Road
 9: connecting_road.insert(intersection)
10:
11: vehicleB = new instance of class Vehicle
12: vehicleB.position ← position of vehicle B
13: vehicleB.speed ← speed of vehicle B
14:
15: vehicleC = new instance of class Vehicle
16: vehicleC.position ← position of vehicle C
17: vehicleC.speed ← speed of vehicle C
18:
19: pedestrian = new instance of class Pedestrian
20: pedestrian.position ← position of pedestrian
21: pedestrian.speed ← speed of pedestrian
22:
23: connecting_road.insert(vehicleB)
24: connecting_road.insert(vehicleC)
25: connecting_road.insert(pedestrian)
```

4.4 The World Model Operational Dynamics

Prior to the autonomous vehicle's journey, the a priori information (which includes positions of intersections, roads, etc.) is loaded from an XML file. During its journey, the World Model operates based on two different mechanisms (Figure 4.5):

- A cyclic updating mechanism is used for the acquisition of information from sensors and communication, as well as for the processing and updating of information about the traffic environment. This information is kept in the World Model's data structure.
- An event-based mechanism is used to notify other subsystems, such as the Real-Time Decision Making subsystem about asynchronous events, i.e. relevant changes of the traffic conditions.

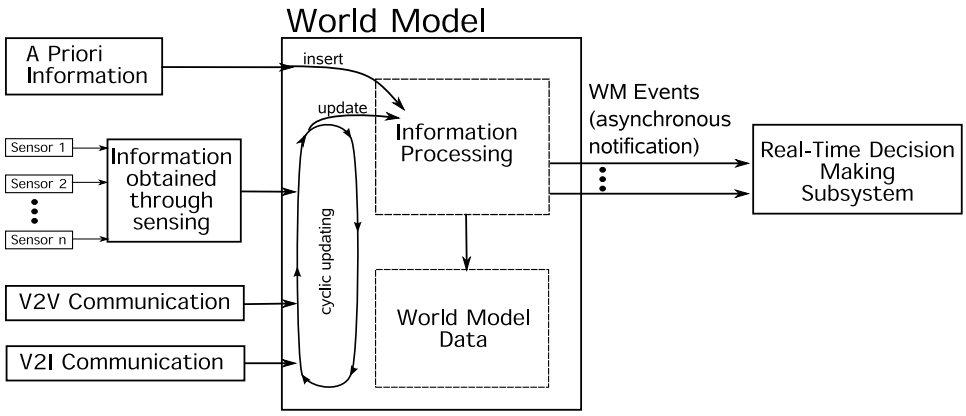


Figure 4.5: Overview on the World Model's operational dynamics.

The cyclic updating mechanism is applied in line with the sensor data sampling theory (Advanced PID Control [57], page 413). The inputs include information obtained from sensors, as well as information obtained through communication (Figure 4.5). Since the prototype implementation is not interrupt-driven, a cyclic mechanism is required for the acquisition of input signals. In each World Model updating cycle, new sensor data is acquired, processed, and the World Model information about the vehicle's road traffic environment is updated accordingly.

The event-based mechanism is used to actively notify other subsystems, such as the Real-Time Decision Making subsystem, when relevant changes in the road traffic environment are detected. Therefore, the World Model's operational dynamics is decoupled from other subsystems, eliminating their need to operate at the same execution rate as the World Model.

The remainder of this chapter explains the operational dynamics, and is structured as follows. Subsection 4.4.1 describes the file format for the a priori information, which is loaded from an XML (Extensible Markup Language) file before the vehicle begins its journey, and Subsection 4.4.2 elaborates on the World Model cyclic updating operation.

4.4.1 A Priori Information

Prior to the autonomous vehicle's journey, the predefined a priori information is loaded from an XML [58] file into the World Model structure. For the sake of brevity, only the XML file structure is addressed in this section.

Figure 4.6 shows a short example of an XML file structure containing World Model a priori information. The root element in the example is called "WorldModel" and contains the attribute "Name". The first child element of "WorldModel" is "Road", which in turn contains the attributes "ID", "Name", "PointsListFileName". The remaining elements are structured similarly.

The XML file structure which contains the a priori information (Figure 4.6) reflects the World Model's object-oriented data structure (Figure 4.2). Each XML element corresponds to a class, and each XML attribute corresponds to a class attribute. The relationships aggregation and composition between classes correspond in the XML structure to child elements.

Consequently, the a priori information can be loaded from a structured XML file in to the World Model data structure, while in the opposite direction, the World Model data structure can be logged as an XML file. As the XML file is readable as text, this is relevant for analysis, testing and debugging purposes.

Assuming that the perception subsystem is able to provide information

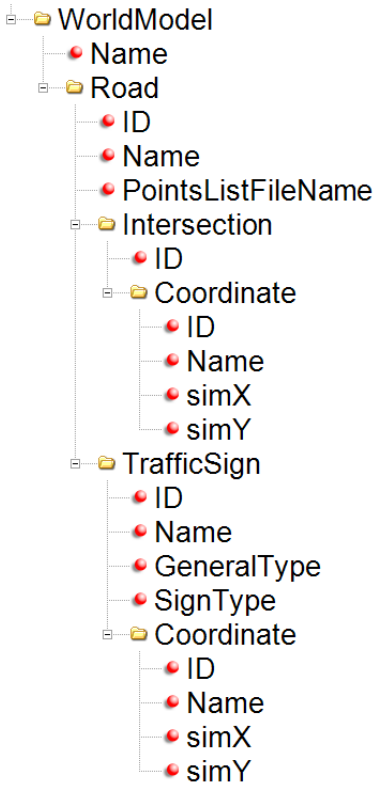


Figure 4.6: Example of an XML file structure defining a priori information. XML elements are denoted with a folder sign (e.g. Road), the dots represent attributes (e.g. ID, Name).

about all other traffic features relevant for driving, such as traffic signs, intersections, etc., the minimum information which needs to be stored in this file is the list of roads and list of GPS waypoints for the vehicle's planned travel route.

4.4.2 Cyclic Updating

On its journey, the autonomous vehicle continuously senses the traffic environment, and updates the World Model data structure with perceived information, such as detected obstacles, detected traffic signs, detected traffic lanes, etc. Accordingly, the World Model data structure needs to be cyclicly updated. The World Model updating operation can be divided into the following three steps (Figure 4.7):

1. adding newly perceived information,
2. updating existing information,
3. removing obsolete information.

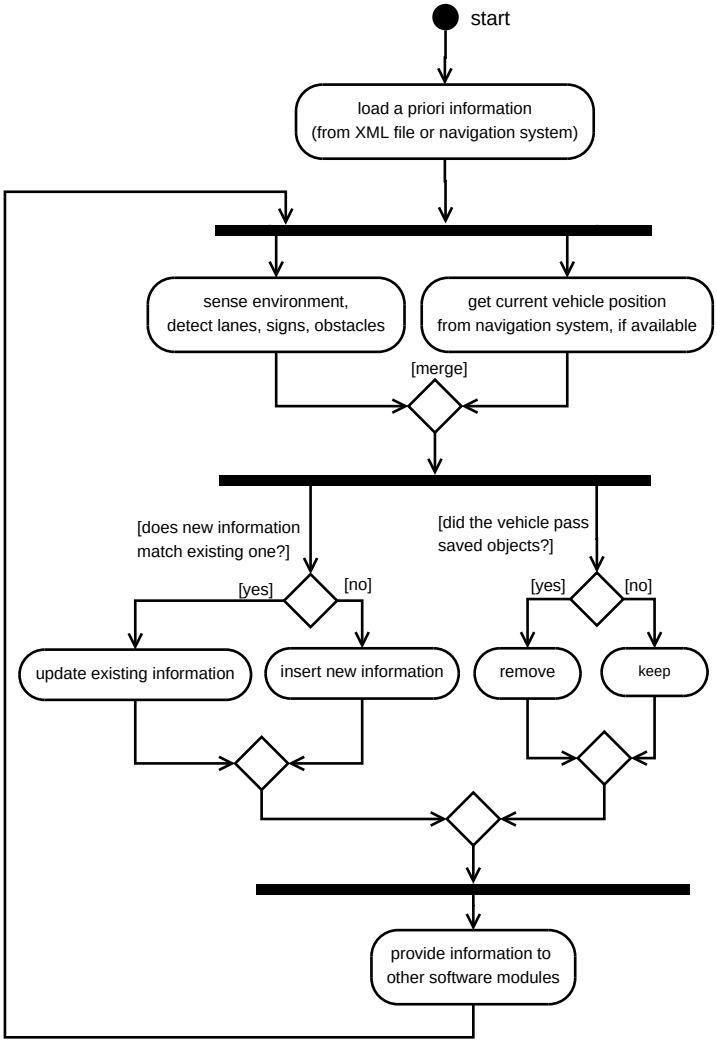


Figure 4.7: World Model UML activity diagram (simplified).

In the first step, newly perceived information, which is provided by various detection algorithms (e.g. obstacle detection) is added to the World Model data structure. This work assumes that such algorithms are already available and cover the required steps to provide accurate and consistent information

(i.e. they include solutions for conflicting or uncertain sensor data). It is assumed that any information which is obtained by the World Model from perception and communication subsystems is correct and accurate to a sufficient degree, which allows it to be used for decision making in this safety-critical system. Nevertheless, this World Model step may include the use of multiple redundant detection algorithms in order to improve the reliability of perceived information.

The second step, the updating of existing information, is required when perceived information matches an already existing one. In this case, only the perceived objects' positions in the World Model need to be updated.

In order to reduce the amount of data stored in the World Model, obsolete information, such as static obstacles which have already been passed by the vehicle, can be removed cyclicly (step 3).

4.5 The World Model Application Programming Interface

This subsection gives an overview of the World Model Application Programming Interface (API). The interface is divided into two entities, each implementing a different concept:

- World Model Events
- Object-oriented data structure.

4.5.1 World Model Events

A World Model Event represents a discrete event which signals the availability of certain information, that certain conditions are suddenly met, or the occurrence of a certain event happening in the real world. The principle is along the line of Discrete Event Systems [59].

World Model Events are used to notify other software components (observers) about the state of certain predefined conditions, which are relevant for the execution of driving maneuvers. The specification of each driving

maneuver requires that, in order to be safely performed, certain conditions have to be met, or certain information has to be available. Such conditions can be, for instance, related to traffic situations (e.g. "pedestrian n meters in front of the vehicle"), but might as well be related to the availability of sensor information (e.g. "GPS localization available").

The World Model Events enable a flexible way to define what information is relevant for observers, and provide an easy mechanism for quick information exchange between the World Model and its observers (e.g. the Real-Time Decision Making subsystem).

Other subsystems are actively notified by the World Model about the occurrence of certain *World Model Events* using the Observer[6] software design pattern. However, in order to enable a flexible exchange of information between the subsystems, while maintaining the theoretical concept of discrete event systems, the information regarding which of the multiple events occurred is encoded in the prototype implementation as Boolean variables, by assigning the value *true* when an event occurred, and the value *false* otherwise.
The state of all defined World Model Events is provided as a k-tuple:

$$WM_{events} = (w_1, w_2, ..., w_k) : w_l \in \{true, false\}, \tag{4.1}$$

where each element w_l $(l = 1, 2, .., k)$ represents an event.

Since the World Model updates its information about the traffic environment cyclically, the k-tuple WM_{events} is also updated cyclically. However, other subsystems are asynchronously notified about relevant changes in the traffic environment, enabling them to check the occurrence of certain traffic conditions, as defined for each event. A timestamp assigned to the event tuple may be used to ensure that the update operation is performed within the required time intervals.

World Model Events are only boolean variables and do not provide sufficient information for driving decisions. For this purpose, the World Model's object-oriented data structure provides full access to all available World

[6] Addressed in Subsection 4.6.3.

Model data.

4.5.2 Object-Oriented Data Structure

Besides World Model Events, the World Model provides other software components full access to all its data in the following way. The control software application contains one single instance of the class "WorldModel". Beginning with this single instance of "WorldModel", all data can be obtained by successively retrieving contained or related classes. For instance, the "WorldModel" operation *getCurrentRoad()* returns the instance of the class "Road" which models the road on which the vehicle is currently driving. The current traffic lane can be retrieved by calling the operation *getCurrentLane()* on the current road instance, and so on.

4.6 Applied Software Design Patterns

During the vehicle's movement, the World Model updates its data cyclicly and provides information in real-time to other software components. Consequently, this process requires parallel or at least concurrent execution, along with the execution of other software components, such as those executing driving maneuvers. In the current implementation, all World Model operations are executed in a multithreading programming model. However, due to a clearly defined interface, the presented concept allows the execution of the World Model software components in parallel, each on its own dedicated CPU.

The remainder of this subsection gives a short overview of the software design patterns applied for the World Model software component. Object-oriented software design patterns help to architect complex software systems in a reusable and extendable way. Furthermore, they help to document and explain complex architectures. Details about the applied design patterns can be found in [60].

4.6.1 Factory Method

The loading procedure of a priori information from an XML file (section 4.4.1) requires the processing of information stored in certian XML nodes and accordingly the creation and instantiation of a large number of various class instances. Consequently, for this task, the *Factory Method* (Figure 4.8) design pattern is most adequate, as it enables to embed the source code for the creation of class instances into the classes itself. Therefore, the application of this design pattern improves the source code readability, leads to better scalability, easier debugging and testing.

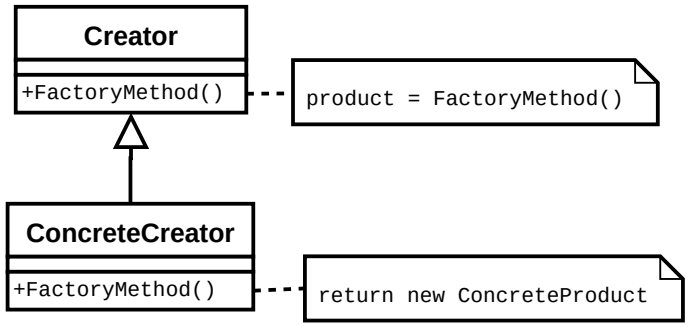


Figure 4.8: The *Factory Method* design pattern (adopted from [60]).

In order to divide and simplify the process of loading the XML file, the XML file is divided into XML element nodes. Each XML element is delegated as a parameter to its corresponding World Model class. Therefore, each class contains only the implementation required to create an instance of itself from an XML element node.

4.6.2 Singleton

Multiple identical representations of the traffic environment at the same time are not necessary, and would lead to wasted computing resources. Therefore, in order to ensure that there exists only one single instance of the class "WorldModel", this class is implemented as a *Singleton*. The *Singleton* design pattern impedes the recreation of a class instance if it already exists, while however enabling the easy access to this single instance.

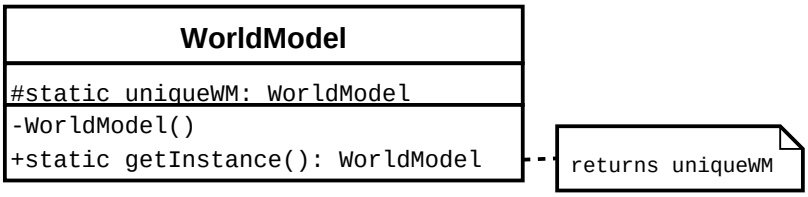


Figure 4.9: The *Singleton* design pattern.

Figure 4.9 shows the structure of this design pattern. The class constructor is protected, and the only possibility to create an instance is to call the public static method *getInstance()*. This method creates the instance *uniqueWM* only if it does not already exist, and returns a reference to it. Otherwise, if *uniqueWM* already exists, *getInstance()* returns a reference to it.

4.6.3 Observer

One of the non-functional requirements for the World Model is a flexible way to exchange information with other subsystems. The World Model actively notifies other subsystems (e.g. Decision Making subsystem) about events of relevance happening in the real world. The World Model Event interface is based on the *Observer* design pattern, as this pattern is best suited for implementing a notification mechanism for multiple subsystems.

Observer is a behavioral design pattern which is ideal for notifying many dependent objects (observers) when the state of one object (subject) has changed.

In this context, the World Model plays the role of the subject, while observers are any software components interested in receiving notifications from the World Model, such as a decision making subsystem. Figure 4.10 shows the interaction diagram between the World Model and its observers. When the World Model changes its state, it informs all the registered observers. The observers, in turn, request more information if needed.

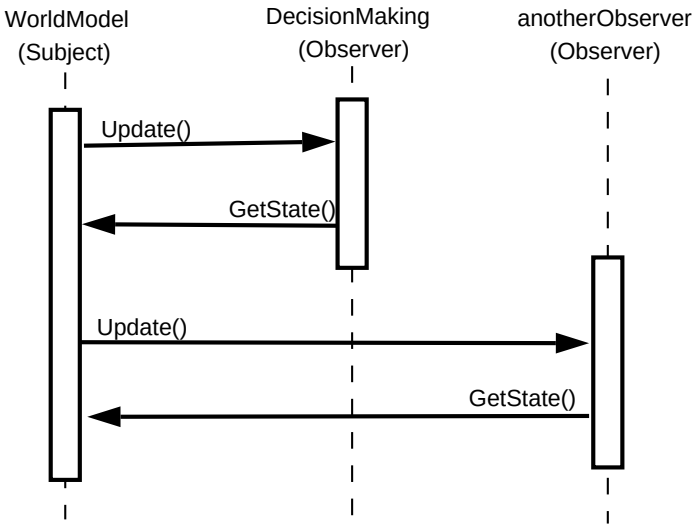


Figure 4.10: UML interaction diagram of the *Observer* design pattern.

4.7 Integration and Experimental Results

The presented World Model is part of a control software for autonomous city vehicles, developed at ICSL. Besides the presented World Model, the control software functionality includes tools to visualize the autonomous vehicle's on-board sensor data, to control its actuators, and to make real-time driving decisions. As the interfaces for the actuator commands for the real vehicle and the simulated vehicle are identical, the same control software is able to maneuver either a simulated autonomous vehicle or a real one.

4.7.1 3D Simulation

A 3D simulation environment for autonomous vehicles (Figure 4.11) has been used to test the integration of the World Model software component. The 3D simulation includes the simulation of an autonomous vehicle and its on-board sensors, such as GPS and LIDAR sensor. The vehicle interface to the simulated vehicle is identical to the vehicle interface of a real experimental vehicle. Furthermore, the 3D simulation environment is able to simulate static and dynamic obstacles, such as buildings, parked cars, pedestrians and other moving vehicles (Figure 4.12). Details about the simulation software

have been published in reference [61].

Figure 4.11: The 3D simulation environment. The traffic environment is a model of the test environment for the real autonomous vehicle at our Nathan Campus (Griffith University).

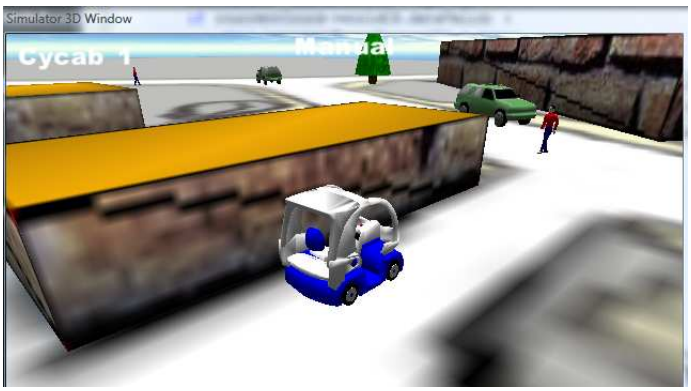


Figure 4.12: The autonomous vehicle in the 3D simulation, approaching an intersection.

The simulated traffic environment is the same as the real traffic environment at the test location for the real experimental vehicle at Griffith University, Nathan campus, Brisbane, Australia. Therefore, the simulation results reflect real road traffic situations.

Figure 4.13 shows a screenshot of the World Model graphical user interface

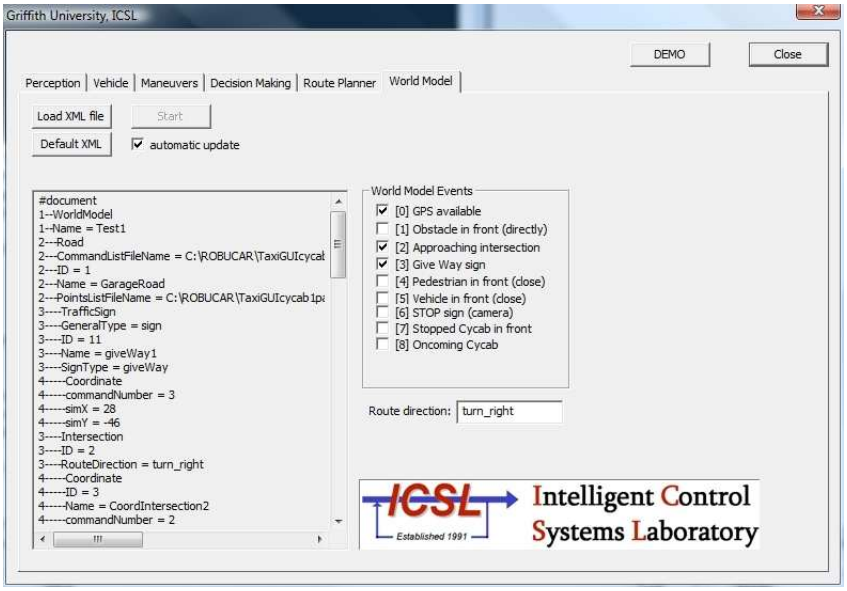


Figure 4.13: Screenshot of the World Model graphical user interface showing the loaded a priori XML file and the status of World Model events.

(GUI). In its current state of development, the control software graphical user interface shows the a priori information (XML file) and the status of world model events. During the autonomous vehicle's movement, the status of the world model events changes accordingly to the simulated traffic environment. This enables the visualization of current data in real-time. Besides that, the world model information can be logged for off-line analysis afterwards.

4.7.2 World Model Processing Time Measurements

One of the main requirements for the World Model is its ability to operate in real-time. Due to its relatively complex structure, one of the main concerns regarding the World Model software component could be the processing time which is required to organize the contained information (i.e. to insert and extract traffic features into and from the data structure). The overhead processing time for organizing the data needs to be kept to a minimum, even for a large number of perceived obstacles, traffic signs, etc.

In order to assess the processing time required to insert and extract information from the World Model data structure, a number of perceived obstacles

and traffic signs was inserted at run time, while running the 3D simulation. As part of the World Model updating operation which is performed in each execution cycle, the position of each obstacle and traffic sign is compared to the current vehicle position (Algorithm 4.3). Therefore, each updating operation requires the extraction of all obstacles and traffic signs. Consequently, measuring the processing time for a World Model execution loop reveals a realistic estimate about the minimum required time to insert and extract data from the World Model data structure.

Algorithm 4.3 measureWMprocessingTime

Require: current vehicle position

```
1: insert specified number of obstacles
2: insert specified number of traffic signs
3: start world model thread
4: start automatic driving
5: {thread loop}
6: while destination not reached do
7:    get current time t_before
8:    for each obstacle do
9:       compare obstacle position to current vehicle position
10:   end for
11:   for each traffic sign do
12:      compare traffic sign position to current vehicle position
13:   end for
14:   get current time t_after
15:   processing_time = t_after - t_before
16:   print processing_time on screen
17: end while
```

Table 4.1 shows the measured processing time in milliseconds (ms) for a World Model execution loop containing the specified number of simulated obstacles and traffic signs. The shown measurements include the processing time required to output the values on the screen. This explains the relatively long processing time (73 ms) for a small number (20) of obstacles and traffic signs, followed by a very slow increase as the number of obstacles and traffic signs increases significantly. Consequently, the World Model data structure is able to deal with a large number of perceived objects in real-time.

Table 4.1: Measured processing time in $[ms]$ for one World Model execution cycle (CPU: Intel T5450, 1.66GHz). The processing time has been measured according to Algorithm 4.3.

#Obstacles	**#Traffic Signs**	**time [ms]**
20	20	73
200	200	75
1 000	1 000	80
2 000	2 000	82
5 000	5 000	85
10 000	10 000	92
20 000	20 000	114

4.8 Discussion

This chapter aimed at modeling the traffic environment for autonomous city vehicles, independently of any specific sensor technology.

The World Model's cyclic updating mechanism, which is used to acquire information from sensors and communication, as well as to process and update information about the traffic environment, facilitates the integration of a large variety of sensor, communication, and perception subsystems, which provide information input to the autonomous vehicle's control system.

Furthermore, the World Model's event-based mechanism, which is used to notify other subsystems, such as the Real-Time Decision Making subsystem about asynchronous events, i.e., relevant changes of the traffic conditions, enables the decoupling of the World Model's operational dynamics from other subsystems, and consequently enables them to operate at an execution rate, which is independent from the one of the World Model.

The developed World Model fulfills its non-functional requirements as follows:

- The World Model real-time performance measurements show that the execution time for basic data management operations (i.e. inserting and retrieving of data) fulfills real-time requirements. It was also found that the execution time of each information updating cycle depends on

the processing time of the incorporated sensor and communication components. This means that time-intensive applications, such as vision algorithms, will have major impact on the processing time values.

Nevertheless, the integration of all data obtained from sensor and communication components into the World Model simplifies the assessment and monitoring of the real-time performance of all incorporated components. This can be for instance achieved by ensuring a minimum update frequency of the World Model events.

- Interoperability with various types of sensor components and communication subsystems is achieved through the definition of the World Model information in a generic, sensor-independent way.

- The object-oriented design allows to model the relationships between entities in an intuitive way, reflecting their relationship in the real traffic environment (e.g. obstacles on a road). Consequently, the object-oriented design simplifies the development process, while the application of software design patterns makes the World Model modular, thus easier to overview and understand.

- Portability to other autonomous vehicle decision making & control system architectures is achieved through a clearly defined API. Along with its flexible World Model Event interface, the World Model, or even the entire Perception Subsystem can be easily integrated into other vehicle decision making & control system architectures, as long as they have a modular structure.

- As it allows to define arbitrary events, the developed World Model's information exchange mechanism enables the incorporation of other subsystems which require information from the World Model, in addition to the Real-Time Decision Making subsystem.

- Reusability of the World Model is possible in other applications, which have similar information requirements as those of a decision making & control system for autonomous city vehicles.

While this chapter has presented the World Model which is used as input by the Real-time Decision Making subsystem, the following Chapter 5 elaborates in details on the real-time decision making approach.

Chapter 5

Real-Time Decision Making

5.1 Introduction

Decision making in this context refers to the problem of identifying the most appropriate[1] driving maneuver to be performed under the given road traffic circumstances. Due to the highly dynamic urban traffic environment, the driving decisions need to be made in real-time and cannot be planned in advance.

As already mentioned, according to the published material, the currently proposed solutions for the real-time decision making problem for autonomous city vehicles are not sufficient to successfully deal with the high complexity of real-world, non-simplified urban traffic conditions.

This chapter addresses this topic, and presents a solution which enables civilian autonomous city vehicles to deal with real-world urban traffic.

5.2 System Specification Development

The specification of requirements is a fundamental part of the development cycle of any complex system. Any attempt to develop a complex system like

[1]Here, the term most appropriate refers to fulfilling driving objectives regarding safety, conforming to traffic rules, but also non-safety relevant objectives, such as improving efficiency and comfort. This is further clarified later in this chapter.

an autonomous city vehicle without a detailed requirement specification is likely to fail.

Regarding the development of the Real-Time Decision Making subsystem, the specification of requirements is crucial for ensuring the road safety (i.e. driving in such a way that any risks which might cause traffic accidents are avoided). Without a specification for this subsystem, the implementation, and eventually human lives will depend on the software developers' interpretation of requirements, without the possibility to verify the algorithm correctness.

A detailed discussion of how to develop a complete specification for the decision making subsystem for autonomous city vehicles would go beyond the scope of this thesis. Nevertheless, this topic is addressed by proposing and demonstrating a generic, and extendable approach, which is adequate to model and implement a complex specification of decision making requirements in the following way.

Table 5.1: Some World Model Events of relevance to the selected driving maneuvers *(OvertakeRight, FollowLane* and *Stop&Go*).

World Model Event	**Explanation**
w_1: right lane boundary detected	The right lane boundary has been detected and its position is known
w_2: left lane boundary detected	The left lane boundary has been detected and its position is known
w_3: right lane boundary crossable	Changing lanes onto the right lane is possible
w_4: right ongoing lane detected	The right lane has been detected
w_5: obstacle on right ongoing lane	A moving or static obstacle has been detected on the right lane
w_6: "no overtaking" sign detected	Overtaking is prohibited
w_7: approaching intersection	The vehicle is approaching a close intersection
w_8: any kind of obstacle in front	A static or moving obstacle has been detected in front
w_9: moving vehicle in front	Another vehicle has been detected in front
w_{10}: static obstacle in front	A detected obstacle is blocking the lane
⋮	⋮

Without the loss of generality, it can be assumed that the decision making specification requires the decision making subsystem to respond to certain traffic conditions in a defined way, i.e. certain traffic conditions imply the execution of certain driving maneuvers. The occurrence of any traffic condition is modelled by a set of discrete events, which can occur simultaneously, and which are detected by the vehicle's on-board sensors, or through communication. These discrete events are received by the decision making subsystem in the form of World Model Events. Depending on a defined set of discrete events, the specification defines which driving maneuvers can be performed safely. Table 5.1 shows examples of selected World Model Events, while Table 5.2 lists an example of World Model Events of relevance to the specified driving maneuvers. For the sake of simplicity, the tables show only a short abstract of an otherwise very large number of events.

Table 5.2: Required status of relevant World Model Events (defined in Table 5.1) and driving maneuvers. A driving maneuver is operational only if the relevant World Model Events have the specified true/false status.

Req. status of relevant WM Events	**Driving Maneuver**
w_1: true w_3: true w_4: true w_5: false w_6: false w_7: false w_8: true	m_1: OvertakeRight (overtaking on the right-hand side)
w_1: true w_2: true w_{10}: false	m_2: FollowLane (following a detected traffic lane)
w_9: true	m_3: Stop&Go
⋮	⋮

Consequently, the decision making subsystem is modeled as a discrete event-driven system [59]. Having in mind the complexity of urban traffic, a complete specification for a typical decision making subsystem will consist of

a large number of defined inputs (i.e. discrete events), and a large number of outputs (i.e. driving maneuvers), which are logically interrelated in a complex way.

5.3 Definition of the Decision Making Problem

The decision making problem can be specified as follows. The following is given:

- a set $M_{all} = \{m_1, m_2, ...m_n\}$, $n \in \mathbb{N}$, of all available driving maneuvers which can be performed by the autonomous vehicle (Chapter 6),
- a k-tuple $(w_1, w_2, ..., w_k) \in W_{events}$ of World Model Events (Chapter 4), $w_l \in \{0, 1\},\ l = 1, 2, ..., k$,
- a route planner[2] direction indication, which specifies the vehicle's future travel direction:

$$d_i \in D_{route} = \{forward\ straight, forward\ right, forward\ left, turn\ around\}$$

The general problem of decision making in this context is to identify the most appropriate driving maneuver $m_{most\_appr.} \in M_{all}$, which leads to an autonomous vehicle driving behavior conforming to the specification.

5.3.1 Decomposition into Subproblems

The general decision making problem is decomposed into two consecutive stages (Figure 5.1):

[2]The route planner provides the planned travel route, which is either given a priori, or generated by a navigation system or path planner.

1. Decision regarding *feasible, safety-critical driving maneuvers* subject to World Model Events and route planner indication. A driving maneuver is defined as feasible if it can be safely performed in a specific traffic situation, and is conforming to the road traffic rules. In any traffic situation, there can be multiple feasible driving maneuvers (e.g. passing a stopped vehicle or waiting for it to continue driving). In order to ensure the road safety, it is assumed that the autonomous vehicle will always obey the traffic rules. However, since today's traffic rules have been developed for human drivers and sense of reason, these rules might need to be adapted for autonomous vehicles in the future.

2. Decision regarding *the* most appropriate driving maneuver. This stage selects and starts the execution of one single driving maneuver, which is the most appropriate for the specific traffic situation. Since only those driving maneuvers are considered in this stage which have been selected as feasible (and therefore safe), this stage is not safety-critical because it does not include any decision making attributes which affect road safety; however, it becomes safety-critical as soon as it enters its execution stage, with respect to meeting real-time execution requirements.

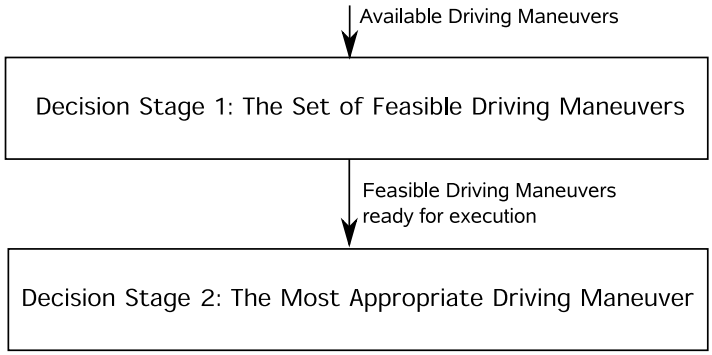


Figure 5.1: The two stages of the decision making process.

The decomposition into two stages with different objectives leads to sub-problems with manageable complexity, enabling the verification and testing

of each stage in particular. While the main focus of the first stage is to determine which driving maneuvers are safe and conform to traffic rules, the second, non safety critical decision stage focuses on improving comfort and efficiency.

The entire decision making process (i.e. the two stages) is executed cyclicly. Once a driving maneuver has been activated, it remains active as long its execution is feasible, and it is executed until it finishes in one of its two final states q_F or q_E. If a driving maneuver becomes infeasible during its execution, the decision making subsystem is able to abort its execution, and activate another driving maneuver instead.

5.4 Requirements

The following requirements are fundamental for developing a solution which is usable for real-world applications:

- *Real-Time Capable*: The ability to make correct decisions within specified time limits is crucial for autonomous vehicles, as it is for human drivers. Being part of a mission-critical real-time system, one of the main requirements to the decision making solution is its ability to deliver a correct decision in real-time. Therefore, in order to prove the real-time capability, the worst-case execution time of the decision making algorithm needs to be below specified time limits.

- *Verifiable*: The verification, i.e. "the systematic approach to proving the correctness of programs" [62], is one of the most important aspects for safety-critical, hard real-time systems [63], such as autonomous city vehicles. Verification approaches which can be used to prove the functional correctness are for instance approaches based on Operational Reasoning or Axiomatic Reasoning [62]. Other verification methods, which are used in practice but do not prove the correctness, are informal reasoning (e.g. source code inspections), methods based on formal specifications, or testing [64]. As solution approaches which are based

on formal methods (e.g. Petri nets) lead in general to an easier verification process [64], this thesis applies a formal method, namely Petri nets for the first, safety-crucial decision making stage. For instance, one of the properties which need to be verified in a decision making algorithm is whether certain driving maneuvers are performed under defined preconditions. Using a Petri net model, this property corresponds to the reachability problem, i.e. whether a marking is reachable from another marking through a firing sequence [64].

- *Scalable*: The decision making algorithm should be scalable, having in mind the future integration of new information sources about the vehicle's environment, such as new sensors or information resulting from cooperation between autonomous vehicles or communication with a vehicle management centre. Furthermore, the solution should allow the addition of new driving maneuvers or the adaptation to changed traffic rules. Such additions should not require a significant change or redesign of the existing decision making functionality.

5.5 Decision Stage 1: Feasible, Safety-Critical Driving Maneuvers[3]

This section presents a modeling approach for the first stage of the decision making problem, the decision regarding the set of feasible driving maneuvers.

5.5.1 General Approach

The goal of this stage is to select the feasible driving maneuvers, i.e. the subset of all driving maneuvers, which can be performed without putting any traffic participants at risk.

The following aspects are relevant for the selection of feasible driving maneuvers (Figure 5.2):

[3]The presented concept of the first decision making stage has been published in [47].

- Information about the vehicle's environment, which is provided in the form of World Model Events.
- Knowledge about traffic rules and compliance with them, which is provided by the specification (Table 5.2).
- Information about the planned travel direction, which is assumed to be provided by the route planner.

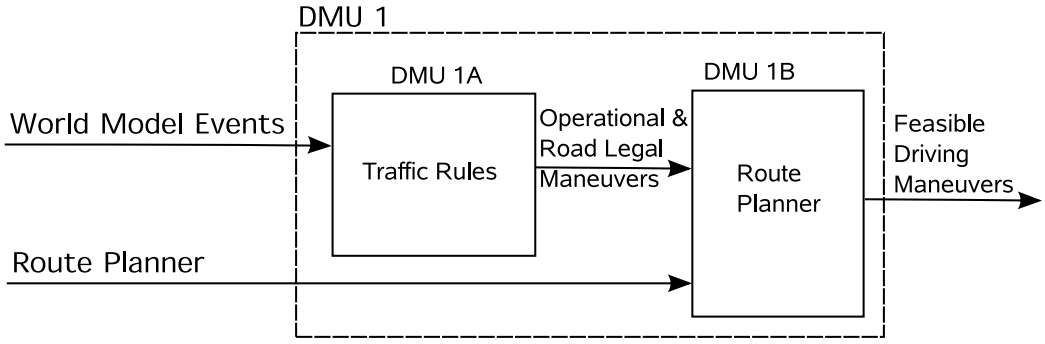


Figure 5.2: Decision Stage1: decision making unit for the selection of feasible, safety-critical driving maneuvers.

In order perform a driving maneuver, information about the vehicle's traffic environment is required. This information is provided by the World Model in the form of events. Therefore, occurring World Model Events define which driving maneuvers are operational, i.e. which maneuvers can be safely performed based on the available information about the traffic environment. In order to comply to traffic rules, additional restrictions of driving maneuvers are required, for example not allowing overtaking in an intersection. This knowledge about traffic rules is embedded in the first step of this stage (DMU1A)[4], by restricting the selection of certain driving maneuvers as feasible, if they do not conform to the traffic rules.

As a third aspect, the planned travel route, which is either given a priori, or generated by a navigation system or path planner, plays a further role in reducing the number of candidate driving maneuvers (DMU1B in Figure

[4]Decision Making Unit

5.2). Driving maneuvers which lead the vehicle into a wrong direction, or maneuver it inadequately with respect to the planned route, are omitted from the set of feasible driving maneuvers. For instance overtaking a slower vehicle while the route planner indicates a necessary U-turn might not be adequate. Therefore, a route planner provides the decision making module with information about the prospective planned route, well in advance (e.g. 50-150m) before required turns and changes of direction.

Consequently, the large number of factors to be considered in this first decision making stage requires a model which enables the design and analysis of a highly complex operational behavior. For this purpose, a Petri net is a suitable formal modeling method [65].

5.5.2 Petri Net Model

The decision making unit shown in Figure 5.2 is modeled as a Petri net[5] (Figure 5.3). The Petri net consists of two subnets, each representing the decision making units DMU1A and DMU1B, respectively.

The structure of the Petri net modeling DMU1 (Figure 5.3) is as follows. The input to the Petri net consists of two sets of input places. The first set of input places represents World Model Events; each World Model Event is represented by one single input place. The second set of input places represents the route planner indication; each route planner direction indication is represented by one input place.

The output of the Petri net modeling DMU1 represents driving maneuvers. There is one output place of the Petri net for each available driving maneuver.

The Petri net in Figure 5.3 is executed cyclically, and operates as follows. At the beginning of an execution cycle, the Petri net's input places are marked. A token is placed into each place which represents a World Model Event[6] which has the value *true*. Similarly, marks are placed in the input

[5] An overview of Petri nets and modeling of systems with Petri nets can be found in Appendix B.

[6] World Model Events represent asynchronous events occurring on the road, and are used as conditions for the firing of Petri net transitions.

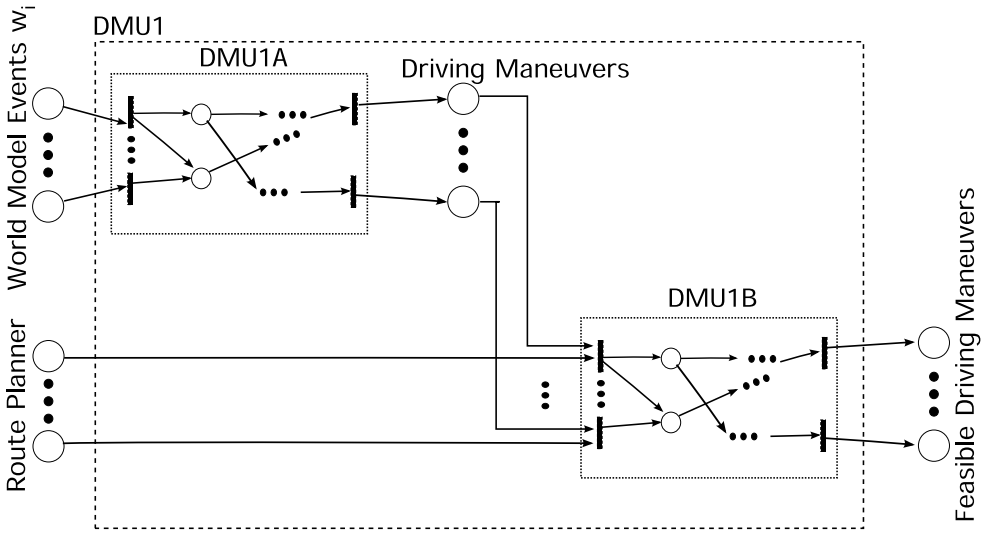


Figure 5.3: Petri net modeling the decision making unit for the selection of feasible, safety-critical driving maneuvers. Two subnets model the subnets DMU1A and DMU1B, respectively (Fig. 5.2).

places which correspond to the planned travel direction, as indicated by the path planner.

The purpose of subnet DMU1A is to select those driving maneuvers which are operational (i.e. executable due to sufficient information about the vehicle's traffic environment) and which conform to traffic rules. Therefore, the knowledge and application of traffic rules is embedded in the execution structure of the subnet DMU1A. This subnet marks only those places which represent operational driving maneuvers. The result of DMU1A is passed to the second subnet DMU1B.

The purpose of the subnet DMU1B is to filter out those driving maneuvers which were determined as operational by DMU1A, however which do not lead the vehicle into the direction indicated by the route planner. Therefore, the subnet DMU1B receives both inputs from the route planner and inputs from the subnet DMU1A.

After its execution, DMU1B places a token into each Petri net output place which represents a driving maneuver which is operational, safe to perform, and according to the route planner indication.

Consequently, the execution of the entire Petri net results in the selection

of feasible driving maneuvers (i.e. operational, safe, and according to the route planner). After each decision making execution cycle, all marked places of the Petri net are cleared, and a new decision making cycle begins.

The following example explains this decision making stage in details.

5.5.3 Example: Deciding between Stop&Go, Overtaking and Lane Following

This example explains the first decision making stage regarding feasible driving maneuvers assuming the traffic situation shown in Figure 5.4.

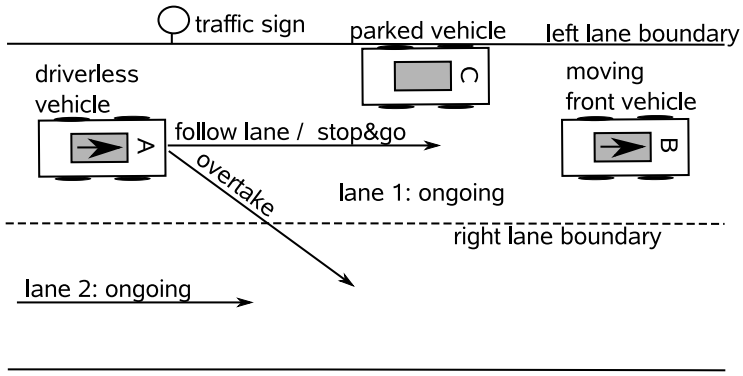


Figure 5.4: The assumed traffic situation for example of decision stage 1 (left-hand side driving).

In this example, the set of all available driving maneuvers is assumed to consist of the following three maneuvers:

$$M_{all} = \{OvertakeRight, FollowLane, Stop\&Go\} = \{m_1, m_2, m_3\}.$$

The driving maneuver *OvertakeRight* is used to overtake a slower vehicle on the right side (assuming left-hand side driving). In order to reduce the complexity of this example, it is assumed that the autonomous vehicle is able to overtake only when a parallel ongoing right lane is available and only when this lane is obstacle-free.

The driving maneuver *FollowLane* is used to follow a detected traffic lane by maneuvering the autonomous vehicle between the two lane boundaries. The purpose of the *Stop&Go* driving maneuver is to follow a front vehicle and

eventually stop when a minimum distance between the autonomous vehicle and the front vehicle is under-run.

The possibility to execute a driving maneuver depends on the status of certain World Model Events. This example uses the definition of World Model Events as listed and explained in Table 5.1. The relation between the driving maneuvers and World Model Events is shown in Table 5.2, which lists the relevant World Model Events and required status for each driving maneuver to be operational. A driving maneuver is defined to be operational only if all relevant World Model Events have correct true/false status as specified in Table 5.2.

Table 5.3: Adequate driving maneuvers depending on the route planner direction indication.

Route Planner Indication d_i	**Adequate Driving Maneuvers**
d_1: forward straight (follow road direction)	m_1: OvertakeRight m_2: FollowLane m_3: Stop&Go ⋮
d_2: forward right (right turn ahead)	m_1: OvertakeRight m_2: FollowLane m_3: Stop&Go ⋮
d_3: forward left (left turn ahead)	m_2: FollowLane m_3: Stop&Go ⋮
d_4: turn around (U-turn)	m_2: FollowLane m_3: Stop&Go ⋮

Table 5.3 lists the driving maneuvers which are defined to be adequate with respect to the route planner indication. The travel direction indicated by the route planner is denoted as

$$d_i \in D_{route} = \{forward\ straight, forward\ right, forward\ left, turn\ around\},$$

where $d_1 = forward\ straight$, $d_2 = forward\ right$, $d_3 = forward\ left$, and $d_4 = turn\ around$. In this example, the indication $forward\ straight$ and $forward\ right$ defines all three driving maneuvers to be adequate, while the indications $forward\ left$ and *turn around* prohibit the maneuver *OvertakeRight* and consider only the driving maneuvers *Stop&Go* and *FollowLane* to be adequate.

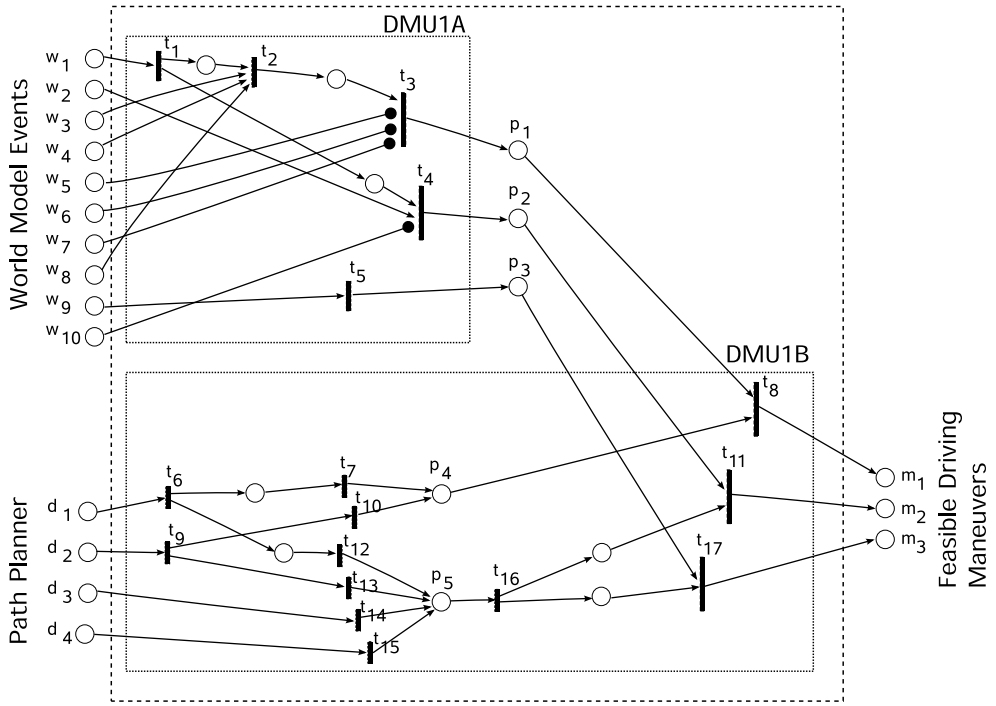


Figure 5.5: Petri net structure modeling the selection of feasible driving maneuvers with ten World Model Events $w_1, .., w_{10}$, four Path Planner directions $d_1, .., d_4$, and three available driving maneuvers m_1, m_2, m_3. The arcs with circles (e.g. from the input place w_5 to the transition t_3) are *inhibitor* arcs. The firing rule for transitions with inhibitor arcs is opposite to usual (arrow) arcs: a transition fires when its inhibitor places are unmarked [65].

The function

$$f_{feasible} : M_{all} \times W_{events} \times D_{route} \rightarrow \{0, 1\}$$

is defined according to the Tables 5.2 and 5.3, and is modeled by the Petri net shown in Figure 5.5. The Petri net is subdivided into two subnets, denoted DMU1A and DMU1B.

The subnet DMU1A receives inputs from the World Model. Each input place $w_1, w_2, ..., w_{10}$ represents a World Model Event, as defined in Table 5.1. A marked input place denotes a *true* event, an unmarked place denotes an event with value *false*. Each Petri net input place which represents a *true* World Model Event is marked with one single marking.

The execution of the Petri net leads to the following transition firing sequence. The transition t_1 doubles the marking of input place w_1, in order to avoid a conflict between transitions t_2 and t_4[7]. Consequently, the transition t_2 fires if $w_1 = w_3 = w_4 = w_8 = true$. The transition t_3 fires if and only if t_2 fires and $w_5 = w_6 = w_7 = false$. Therefore, the place P_1 is marked if and only if $w_1 = w_3 = w_4 = w_8 = true$ AND $w_5 = w_6 = w_7 = false$. Similarly, the places P_2 and P_3 are marked accordingly.

The subnet DMU1B receives both input from the route planner and from the subnet DMU1A. Each input place $d_i,\ i \in \{1, 2, 3, 4\}$ represents one of the direction indications, where $d_1 = forward\ straight$, $d_2 = forward\ right$, $d_3 = forward\ left$, and $d_4 = turn\ around$. Therefore, the place P_4 contains at least one marking when d_1 OR d_2 are marked. Similarly, the place P_5 contains at least one marking when at least one of the $d_i, i \in \{1, 2, 3, 4\}$ is marked.

Combined, the subnets DMU1A and DMU1B mark the Petri net's output places m_1, m_2 and m_3 according to the specification. The Petri net's marked output places represent the driving maneuvers which are feasible. A driving maneuver m_i is selected as feasible, if the corresponding Petri net output place contains at least one marking.

In the current prototype implementation, since this is the safest option, in the unlikely event that there is no feasible driving maneuver, the vehicle stops, and waits for a maneuver to become feasible.

[7]The purpose of transitions t_1, t_6, t_9 and t_{16} is to avoid conflict situations, where the firing of one transition disables another transition which shares a common input place. For example, without t_1, the transitions t_2 and t_4 would share the same input place w_1. In that case, the firing of t_2 would make the firing of t_4 impossible.

5.6 Decision Stage 2: Selecting the Most Appropriate Driving Maneuver using MCDM[8]

The goal of the second decision making stage (Figure 5.1) is to select and execute the most appropriate alternative from those driving maneuvers which have been determined to be feasible in the current traffic situation. Because the set of feasible driving maneuvers already contains only those maneuvers which can be safely performed in the specific road traffic situation, this stage does not include any safety-relevant decision making attributes, and its main objective is to maximize the efficiency and comfort. Nevertheless, it is safety-relevant with respect to fulfilling the real-time execution requirements.

Also, each driving maneuver offers multiple *execution alternatives*, which can be selected through discrete driving maneuver parameters. For instance, the overtaking maneuver could be performed at low or high speed, in distant or close proximity to the front vehicle, and on the right or left hand side. In order to select the most appropriate driving maneuver, and for it the most appropriate execution alternative, Multiple Criteria Decision Making (MCDM) is applied as follows.

Objectives: a hierarchy of objectives is defined starting from a main, most general driving objective, which is then further successively broken down into more specific and therefore more operational objectives on lower hierarchy levels. Eventually, the bottom level of the objective hierarchy contains only objectives which are fully operational and which are measurable through their attributes (Figure 5.6).

The most general objective for autonomous driving is to safely reach the specified destination. More precisely, this objective is broken down into a lower hierarchy level containing more specific objectives, which specify how to achieve the objective of the higher level. Thus, we define the following objective hierarchy consisting of four ($k = 4$) level 2 objectives:

[8]This topic is addressed in our publication "Multiple Criteria-Based Real-Time Decision Making by Autonomous City Vehicles" [66].

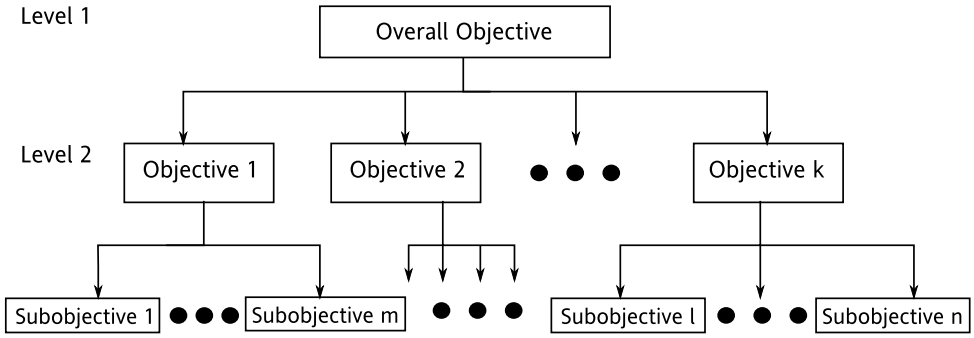


Figure 5.6: Hierarchy of objectives [67].

- Drive to destination safely =: obj^{Level1}
  - Stay within road boundaries =: obj_1^{Level2}
    - keep distance to right boundary := $attr_1$
    - keep distance to left boundary := $attr_2$
  - Keep safety distances =: obj_2^{Level2}
    - keep distance to front vehicle := $attr_3$
    - keep distance to moving obstacles := $attr_4$
    - keep distance to static obstacles := $attr_5$
  - Do not collide =: obj_3^{Level2}
    - keep minimum distance to obstacles := $attr_6$
    - drive around obstacles := $attr_7$
    - avoid sudden braking := $attr_8$
    - avoid quick lane changes := $attr_9$
  - Minimize waiting time =: obj_4^{Level2}
    - maintain minimum speed := $attr_{10}$
    - avoid stops := $attr_{11}$

Attributes: a set of measurable attributes

$$\{attr_1, attr_2, .., attr_p\},\ p \in \mathbb{N}\ (\mathbb{N}\text{=set of natural numbers})$$

is assigned to each objective on the lowest hierarchy level (in our example it is level 2 with $p = 11$). An attribute is a property of a specific objective. In order to define various levels of importance, *weights* may be assigned to each attribute.

Alternatives: in the context of our application, decision alternatives correspond to the execution of driving maneuvers. Therefore, in a first step, we

regard each element of the set of driving maneuvers $\{M^1, M^2, .., M^n\}(n \in \mathbb{N})$ to be an element of the *set of alternatives* A:

$$A = \{M^1, M^2, .., M^n\}$$

However, each driving maneuver M^m $(1 \leq m \leq n)$ offers one or multiple *execution alternatives* by specifying discrete[9] attribute values (e.g. fast/slow, close/far, etc.).
Therefore, we obtain:

$$\begin{aligned} M^1 &= \{M_1^1, M_2^1, .., M_j^1\} \\ M^2 &= \{M_1^2, M_2^2, .., M_k^2\} \\ &\vdots \\ M^n &= \{M_1^n, M_2^n, .., M_l^n\}, \end{aligned}$$

where n denotes the number of driving maneuvers, and j, k, l the number of execution alternatives for the maneuvers M^1, M^2, and M^n respectively.

Therefore, the set of alternatives A contains all execution alternatives of all n driving maneuvers:

$$A \;=\; \bigcup_{m=1}^{n} M^m = \{M_1^1, M_2^1, .., M_j^1, M_1^2, .., M_k^2, .., M_1^n, ..M_l^n\}$$

For the sake of readability, we denote all alternatives as:

$$A = \{a_1, a_2, .., a_q\},\ (q = j + k + .. + l)$$

Utility Functions: utility functions $f_1(a_i), ..., f_p(a_i)$ specify the level of achievement of an objective by an alternative $a_i \in A$ $(i \in [1, q])$ with respect to each of the p attributes.
For each attribute $attr_i$ $(i \in [1, p])$, we define a utility function $f_{attr_i} = f_i$:

$$f_i : A \rightarrow [0, 1]$$

Consequently, defining utility functions f_i for all alternatives $a_1, a_2, .., a_q$ and all attributes $attr_i$ $(i \in [1, p])$ results in the following decision matrix:

[9] In order to reduce the computational costs, we discretize the otherwise continuous parameter values of driving maneuvers.

a_i	$attr_1$	$attr_2$	...	$attr_p$
a_1	$f_1(a_1)$	$f_2(a_1)$	...	$f_p(a_1)$
a_2	$f_1(a_2)$	$f_2(a_2)$	...	$f_p(a_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
a_q	$f_1(a_q)$	$f_2(a_q)$	...	$f_p(a_q)$

The remaining problem is to select the best among the feasible alternatives. A variety of MCDM methods can be applied in order to solve this problem, such as dominance methods, satisficing methods, sequential elimination methods, or scoring methods [68]. In the following example we choose a widely used scoring method, the Simple Additive Weighting Method, in which the value $V(a_i)$ of an alternative a_i is calculated by multiplying the utility function values with the attribute weights and then summing the products over all attributes (see equation 5.1) [68]. The alternative with the highest value is then chosen.

Instead of using the Simple Additive Weighting Method, other methods can also be applied, in order to seek a Pareto optimal (noninferior) solution, i.e. a solution where no other alternative will improve one attribute without degrading at least another attribute [67]. This comparison has to therefore be performed in the context of finding safety-critical solutions. Also, having in mind the decision making algorithm's hard real-time requirements which in turn are safety-crucial, the benefits of finding a Pareto optimal solution will be achieved if and only if the necessary computing power is available.

5.6.1 Example

In this example we assume the traffic situation shown in Figure 5.7. The vehicle on the left side is an autonomous vehicle, passing a stopped vehicle.

For this situation, without oncoming traffic, the first decision making stage determined the following two driving maneuvers as *feasible*: Passing the stopped vehicle, or Stop&Go (i.e. waiting behind the temporarily stopped vehicle).

Figure 5.7: The autonomous vehicle (left) passing a stopped vehicle (right) (right-hand side driving).

For the sake of simplicity, we assume that only the following few execution alternatives for the two driving maneuvers are possible:

- Passing maneuver M^1:
  - a_1 := *speed*=slow, *lateral distance*=small
  - a_2 := *speed*=slow, *lateral distance*=large
  - a_3 := *speed*=fast, *lateral distance*=small
  - a_4 := *speed*=fast, *lateral distance*=large
- Stop&Go maneuver M^2:
  - a_5 := *distance to front vehicle*=small
  - a_6 := *distance to front vehicle*=large

Consequently, the set of feasible alternatives is:

$$A = \{a_1, a_2, .., a_6\}$$

The utility functions $f_i(A)$ evaluate the achievement level of each attribute i for each of the 6 alternatives. In order to allow comparisons between the levels of achievement of different objectives, the values of the utility functions f_i are scaled to a common measurement scale, the interval of real numbers between 0 and 1. We define:

$$f_i \in [0, 1] \subset \mathbb{R},$$

where the value 1 denotes the optimal achievement of an objective, while 0 denotes that the objective is not achieved at all.

We define the utility functions as follows. Each of the 6 alternatives are rated regarding on how well they fulfill the driving objectives on the lowest hierarchy level. We rate the alternatives on a scale from 0 to 1, where:

- 1 denotes optimal fulfillment of the objective,
- 0.75 denotes good fulfillment,
- 0.5 denotes indifference,
- 0.25 denotes bad fulfillment,
- 0 denotes unsatisfactory fulfillment.

In our example, the utility function values are assigned based on heuristics reflecting the preferences of a human driver, as listed in Table 5.4.
For calculating the best solution, we choose in this example the Simple Additive Weighting Method [68]. We define the *value* of an alternative a_i as follows:

$$V(a_i) := \sum_{j=1}^{p} w_j f_j(a_i), \tag{5.1}$$

where p denotes the number of attributes.

Each attribute is assigned a weight w_j, which reflects its importance. For autonomous driving, the importance of various objectives changes depending on the road conditions. For example, on a wide boulevard at higher speed, the attribute "$attr_8$: avoid sudden braking" is more important than the

Table 5.4: Heuristic definition of utility functions $f_i : A \rightarrow [0,1]$ for the 6 alternatives $a_1, .., a_6$ and 11 attributes $attr_1, .., attr_{11}$. Weights w_i indicate the level of importance. The column $V(a_i)$ lists the calculated values for each alternative a_i.

a_i	$attr_1$	$attr_2$	$attr_3$	$attr_4$	$attr_5$	$attr_6$	$attr_7$	$attr_8$	$attr_9$	$attr_{10}$	$attr_{11}$	$V(a_i)$
a_1	1	0.5	0.5	0.5	0.25	0.25	1	0.5	0.75	0.75	1	11
a_2	1	0.25	0.5	0.5	1	1	1	0.5	0.75	0.75	1	12.25
a_3	1	0.5	0.5	0.5	0.25	0.25	1	1	0.25	1	1	12
a_4	1	0.25	0.5	0.5	1	1	1	1	0.25	1	1	**13.25**
a_5	0.5	0.5	0.25	0.5	0.25	0.25	0	0	1	0	0	4.5
a_6	0.5	0.5	1	0.5	0.75	1	0	0.25	1	0	0	8
w_i	$w_1 = 1$	$w_2 = 1$	$w_3 = 2$	$w_4 = 1$	$w_5 = 1$	$w_6 = 1$	$w_7 = 1$	$w_8 = 3$	$w_9 = 2$	$w_{10} = 2$	$w_{11} = 2$	

attribute "$attr_1$: keep distance to right boundary". However, in a residential area, the opposite might be the case. Consequently, instead of defining invariable attribute weights, this method offers the possibility to adapt the attribute weights, and therefore the decision preferences, according to the current traffic environment. In our example, the values which specify the level of importance w_j for each attribute of Section 5.6, are listed in Table 5.4.

Using the utility functions and attribute weights as listed in Table 5.4, we calculate the value of each alternative:

$$\begin{aligned} V(a_1) &= \sum_{j=1}^{11} w_j f_j(a_1) \\ &= 1*1 + 1*0.5 + 2*0.5 + 1*0.5 + 1*0.25 \\ &\quad + 1*0.25 + 1*1 + 3*0.5 + 2*0.75 + 2*0.75 \\ &\quad + 2*1 = 11.0 \\ V(a_2) &= 12.25;\ V(a_3) = 12.0;\ V(a_4) = 13.25;\ V(a_5) = 4.5 \\ V(a_6) &= 8.0 \end{aligned}$$

The highest value $\underset{1 \leq i \leq 6}{max}\ V(a_i) = 13.25$ is achieved by alternative a_4 (passing at fast speed with a large lateral distance to the stopped vehicle). Therefore, this alternative is chosen for execution.

5.7 Ensuring Real-Time Performance

As elaborated in the list of requirements, one of the most important aspects for the decision making process its ability to perform in real-time, i.e. to deliver correct results within specified time limits. However, in this application, the real-time requirements depend on a variety of factors, such as the vehicle's speed, the surrounding environment, etc. At high vehicle speed, the decision making subsystem, and all other safety-relevant components, need to react faster than at slow vehicle speed. On the other side, even at low speed, while driving in an urban area with pedestrians nearby, the decision making subsystem needs to react very quickly in order to avoid collisions. Therefore, these variable real-time requirements need to be elaborated and specified accordingly, before implementing the system.

Since the goal is to achieve a driving performance similar to a human driver, however with improved safety, it can be estimated that the overall guaranteed response time of the entire autonomous vehicle control system needs to be at least as short as the typical reaction time of a human driver. For the calculation of the autonomous vehicle's control system response time, all hardware and software subsystems need to be included. Therefore, since it is a critical subsystem of the autonomous vehicle's control software, the decision making subsystem's real-time performance needs to be ensured.

The developed decision making approach facilitates the analysis, simulation, and testing of real-time performance in the following ways:

- Decision Stage 1: The Petri net, which models the first, safety-critical stage of the decision making process can be analyzed, simulated, and tested independently from the rest of the autonomous vehicle's control system components. This also allows to measure the execution times for the Petri net execution on the actual computing system, including the worst-case scenarios, when all transitions fire.

- Decision Stage 2: Although the second decision making stage does not include safety-critical attributes, its real-time performance ability does have an effect on the entire decision making process. Therefore, the

number of MCDM objectives, the choice of the MCDM method, and for instance the calculation cost for seeking Pareto optimal solutions, need to be in line with real-time requirements. For this reason, the Simple Additive Weighting Method has been chosen in the implementation, due to its low calculation costs.

Consequently, the presented approach allows to analyse and ensure the real-time performance of the entire decision making process, without the effects of its depending input and output subsystems (i.e. World Model and Driving Maneuvers). Eventually, besides ensuring the real-time performance of each subcomponent, the real-time execution performance of the entire autonomous vehicle decision making & control system needs to be analyzed and tested using the usual methods and tools for real-time systems [63,64,69].

5.7.1 Real-Time Performance Measurements

In order to assess the real-time performance of the first, Petri net based decision making stage, the Petri net implementation has been tested and its execution performance has been measured independently from the vehicle control software. For this purpose, two different Petri net structures have been created, which reflect the building blocks of a complex decision making net:

- A Petri net with a single transition with multiple inputs and multiple outputs (Figure 5.8), and;
- A Petri net with multiple transitions with single inputs and single outputs (Figure 5.9).

Single Transition with Multiple Inputs and Multiple Outputs

The first measured Petri net structure consists of a single transition with a multiple inputs and multiple outputs (Figure 5.8). In order to assess the execution time of this structure, a large number of input places and output places has been created, the input places have been marked, and the execution time for the execution of the Petri net (i.e. the firing of the single

transition) has been measured. Table 5.5 shows the average execution times with respect to the number of input and output places, as well as the time required to remove all Petri net markings after the execution of the entire Petri net. The removal of remaining markings after each execution cycle is required, in order to reset the Petri net to its original state before the execution of a new decision making cycle.

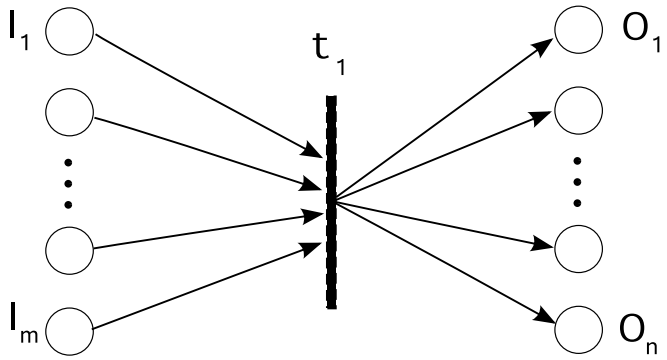


Figure 5.8: Petri net with a single transition with multiple inputs and multiple outputs.

Multiple Transitions with Single Input and Single Output

The second measured Petri net structure consists of multiple transitions, each with a single input and single output (Figure 5.9). In order to assess the execution time of this structure, a large number of transitions has been created, each connected by a single input and single output place, the Petri net's input place has been marked, and the execution time for the execution of the Petri net (i.e. the firing of all transitions) has been measured. Table 5.6 shows the average execution times with respect to the number of transitions. Since the execution times for the removal of remaining markings was below timing measurement limits (less than 1ms) for the listed number of places, these values have been omitted.

The significantly longer and rapidly increasing execution times for a Petri net with multiple transitions (Table 5.6) is due to the fact that in each Petri net execution cycle all transitions need to be checked whether they are enabled

Table 5.5: Measured processing time in $[ms]$ for the execution of a Petri net with a single transition with multiple inputs and multiple outputs (Figure 5.8). The measured execution times are average values for multiple (1000 - 10000) executions. The resetting times are measured execution times for the removal of all markings from the Petri net. Values of 0 indicate that the execution time was below measurable limits of 1ms. (Notebook CPU: Intel T5450, 1.66GHz).

#Inputs	**#Outputs**	**Exec. Time [ms]**	**Resetting Time [ms]**
100	20	0.4	1
1000	20	3	2
5000	20	16	2
10000	20	31	20
100	100	0.5	1
1000	100	4	1
5000	100	16	2
10000	100	32	31
100	1000	2	0
1000	1000	5	0
5000	1000	18	8
10000	1000	34	36

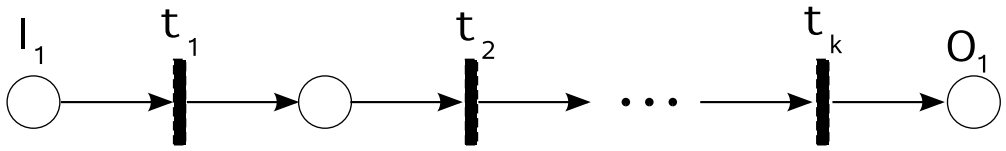


Figure 5.9: Petri net with multiple transitions with a single input and a single output.

or not. Enabled transitions fire by removing a marking from their input place and placing it into their output place, which in turn enables other following transitions to fire. The search for enabled transitions and their firing is executed until there are no more enabled transitions, which results in increased processing times.

The measured execution time values listed in tables 5.5 and 5.6 can be used to estimate the maximum required processing times for a decision making Petri net in a worst case scenario.

Table 5.6: Measured processing time in $[ms]$ for the execution of a Petri net with multiple transitions, each with a single input and single output. The measured execution times are average values from 1000-10000 executions (CPU: Intel T5450, 1.66GHz).

#Transitions	**Exec. Time [ms]**
10	5
20	7
50	38
100	78
200	296
500	1600
1000	6300
2000	25000
5000	160000

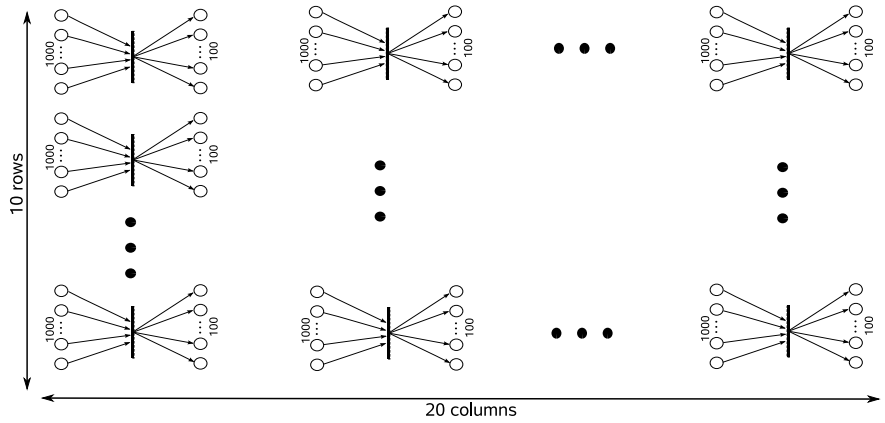


Figure 5.10: Example for the estimation of the Petri net execution time in a worst case scenario (Example 5.7.1). The Petri net consists of 10 rows with each 20 columns of transitions with each 1000 input and 100 output places. Therefore, this net is able to model the input of 10000 World Model Events and Path Planner indications, and the output of 1000 driving maneuvers.

Example 5.7.1. In this example, in order to estimate the execution time in a worst case, it is assumed that the Petri net consists entirely of transitions with each 1000 input and 100 output places (Figure 5.10). According to Table 5.5, the firing of each transition of this type requires in average 4ms. If for example the entire Petri net consists of 10 rows with each 20 consecutive

transitions (i.e. 200 transitions in total), its execution will require on this specific hardware in a worst case $200 * 4ms = 800ms$. Additionally, the time required to reset the net (i.e. to clear all markings) needs to be added, which is around 1ms for each transition. Therefore, the execution time for the entire Petri net on this specific CPU is: $800ms + 1ms * 200 = 1000ms$

Additionally to the execution time measurements for the execution of the first decision making stage, both decision making stages have been integrated and tested as a subsystem of a prototype control software for autonomous vehicles, for both 3D simulator and a real vehicle. The execution time for the Petri net used in the experimental tests consisting of and 16 transitions and 21 places required on the same CPU in average 0.5ms.

Although the implemented prototype system did not include all required aspects for an autonomous vehicle to be fully operational in real-world traffic, the conducted tests indicate that the developed approach is suitable to fulfill real-time requirements. So far, the developed software has been successfully tested on a Windows Vista notebook PC with an Intel T5450 CPU at 1.66GHz with satisfactory results. On this low-end notebook CPU and general purpose (non-realtime) operating system, the decision making process was executed at 1-2 Hz, concurrently to all other vehicle control tasks on the same CPU (at a vehicle speed of max. 1m/s). The low CPU load of 30-40% during the execution of the entire control software indicated that the computing power, especially with respect to ensuring the real-time execution of the decision making subsystem was sufficient (i.e. the CPU was not overloaded).

5.8 Error Recovery

Due to quickly and unexpectedly changing traffic conditions, in some situations, the decision making subsystem may need to abort the execution of certain driving maneuvers, such as overtaking, and to switch to the execution of driving maneuvers for error recovery, as addressed in section 6.3.2. Since driving maneuvers for error recovery are identical to normal driving

maneuvers, this process does not require any changes of the decision making approach. When there is a need to perform an error recovery driving maneuver, the decision making process is the same as that for executing any other driving maneuver.

However, the main challenge for the near future, and a so far not addressed question, will be to develop a system specification for autonomous driving in urban traffic conditions, which foresees and includes how to deal with unexpected traffic conditions which may occur in real-world urban traffic conditions.

5.9 Implementation

A prototype implementation of the presented solution has been realized in C++ using the Microsoft Visual Studio .NET compiler, for the Windows (XP and Vista) operating system. The algorithm for decision making (class DecisionMaker) is executed concurrently in a multithreading model with the World Model, the Driving Maneuvers, and a large variety of other tasks (e.g. CAN bus, sensors, GUI, etc.). In order to ensure an identical programming interface, all driving maneuver are derived from the abstract base class AbstractDrivingManeuver (Figure 5.11).

The connection between the World Model and the class DecisionMaker is based on the Observer design pattern (section 4.6.3), where the class DecisionMaker plays the role of the observer. After registering as an observer, the World Model actively notifies the DecisionMaker about World Model Events (Figure 5.12).

In order to enable the quick starting and stopping of driving maneuvers, all driving maneuver threads are created and initialized by the DecisionMaker at the same time, however remain in a waiting state until the decision making process activates their execution (Figure 5.12). Before starting the execution of a driving maneuver, the decision making process ensures that any other active driving maneuver thread has been put in a waiting state (*stop()* method call), and therefore does not send any driving commands to the vehicle.

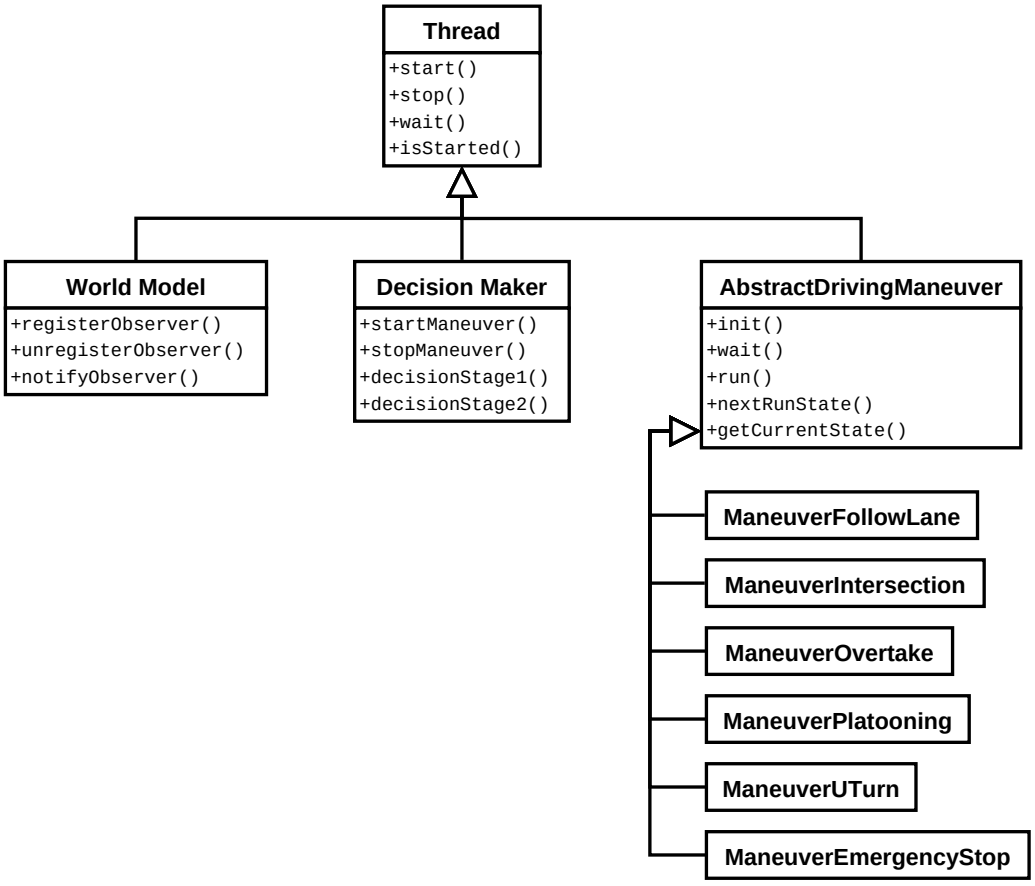


Figure 5.11: UML class diagram of the Decision Making, World Model, and Driving Maneuver classes. All classes which require concurrent execution are derived from the base class Thread. The base class AbstractDrivingManeuver ensure an identical programming interface for all driving maneuvers.

The World Model, the DecisionMaker, and Driving Maneuver threads are executed at their own frequency, with minimum interaction between them. This reduces the danger of thread interlocking[10].

The selection of feasible driving maneuvers is the first stage of the decision making subsystem. It consists of a Petri net, which models the selection of driving maneuvers depending on World Model Events and Route Planner indication.

[10]Thread interlocking occurs when two or more threads wait for each other to finish or to release an exclusively accessible, shared resource. In such a situation both threads are blocked.

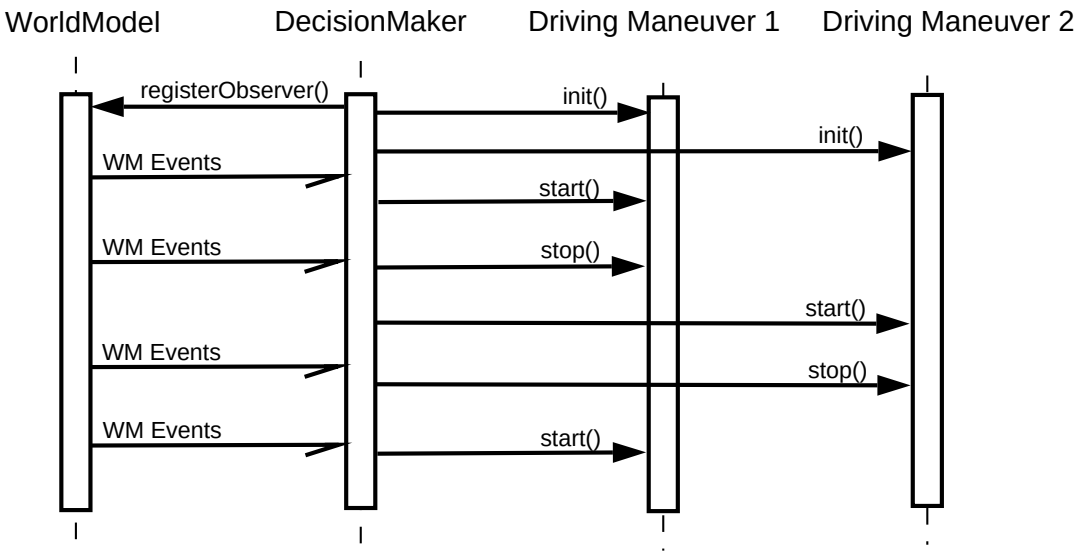


Figure 5.12: UML interaction diagram of the Decision Making, World Model, and two driving maneuver threads. In order to ensure a quick restarting the *stop()* method call puts the thread into a waiting state, while the thread remains active.

In order to decouple the decision making logic from the implementation, the Petri net structure is loaded from an XML (Extensible Markup Language) file into an object-oriented Petri net structure, which is implemented in C++. Figure 5.13 shows an example of a Petri net stored in an XML file. Figure 5.14 shows the UML class diagram of the Petri net implementation.

Having the Petri net structure in an external XML file brings a variety of benefits, such as the possibility to make changes to the Petri net structure without the need to make source code changes and to recompile the control software. As the implementation can load and execute any Petri net structure, the implementation can remain unchanged as soon as it has been proven to operate correctly. Consequently, in order to change the decision making logic, changing and reloading the external XML file is sufficient.

Furthermore, this approach enables the design, simulation, analysis and verification of an eventually very complex Petri net structure using already available Petri net analysis tools [65]. For instance, such tools can verify whether certain output places are reachable, and they can reveal design er-

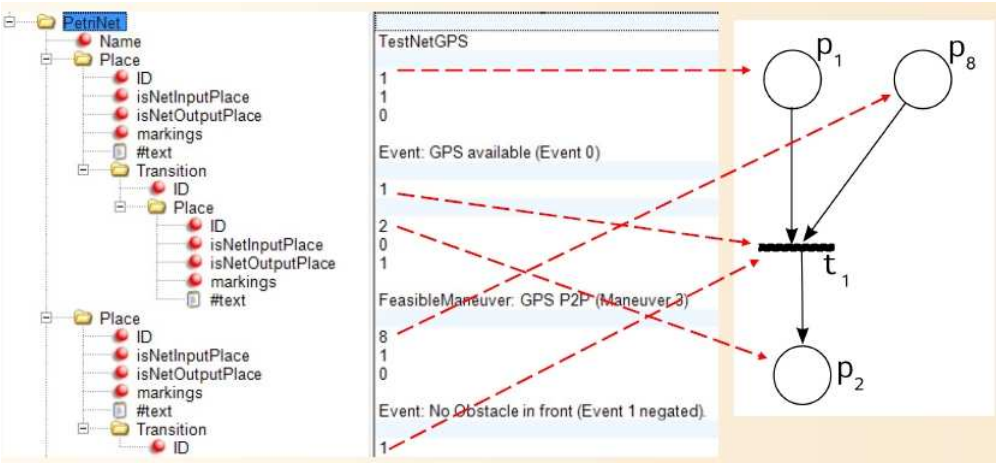


Figure 5.13: Example of a Petri net structure stored as XML file.

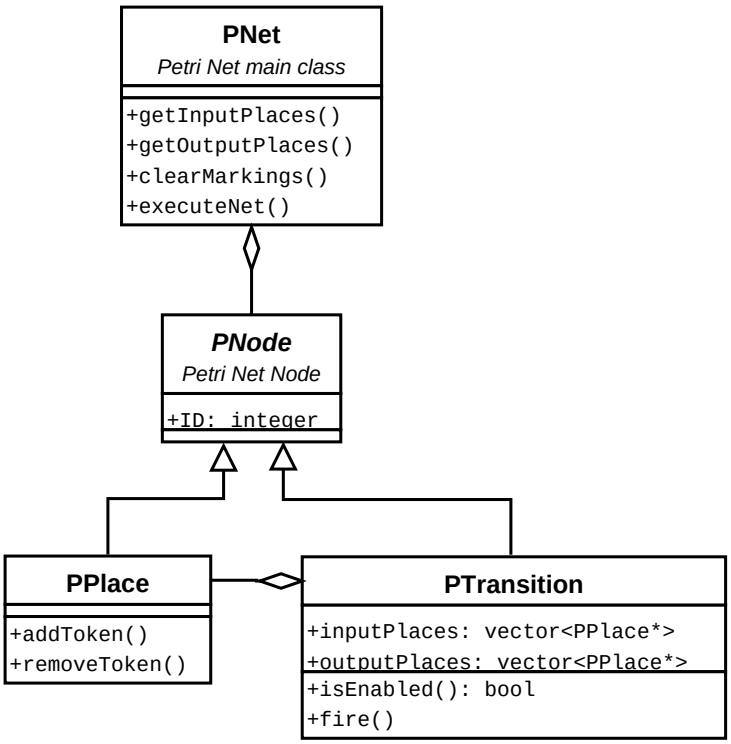


Figure 5.14: UML class diagram of the Petri net implementation.

rors, which would lead to the erroneous selection of certain driving maneuvers.

After loading the Petri net structure from the XML file, the decision

making subsystem is executed cyclicly in its own execution thread. Each execution cycle consists of the following three steps:

1. Mark the Petri net's input places according to World Model Events and Route Planner indication,

2. Execute the Petri net,

3. Obtain the Petri net's marked output places, which represent the feasible driving maneuvers.

As the number of Petri net places and transitions is constant and does not include cycles, the execution of the Petri net, and therefore the execution time of this decision making stage can be analyzed in order ensure the real-time performance of this decision making stage.

The second decision making stage implements the MCDM algorithm elaborated in section 5.6 based on the pseudocode listed in Algorithm 5.2. Algorithm 5.3 lists a simplified pseudocode of the entire decision making process.

Algorithm 5.1 calculateFeasibleDrivingManeuvers

Require: array of World Model Events
Require: path planner direction

```
1: clear Petri net places
2: mark Petri net with World Model Events
3: mark Petri net with path planner direction
4: feasibleManeuvers ← Petri net output markings
5: return feasibleManeuvers
```

Algorithm 5.2 calculateMostAppropriateDrivingManeuver

Require: feasible driving maneuvers (algorithm 5.1)
Require: MCDM utility functions $f_j(a_i)$, attribute weights w_j

1: feasibleManeuvers ← calculateFeasibleDrivingManeuvers
2: **for** each feasibleManeuver a_i **do**
3: calculate value function $V(a_i) := \sum_{j=1}^{p} w_j f_j(a_i)$ (equation 5.1)
4: **end for**
5: maximumValueManeuver ← $max(V(a_i))$
6: **return** maximumValueManeuver

Algorithm 5.3 drive to destination (make driving decisions)

Require: array of World Model Events
Require: path planner direction

1: initialize driving maneuver threads
2: register as observer of the World Model
3: {thread loop}
4: **while** destination not reached **do**
5: get World Model Events
6: get path planner direction
7: calculateFeasibleDrivingManeuvers (Algorithm 5.1)
8: mostAppropriateManeuver ← calculateMostAppropriateDrivingManeuver (Algorithm 5.2)
9: **if** activeManeuver != mostAppropriateManeuver **then**
10: stop activeManeuver
11: start mostAppropriateManeuver
12: **end if**
13: **end while**
14: stop all driving maneuvers (stops vehicle).

5.10 Discussion

This chapter has addressed and presented a solution for the problem of Real-Time Decision Making for autonomous city vehicles, including the system specification, elaboration of requirements, ensuring real-time performance, and error recovery due to unexpectedly changing traffic conditions. Figure 5.15 shows an overview on the operational dynamics, including the World Model and the Real-Time Decision Making subsystem.

The decision making process is based on Petri nets and Multiple Criteria Decision Making (MCDM). The decision making process has been divided into two consecutive stages. While the first decision making stage (DMU1) is safety-critical and focuses on selecting the feasible and safe driving maneuvers, the second decision making stage (DMU2) focuses on non safety-critical driving objectives, such as improving comfort and efficiency.

The main reason for the use of a Petri net model for this decision making stage is the Petri net's suitability to model, analyze, and verify the correctness of a very complex operational behavior of the first, safety-critical decision making stage[11]. In addition to a large number of inputs in the form of World Model events (i.e. Petri net input places), this first decision making stage requires the capacity to integrate a complex logic, which makes decisions among a large number of output alternatives, i.e. the feasible driving maneuvers. Therefore, since a Petri net model fulfills these requirements, the developed approach allows the decision making subsystem to deal with very complex real-world traffic situations.

Furthermore, since the Petri net structure is decoupled from the control software implementation, and loaded from an external XML file, only its execution is implemented in the vehicle decision making & control software source code. Consequently, changes of the decision making operational behavior (in the Petri net XML file) do not require changes of the source code, and this in turn minimizes the possibility to introduce new software errors.

The application of MCDM methods for the second decision making stage

[11] Another feature of Petri nets is the capacity to model parallel execution, however this is not applied in this case.

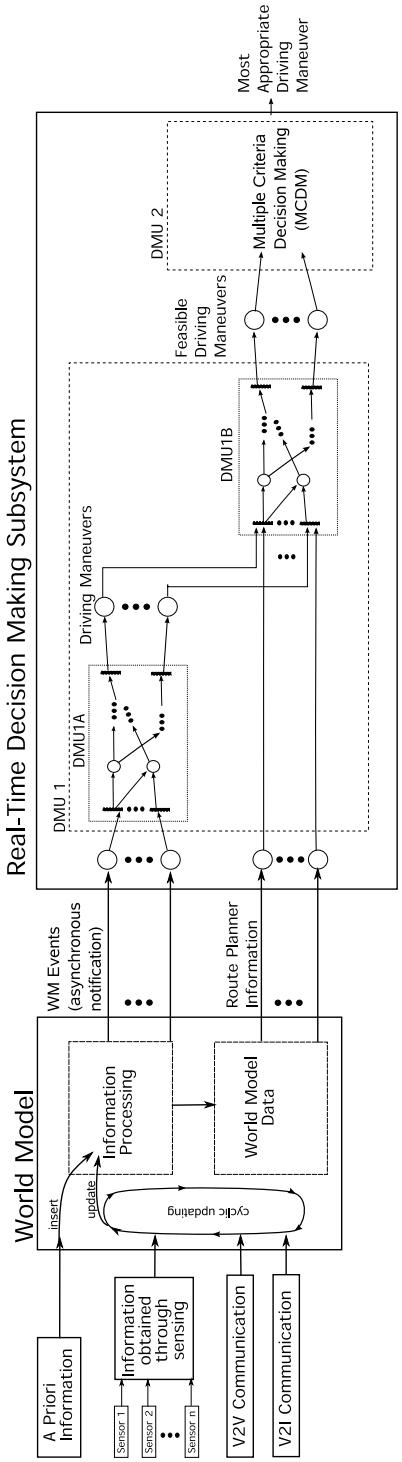


Figure 5.15: Overview on the World Model's and the Real-Time Decision Making Subsystem's operational dynamics.

enables the consideration of a large number of driving objectives, including possibly conflicting ones, and leads to a powerful and flexible solution for non-simplified urban traffic conditions. Furthermore, compared to so far existing solutions, which were however intended only for simplified traffic conditions, the application of MCDM in this new research area offers a variety of benefits with respect to the problem specification, decision flexibility, and scalability.

Multicriteria Decision Making methods are established mathematical tools which are widely applied in a large variety of engineering and science fields with complex decision problems [46, 70]. For our problem of autonomous driving, MCDM offers a variety of benefits:

- The hierarchy of objectives allows a systematic and complete specification of goals to be achieved by the vehicle.
- The utility functions can be defined heuristically to reflect the choices of a human driver, or alternatively, learning algorithms can be applied.
- MCDM allows the integration and evaluation of a very large number of driving alternatives.
- Decision flexibility can be achieved by defining the set of attribute weights depending on the road conditions.
- Additional objectives, attributes, and alternatives can be added without the need of major changes, and therefore enables a very scalable solution.

However, since the method is highly based on heuristics (i.e. heuristic definition of objectives, utility functions, and attributes), if MCDM alone is used (i.e. by omitting the first decision making stage), it is difficult the ensure that all made decisions will always lead to safe driving. This problem has been addressed, by ensuring that the MCDM process only selects the most appropriate driving maneuver from the set of feasible driving alternatives, which is the outcome of the first decision stage.

As already mentioned in this chapter's introduction, it is assumed that the autonomous vehicle's driving decisions need to be made in real-time and

cannot be planned in advance. The decision making approach presented in this thesis focuses only on the current road traffic conditions, and does not include the consideration of future traffic conditions or predictive planning of future driving decisions. Furthermore, since future predictions about the traffic environment cannot be considered to be certain, it is difficult to justify that such considerations would contribute towards improving the safety of such vehicles.

While this chapter has elaborated on the Real-Time Decision Making subsystem, the following Chapter 6 focuses on its output, the driving maneuvers and their design concept.

Chapter 6

The Driving Maneuver Design Concept

6.1 Introduction

The highly complex task of autonomous driving in non-simplified urban traffic conditions requires the subdivision into subtasks with lower, and therefore manageable complexities. While this obvious idea of dividing the whole driving problem into manageable subtasks has been widely accepted and adopted by many research groups [9, 71, 72], the published material of the proposed approaches is often superficial, with few detailed elaborations. Furthermore, a wide variety of terminologies such as "behaviors", "actions", "navigation modules", etc., are often used to express similar ideas, which makes their direct comparison more difficult.

This chapter elaborates in detail on the subdivision of the entire driving task into subtasks, called driving maneuvers, which represent the output of the Real-Time Decision Making process (Figure 3.2). The division into several subtasks is based on the required feedback information about the vehicle's surrounding environment. Each driving maneuver uses the minimum amount of feedback information necessary to safely maneuver the vehicle over a certain distance or time.

6.2 Requirements

The main objective for driving maneuvers is to implement control algorithms which drive the vehicle in specific traffic situations over a defined distance, or time. In order to do this, a sequence of complex closed-loop control tasks needs to be executed concurrently[1] with the other tasks of the autonomous vehicle's decision making & control system. Furthermore, since their execution is started and stopped by the decision making subsystem, a common architecture and interface for all driving maneuvers is necessary. The following subsections address these functional and non functional requirements in detail.

6.2.1 Functional Requirements

The following are the most relevant functional requirements for driving maneuvers:

- Enable execution in real-time.
- Wait and respond to start/stop signals from the decision making subsystem.
- Begin and stop the execution "immediately" (i.e. within real-time limits) as soon as the start/stop signals have been received from the decision making subsystem.
- Enable their execution by sending driving commands (e.g. speed and steering angle) to the vehicle interface.
- Enable the smooth and immediate switch of execution between driving maneuvers.
- Send feedback information about the execution success or failure to the decision making subsystem.
- Restart the execution when requested by the decision making subsystem.

6.2.2 Non Functional Requirements

The following are the most relevant non functional requirements for driving maneuvers:

[1] The prototype system has been implemented using a multithreading model on a single CPU.

- Modularity: enable the specification, design, implementation and testing of each phase of a driving maneuver in particular.
- Enable arbitrarily complex sequences of driving maneuver phases.
- Common and generic structure for all driving maneuvers, regardless of their complexity with respect to the number of driving maneuver phases.

6.3 Modeling of Driving Maneuvers

Driving maneuvers are implementations of closed-loop control algorithms, each capable of maneuvering the autonomous vehicle in a specific traffic situation. In line with the specified requirements, the driving maneuvers are based on a common general structure. Their operational behavior is designed via *deterministic finite automata*[2] [59] which includes (Figure 6.1):

- a *start state* q_0,
- two final states $\{q_F, q_E\} = F$,
- a set of *Run states* $Q^{run} = \{q_1^r, q_2^r, ..., q_n^r\} \subset Q$,
- a set of input symbols Σ, which consists of at least the symbols: $Run, Stop, Restart, Error$,
- the state transition function $\delta : Q \times \Sigma \rightarrow Q$, which defines the automaton's transitions from state to state.

The *start state* q_0 is the *waiting* or *idle* state, in which the automaton is waiting for the *Run* signal from the Real-Time Control & Decision Making subsystem.

The Run states $q_1^r, q_2^r, ..., q_n^r$ perform the maneuvering of the vehicle. Each of them includes checking of necessary preconditions, such as the availability of world model information and road safety conditions. As long as the preconditions are met, the Run states execute closed-loop control algorithms.

[2] i.e. deterministic finite state machines

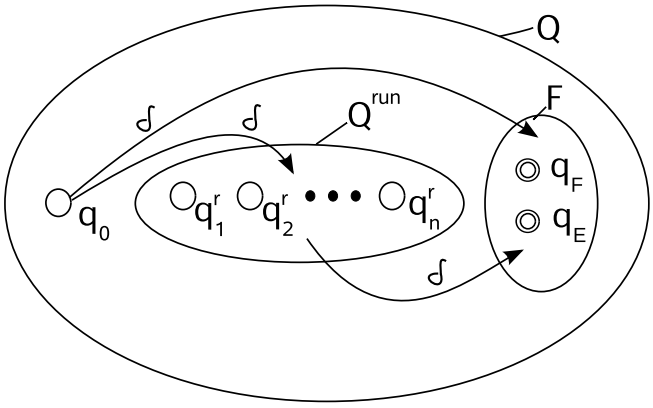


Figure 6.1: Sketch of the driving maneuver automaton structure.

Otherwise, if certain preconditions are not met, the *Error* symbol is generated, and the automaton changes into the error state q_E, which terminates the execution of driving maneuver.

A driving maneuver finishes in one of the *final states* q_F (finished) or q_E (error). The final state q_F represents a successful completion of the driving maneuver, while the error state q_E signals that the driving maneuver has been aborted due to an error or some other reason. The information regarding the final state of a driving maneuver is used by the Real-Time Control & Decision Making subsystem.

Figure 6.2 shows a driving maneuver automaton, consisting of the minimum possible number of states and input symbols. The set of input symbols is $\Sigma = \{Run, Stop, Restart, Error\}$ and the input symbols have the following meaning:

- *Run*: request to begin the execution of the driving maneuver,
- *Stop*: request to stop the execution of the driving maneuver,
- *Restart*: request to restart the driving maneuver (i.e. to reset the automaton to its start state q_0),
- *Error*: some error occurred, which makes the continued execution of the driving maneuver impossible. The error can be for example due to one or more missing parameters necessary for the execution of the

driving maneuver, or due to unfulfilled preconditions which are required for the successful execution of the driving maneuver.

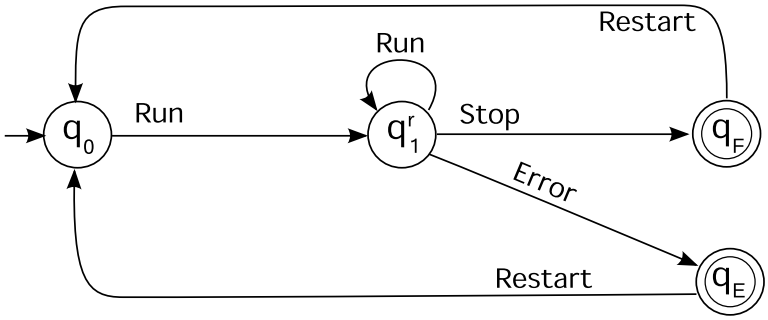


Figure 6.2: A driving maneuver finite automaton with only one Run state q_1^r.

Table 6.1: Tabular representation of the transition function $\delta : Q \times \Sigma \to Q$ of the driving maneuver finite automaton with only one Run state (Figure 6.2). The rows represent states, the columns represent input symbols. The start state and final states are marked with a right arrow ($\to$) and a star ($*$), respectively. The first column indicates the current state, while the following columns indicate the state into which the automaton switches when it receives the corresponding input symbol (Run, Error, Stop, or Restart).

State	Run	Error	Stop	Restart
$\to q_0$	q_1^r	q_0	q_0	q_0
q_1^r	q_1^r	q_E	q_F	q_0
$*q_F$	q_F	q_F	q_F	q_0
$*q_E$	q_E	q_E	q_E	q_0

The automaton in figure 6.2 starts in the state q_0. In this state the automaton waits for the input symbol *Run*, which is the request from the Real-Time Control & Decision Making subsystem to begin the execution of the driving maneuver in the Run state q_1^r. The automaton remains in the state q_1^r and executes the driving maneuver as long as the input symbol *Run* is received. A request from the Real-Time Control & Decision Making subsystem to stop the driving maneuver is expressed by the input symbol

Stop, which leads to a state transition into the final state q_F (finished). The state q_F signals a successful completion of the driving maneuver.

The state transition from the Run state q_1^r into the final state q_E (error) occurs in the case when parameters required to maneuver the vehicle are missing (i.e. world model data needed to maneuver the vehicle is not available), or certain safety relevant conditions are no met (e.g. oncoming vehicle while overtaking). Once one of the final states q_F or q_E has been reached, the driving maneuver automaton can be reset to its start state q_0 by the Real-Time Control & Decision Making subsystem using the input symbol *Restart*.

While the input symbols *Run*, *Stop* and *Restart* are generated by the Real-Time Control & Decision Making subsystem, the input symbol *Error* is generated by the automaton itself based on information received from the World Model, when the driving maneuver cannot be continued due to certain road traffic conditions. For example, when another vehicle is oncoming during overtaking, the overtaking maneuver automaton creates the *Error* symbol and switches into its Error state. Table 6.1 shows the complete state transition table for the finite deterministic automaton shown in figure 6.2.

Definition 6.3.1. *Driving Maneuver.*
A driving maneuver is an algorithm which implements a deterministic finite automaton $D = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$, of the form:

- $Q = \{q_0\} \cup Q^{run} \cup F$,
- $F = \{q_E, q_F\}$ (the final states),
- the set of input symbols $\Sigma = \{Run, Stop, Restart, Error, Next\_Phase\}$,
- the state transition function $\delta : Q \times \Sigma \rightarrow Q$ is of the form as defined in table 6.2,
- each $q_i^r \in Q^{run}$ implements a closed-loop control algorithm.

The algorithm's parameter is a driving maneuver parameter vector $\overrightarrow{p}$ containing the reference values for the Run states (definition 6.3.2).

Example 6.3.1. A road lane following driving maneuver can be modeled using the automaton shown in figure 6.2. In its start state q_0 the driving maneuver automaton waits for the input *Run*, which represents the request to begin the execution of the driving maneuver. The input symbol *Run*, which is generated by the Real-Time Control & Decision Making subsystem, changes the automaton's state into the Run state q_1^r.

In the Run state the automaton checks whether information about the lane marking position is available. If this information is missing, the input symbol *Error* is generated and the driving maneuver changes into the error state q_E. Otherwise, the automaton remains the Run state q_1^r, which is executed continuously. The Real-Time Control & Decision Making subsystem can stop a driving maneuver any time using the input symbol *Stop* (Figures 3.2, 5.12).

The input symbol *Stop* leads to a successful termination of the driving maneuver by taking the automaton from the state q_1^r to the final state q_F. However, if the lane marking position is not available anymore while the automaton is in q_1^r, the input symbol *Error* is generated, which induces a state transition into the final state q_E. The state q_E stops the driving maneuver in its Error state, which signals an error. The driving maneuver can be reset by the Real-Time Control & Decision Making subsystem to its start state q_0 using the input symbol *Restart*.

The simple deterministic finite automaton shown in figure 6.2 is however not sufficient to model a more complex driving maneuver, such as overtaking (Example 4.3.1). Assuming that any complex driving maneuver is dividable into a set of consecutive maneuver phases, each being modeled by its own Run state, the finite automaton in figure 6.2 can be expanded to fit these requirements by adding an additional input symbol *Next_Phase* and by increasing the number of Run states. The resulting new automaton is shown in figure 6.3, its state transition function is shown in table 6.2.

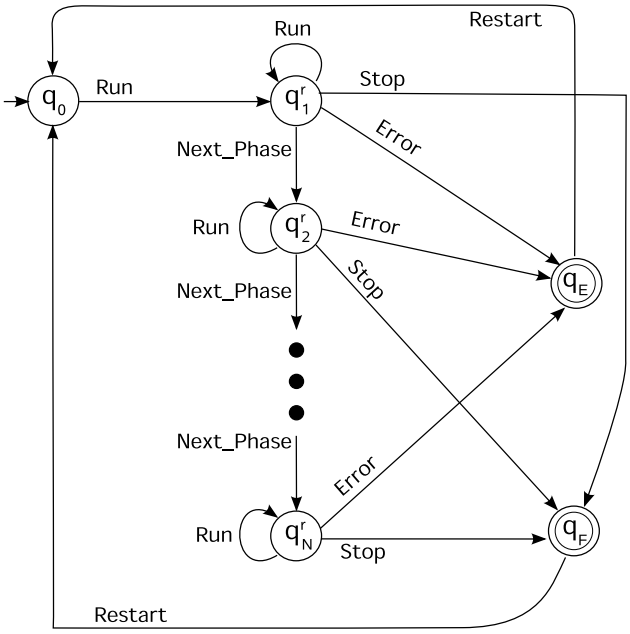


Figure 6.3: A driving maneuver finite automaton with multiple Run states. The number of Run states $q_i^r, i = 1, 2, ...N$ equals the number of driving maneuver phases. The arrow into the q_0 indicates the automaton's start state.

Table 6.2: Tabular representation of the transition function $\delta : Q \times \Sigma \rightarrow Q$ of the driving maneuver finite automaton with multiple Run states (Figure 6.3). The rows represent states, the columns represent inputs. The start state and final states are marked with an $\rightarrow$ and $*$, respectively.

State	Run	Error	Next_Phase	Stop	Restart
$\rightarrow q_0$	q_1^r	q_0	q_1	q_0	q_0
q_1^r	q_1^r	q_E	q_2^r	q_F	q_0
q_2^r	q_2^r	q_E	q_3^r	q_F	q_0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
q_i^r	q_i^r	q_E	q_{i+1}^r	q_F	q_0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
q_N^r	q_N^r	q_E	q_F	q_F	q_0
$*q_F$	q_F	q_F	q_F	q_F	q_0
$*q_E$	q_E	q_E	q_E	q_E	q_0

Example 6.3.2. The overtaking driving maneuver (initially introduced in example 4.3.1) can be modeled using the finite automaton of the form shown in figure 6.3, however with five Run states $q_1^r, q_2^r, ..., q_5^r$ (Figure 6.5). Each of the five Run states represents a driving maneuver phase. The driving maneuver is divided into the following five phases, assuming overtaking on the right hand side (Figure 6.4):

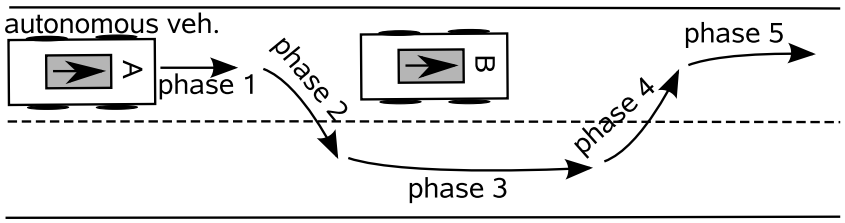


Figure 6.4: The overtaking maneuver is decomposed into five phases. In the finite automaton each phase is represented by a Run state.

Phase 1 (q_1^r): approach the vehicle in front,

Phase 2 (q_2^r): change onto the right lane,

Phase 3 (q_3^r): drive in parallel to the vehicle to be overtaken and pass,

Phase 4 (q_4^r): change back onto the left lane,

Phase 5 (q_5^r): drive in front of the overtaken vehicle.

Example 6.3.3. A driving maneuver for intersection crossing (initially introduced in example 4.3.2) can be modeled with four Run states $q_1^r, q_2^r, q_3^r, q_4^r$ (Figure 6.7), where each of the four Run states represents a driving maneuver phase (Figure 6.6):

Phase 1 (q_1^r): approach the intersection, reducing speed,

Phase 2 (q_2^r): give way,

Phase 3 (q_3^r): cross the intersection,

Phase 4 (q_4^r): continue following the lane.

Table 6.3: Tabular representation of the transition function $\delta : Q \times \Sigma \to Q$ of the overtaking driving maneuver in Example 6.3.2

State	Run	Error	Next_Phase	Stop	Restart
$\rightarrow q_0$	q_1^r	q_0	q_1	q_0	q_0
q_1^r	q_1^r	q_E	q_2^r	q_F	q_0
q_2^r	q_2^r	q_E	q_3^r	q_F	q_0
q_3^r	q_3^r	q_E	q_4^r	q_F	q_0
q_4^r	q_4^r	q_E	q_5^r	q_F	q_0
q_5^r	q_5^r	q_E	q_F^r	q_F	q_0
$*q_F$	q_F	q_F	q_F	q_F	q_0
$*q_E$	q_E	q_E	q_E	q_E	q_0

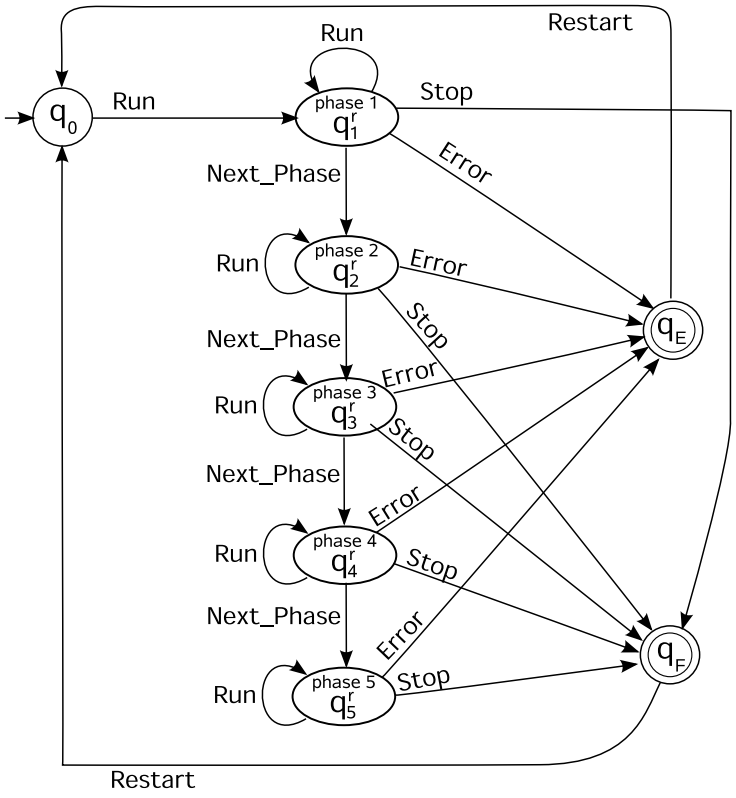


Figure 6.5: The finite automaton for the overtaking driving maneuver in Example 6.3.2.

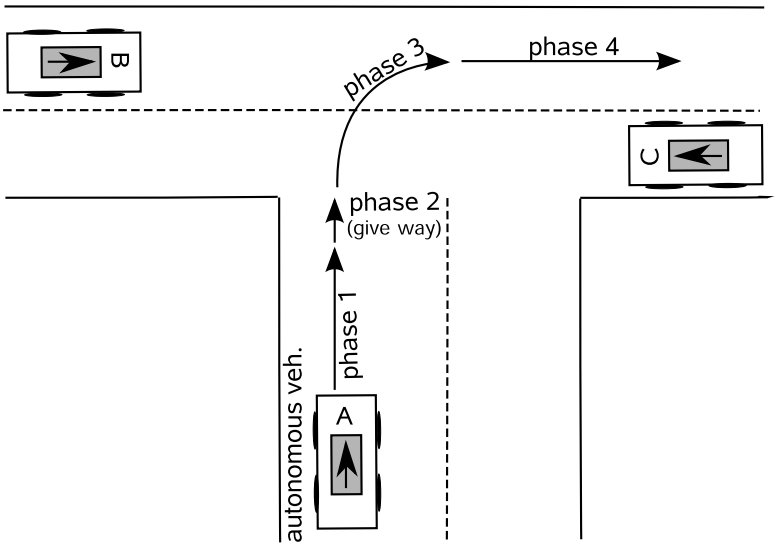


Figure 6.6: The intersection crossing driving maneuver is decomposed into four phases, each represented by a Run state (assuming driving on the left hand side).

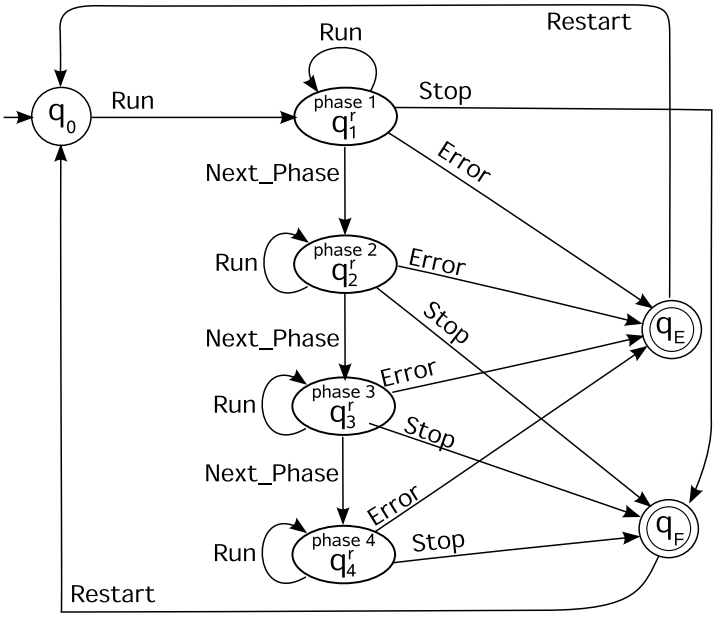


Figure 6.7: The finite automaton for the intersection crossing driving maneuver in Example 6.3.3.

6.3.1 Driving Maneuver Reference Values

Definition 6.3.2. *Parameter Vector $r \in \mathbb{R}^n$ of a Driving Maneuver.*
A driving maneuver parameter vector is a n-dimensional vector of real numbers, containing all required reference (setpoint) values for the closed-loop control algorithms implemented in the Run states Q^{run}. As r is the reference (setpoint) vector, the dimension of r equals the dimension of the controlled variables vector $\overrightarrow{y}$ $(n = dim(r) = dim(y))$.

The purpose of the parameter vector r is to allow the setting of setpoint (reference) values for the execution of the closed-loop control algorithms of driving maneuvers. Examples for such setpoints are for instance the vehicle speed, at which a driving maneuver is to be performed, or the distances to other vehicles.

Consequently, the parameter vector enables the execution of driving maneuvers with different characteristics, offering the Real-Time Decision Making subsystem the possibility to choose among various execution alternatives for each driving maneuver. This is further explained in section 5.6.

6.3.2 Implementation of Driving Maneuvers for Error Recovery

During the execution of a driving maneuver, the traffic situations can change unexpectedly and in an unforeseeable way. For example, during overtaking another oncoming vehicle might appear on the overtaking lane. Similar to a human driver, in such a situation, the decision making subsystem needs to reassess the situation, if necessary abort the currently performed driving maneuvers, and make appropriate decisions with respect to the new traffic situation.

Due to its continuous execution, the decision making process is able to abort the execution of a driving maneuver and switch to the execution of another driving maneuver for error recovery, which can be specifically designed for recovering from unexpected situations (e.g. abort overtaking and clear the oncoming lane).

Since the principle of driving maneuvers for error recovery does not differ from any other driving maneuvers, their specification, model, and implementation process is identical. The only aspect which differs them from usual driving maneuvers are the traffic conditions, in which they are to be activated and performed, as defined by the system specification.

Therefore, the presented generic driving maneuver concept is very likely suitable for the implementation of maneuvers for error recovery from driving decision errors due to unexpected changes of the traffic conditions. Nevertheless, since the implementation of driving maneuvers would go beyond the scope of this thesis, such driving maneuvers for error recovery have not been implemented.

6.4 Discussion

After elaborating the functional and non functional requirements, this chapter has presented the developed driving maneuver concept, which enables the subdivision of the complex task of autonomous driving into subtasks with lower complexities.

The developed model, based on deterministic finite automata, allows the decomposition of complex driving maneuvers into phases. Each driving maneuver phase is represented by a finite automaton Run state, which contains the closed-loop control algorithms required to maneuver the vehicle. This approach enables the application of control theory design tools for each Run state in particular, in order to obtain the desired system performance in terms of stability, robustness, and accuracy [73].

Furthermore, the developed approach allows the development of error recovery driving maneuvers, which need to be performed when the traffic conditions change in an unforeseen way.

While the developed approach facilitates the problem decomposition into subtasks and the implementation of the required closed-loop control algorithms, it is important to note that the development of driving maneuvers needs to be in line with a detailed, complete, and consistent specification of requirements for autonomous driving in urban traffic. A complete system

specification is needed, which covers the large variety of possible urban traffic situations, and defines which driving maneuver to perform.

The following Chapter 7 presents the experimental results obtained in both 3D simulation and real-world experiments.

Chapter 7

Experimental Results

7.1 Introduction

The presented concepts for World Model, Driving Maneuvers, and Decision Making have been implemented and integrated into a single control software application for both a 3D simulator and a real experimental vehicle (Cycab). While the implementations for these three concepts are identical for the simulator and real vehicle, different implementations for the sensor interface and the vehicle interface can be exchanged, according to whether the simulator, or the real vehicle is used. This is realized through abstract sensor and vehicle interfaces, which remain the same for both simulator and real vehicle (Figure 7.1).

7.2 3D Simulation

A 3D simulation environment [61] (Figures 7.2, 7.3) has been used to test the entire autonomous vehicle decision making & control software, focusing mainly on the decision making process. The 3D simulation environment and the vehicle control software are running as two different applications, if necessary on different computers, communicating over a network.

The 3D simulation environment includes the simulation of an autonomous vehicle which can be controlled in the same way as a real autonomous ve-

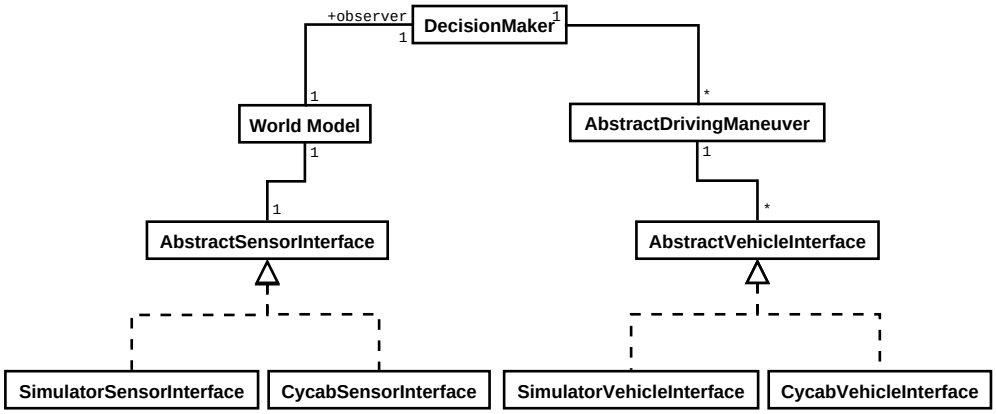


Figure 7.1: Simplified UML class diagram of the control software implementation for both 3D simulator and real experimental vehicle (Cycab).

Figure 7.2: The 3D simulation environment is a model of the test environment for the real autonomous vehicle at Nathan Campus (Griffith University).

hicle (i.e. through speed and steering angle commands), on-board sensor data (e.g. LIDAR, GPS), and static and dynamic obstacles, such as other vehicles, motorcycles, and pedestrians, which move randomly on predefined travel paths. Vehicles and motorcycles move at various speeds on the road,

while pedestrians move at slower speeds on pathways, but can also cross the road. The simulated sensor data is provided in real-time according to the vehicle's traffic environment in the simulation, using identical data structures as real sensor data. Therefore, the vehicle and sensor interfaces of the control software for the simulation are identical to the interfaces of the real experimental vehicle. Furthermore, the simulated traffic environment is a 3D model of the real traffic environment at the test location for the real experimental vehicle at Griffith University, Nathan campus. Consequently, the decision making results obtained through the 3D simulation closely reflect realistic results in real-world road traffic conditions.

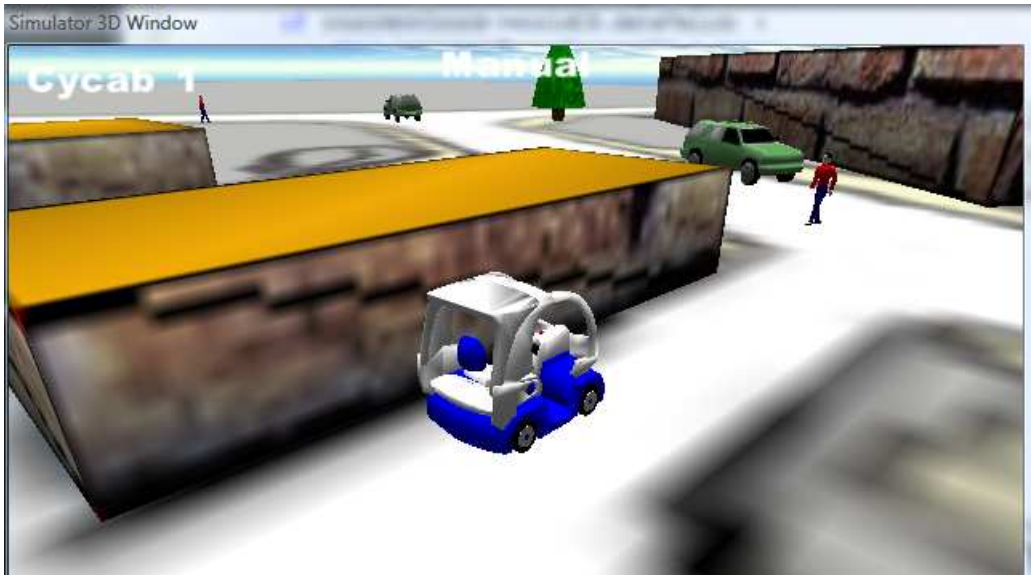


Figure 7.3: The autonomous vehicle in the 3D simulation.

Figure 7.4 shows the GUI (graphical user interface) of the decision making module. When the decision making process is running, the GUI lists the feasible driving maneuvers for the current traffic situation, the most appropriate driving maneuver and, if in automatic driving mode, it marks the driving maneuver which is currently in execution.

Both stages of the decision making process have been tested in two ways: in manual driving mode, and in automatic driving mode. Manual driving mode enables the user to manually maneuver the vehicle, while the deci-

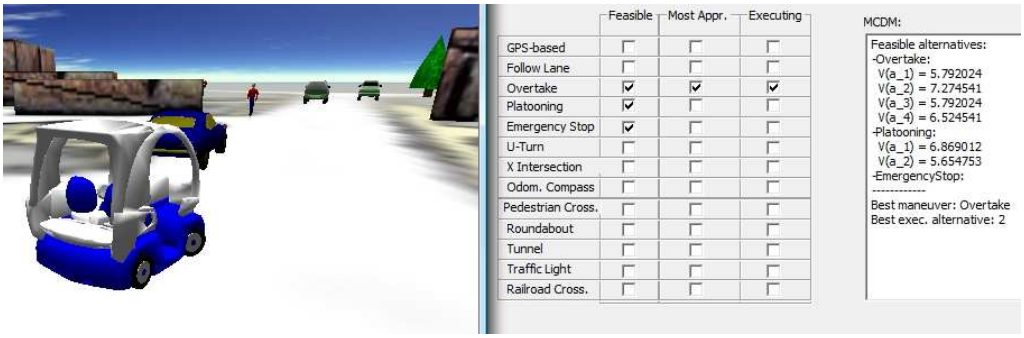


Figure 7.4: 3D simulation environment and the decision making graphical user interface of the autonomous vehicle decision making & control software (screenshot, left-hand side driving).

sion making process is executed simultaneously. Although the vehicle is not performing automatic driving maneuvers in the manual driving mode, the decision making algorithm is running and updating the GUI, which enables the user to supervise the decision making results in real-time. This mode is useful for debugging and testing the decision making algorithm without actually implemented driving maneuvers. In the automatic driving mode, the decision making module not only selects the feasible driving maneuvers and calculates the best driving maneuver execution alternative, but also starts the automatic execution of the most appropriate driving maneuver.

The following 11 attributes $attr_1, .., attr_{11}$ have been used in the experimental tests:

- keep distance to right boundary := $attr_1$
- keep distance to left boundary := $attr_2$
- keep distance to front vehicle := $attr_3$
- keep distance to moving obstacles := $attr_4$
- keep distance to static obstacles := $attr_5$
- keep minimum distance to obstacles := $attr_6$
- drive around obstacles := $attr_7$
- avoid sudden braking := $attr_8$
- avoid quick lane changes := $attr_9$
- maintain minimum speed := $attr_{10}$

- avoid stops := $attr_{11}$

The list of the considered driving maneuver alternatives is as follows:

- a1 = GPS_Point2Point fast
- a2 = GPS_Point2Point medium speed
- a3 = GPS_Point2Point slow
- a4 = Intersection crossing fast
- a5 = Intersection crossing slow
- a6 = Pass slow/small distance
- a7 = Pass slow/large distance
- a8 = Pass fast/small distance
- a9 = Pass fast/large distance
- a10 = Platooning large distance
- a11 = Platooning small distance
- a12 = Emergency Stop

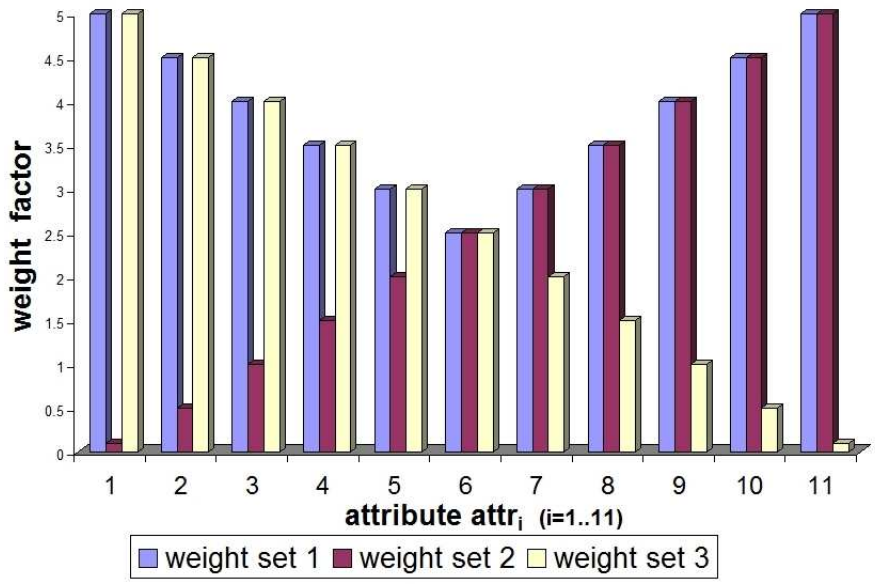


Figure 7.5: MCDM attribute weight distributions applied to the 11 attributes.

In order to test the sensitivity of the developed Decision Stage 2 method, three different attribute weight distribution sets have been applied for each of these 11 attributes, as shown in Figure 7.5. The Weight Set 1 was chosen in such a way that the highest weight factors of 5 is assigned to both attributes 1 and 11, while the weight factor of 2.5 is assigned to attribute 6; the Weight Set 2 assigns increasing weight factors in steps of 0.5, while the Weight Set 3 assigns decreasing weight factors in steps of 0.5.

The outcomes of the first decision making stage are presented in Table 7.1, Column 2, as follows:

- Situation 1: the autonomous vehicle is following a road, relatively far from the coming intersection. In this situation, the first, Petri net based decision making stage has determined the driving maneuver for following a road using GPS coordinates, and its three execution alternatives (i.e. fast, medium speed, slow), as feasible.

- Situation 2: the autonomous vehicle is approaching an intersection, with a road crossing pedestrian. In addition to the road following maneuver, the first decision making determines the intersection crossing maneuver as feasible.

- Situation 3: the autonomous vehicle is following another vehicle. The first decision making stage determines the driving maneuvers "Pass" and "Platooning" as feasible.

- Situation 4: a static obstacle (i.e. a tree) is in front of the autonomous vehicle. The only feasible driving maneuver is the emergency stop.

The second, MCDM-based decision making stage calculates the values $V(a_i)$ for each of the feasible alternatives using Equation 5.1, which provided in this case the following results (Table 7.1, column 3):

- Situation 1: the use of the Weight Set 1 and Weight Set 2 result in the alternative $a1$ to be most appropriate (i.e. maximum value), while the Weight Set 3 results in the alternative $a3$ to be the most appropriate.

- Situation 2: the alternative $a4$ is the most appropriate for weight sets 1 and 2, while alternative $a3$ becomes the most appropriate if the Weight Set 3 is applied.

- Situation 3: the use of the Weight Set 1 results in the decision to execute alternative $a9$, while alternatives $a7$ and $a10$ are chosen for weight sets 2 and 3 respectively.

- Situation 4: the only feasible driving maneuver is the emergency stop. In our implementation the emergency stop is not evaluated by the MCDM based stage, but is instead immediately executed whenever it is determined to be the only feasible alternative.

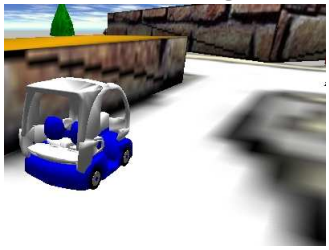
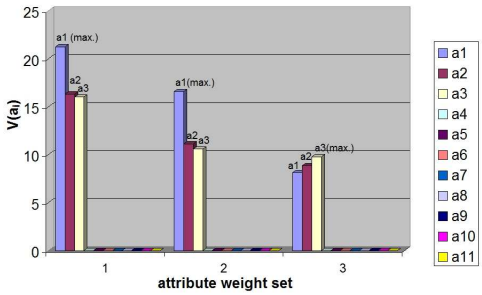
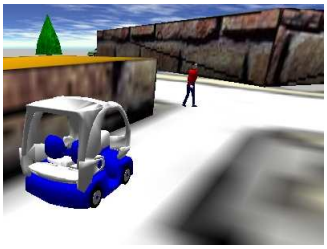
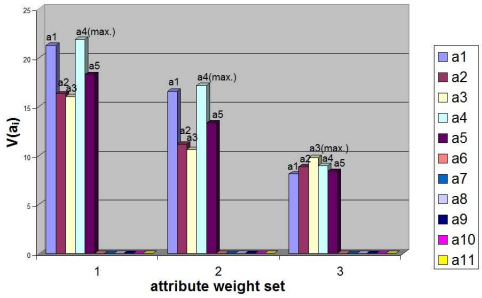
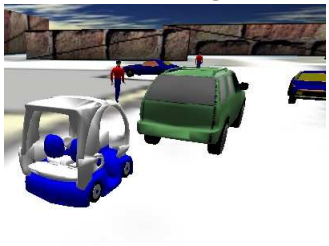
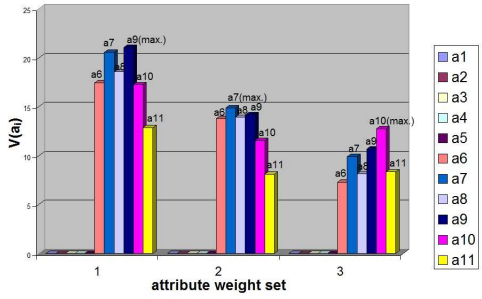
As expected, whenever the World Model fails to provide accurate information, such as for instance information regarding oncoming vehicles (e.g. Table 7.1, Situation 3), the first decision making stage may make the wrong decision about the feasible driving maneuvers. This error is then further passed on to the second stage, and may result in inappropriate or even unsafe driving decisions. Therefore, since it is mainly responsible for safety aspects, the first decision making stage has a crucial impact on the entire decision result.

On the other side, the MCDM results in Table 7.1 show that even when the attributes and utility functions of the second, MCDM based decision making stage have not been specified appropriately, for instance by assigning weights too high to irrelevant attributes, the resulting driving decisions are still safe. For example, applying the first set of attribute weights in Situation 1 results in the decision to approach the intersection at fast speed (i.e. alternative 1), while applying the third attribute weight set results in the more appropriate decision to approach the intersection at a lower speed (i.e. alternative 3).

Consequently, the developed decision making approach delivers correct decision results under the following conditions:

- accurate and sufficient information is provided by the World Model in real-time, especially regarding the MCDM attributes;

Table 7.1: Real-Time Decision Making Results of both decision making stages. Column 2 shows the feasible driving maneuvers, while column 3 shows the result values of the second, MCDM-based stage. There are 11 alternatives in total, but only those which are determined as feasible in the first stage are evaluated in the MCDM based stage. Each of the 11 alternatives are evaluated three times, applying the 3 different weight sets shown in Figure 7.5.

	DM Stage 1 Results	**DM Stage 2 Results**
Sit. 1: Following Road 	a1 = GPS_Point2Point fast a2 = GPS_Point2Point med. a3 = GPS_Point2Point slow	
Sit. 2: Intersection 	a1 = GPS_Point2Point fast a2 = GPS_Point2Point med. a3 = GPS_Point2Point slow a4 = Intersection fast a5 = Intersection slow	
Sit. 3: Following Vehicle 	a6 = Pass slow/small dist. a7 = Pass slow/large dist. a8 = Pass fast/small dist. a9 = Pass fast/large dist. a10 = Platooning large dist. a11 = Platooning small dist.	
Sit. 4: Static Obstacle 	a12 = Emergency Stop (not considered in MCDM evaluation)	Only Emergency Stop as feasible alternative. ⇒ Decision: Emergency Stop

- the Petri net based logic of the first decision making stage is defined according to a complete specification (i.e. a specification which defines the decision logic for all urban traffic conditions);

- the MCDM attributes and their weights are specified according to their importance as judged by the transport system experts.

7.3 Real-World Experiments

Experimental decision making tests have been carried out, which show the application of communication for this purpose. In these experiments, 3 vehicles have been used:

- an autonomous vehicle (Cycab, manufactured by Robosoft, France),

- a second, manually driven Cycab,

- a conventional car (Citroen C3).

All vehicles, sensors, and test facilities haven been provided by the French research institute INRIA (team IMARA). The autonomous vehicle was controlled by the autonomous vehicle decision making and control software presented in this thesis.

The three vehicles, including the conventional car, were equipped with differential GPS (DGPS) and were able to communicate over the communication framework presented in the following section 7.3.1 (Figure 7.6). In addition to its own GPS position, the autonomous vehicle was able to receive the GPS positions of the other two vehicles. Furthermore, the autonomous vehicle's world model included a priori information, such as the position of intersections and positions of imaginary stop signs. In order to test the decision making approach, three different traffic scenarios have been set up, all showing a common decision situation: passing a stopped vehicle under different traffic conditions. The following subsection explains the test setup, and the communication framework for autonomous vehicles.

7.3.1 The Communication Setup

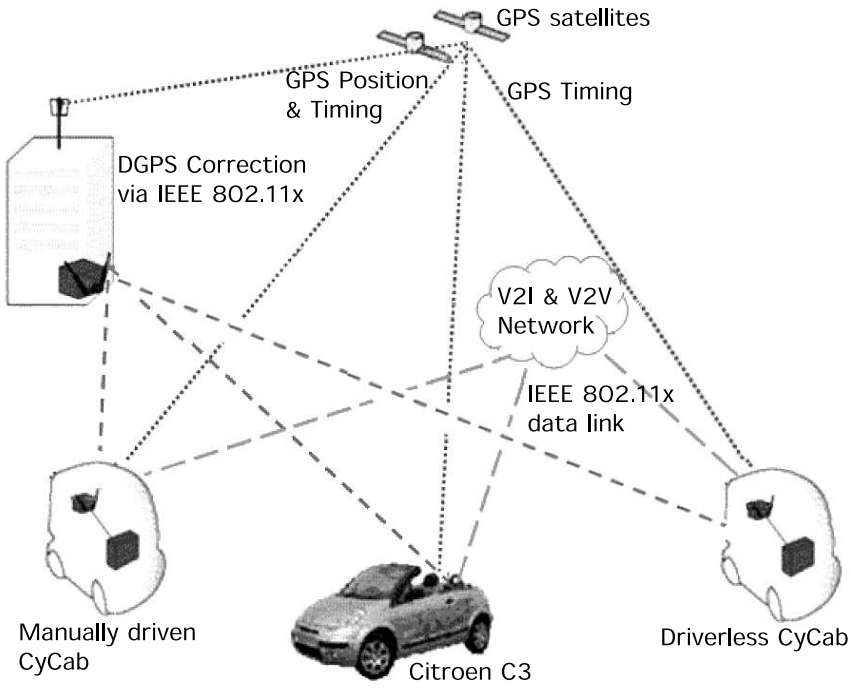


Figure 7.6: Overview of the example communication setup (adapted from [74]). The driverless CyCab communicated with a manually driven CyCab and a conventional car.

The main objective of the Cybercars-2 Communication Framework [74] is to enable autonomous vehicles to safely perform cooperative driving maneuvers. Cybercars are Intelligent Transportation Systems based on road vehicles with fully automated driving capabilities [8, 33]. In the current stage, Cybercars are not intended to operate in public traffic, but in restricted environments, such as airports or theme parks.

The Cybercars-2 Communication Framework enables vehicles to send and receive data, regardless of whether they are driverless or manually driven. Additionally, the communication framework concept includes communication with a traffic management centre, which is however beyond the scope of this work.

Figure 7.6 shows an overview of the Cybercars-2 communication setup.

The example setup consists of three communicating vehicles: a driverless CyCab [75, 76], a human-driven CyCab and a conventional car (Citroen C3) equipped with an Advanced Driver Assistance System. Advanced Driver Assistance Systems provide driving assistance functions, integrating telematic services and interaction between vehicles and the infrastructure [74]. CyCabs are computer controlled vehicles developed by INRIA and manufactured by Robosoft. They provide the option to be driven either autonomously or manually, using a joystick.
The communication framework consists of five layers (Figure 7.7):

- Physical Layer,
- MAC Layer,
- Network Layer,
- System Service Layer, and
- Application Layer.

The Cybercars-2 Communication Framework accommodates the use of the following three communication standards/recommendations: IEEE 802.11p, IEEE 802.11a/b/g and WWAN[1] technologies, such as GPRS, UMTS or WiMAX. The IEEE 802.11p recommendation is used for V2V/V2I communication, IEEE 802.11a/b/g for support information, and WWAN to monitor the traffic flow and to improve its efficiency. At the current stage, communication equipment compliant to IEEE 802.11b/g is used for V2V and V2I communication. Therefore, the MAC Layer includes the functionalities which are available for the commercial IEEE 802.11 compliant equipment.

The hardware and software enabling communication is integrated into the 4G-Cube (Figure 7.8), which is a MIPS-based computer running Linux. It is equipped with one or two Mini-PCI Wi-Fi(b/g) cards and an Ethernet interface [74], providing connections to the vehicle network and wired or wireless connectivity for in-car devices.

In order to fulfill the main objective, which is to enable cooperative maneuvers, the focus is mainly on close proximity communication between

[1]Wireless Wide Area Network

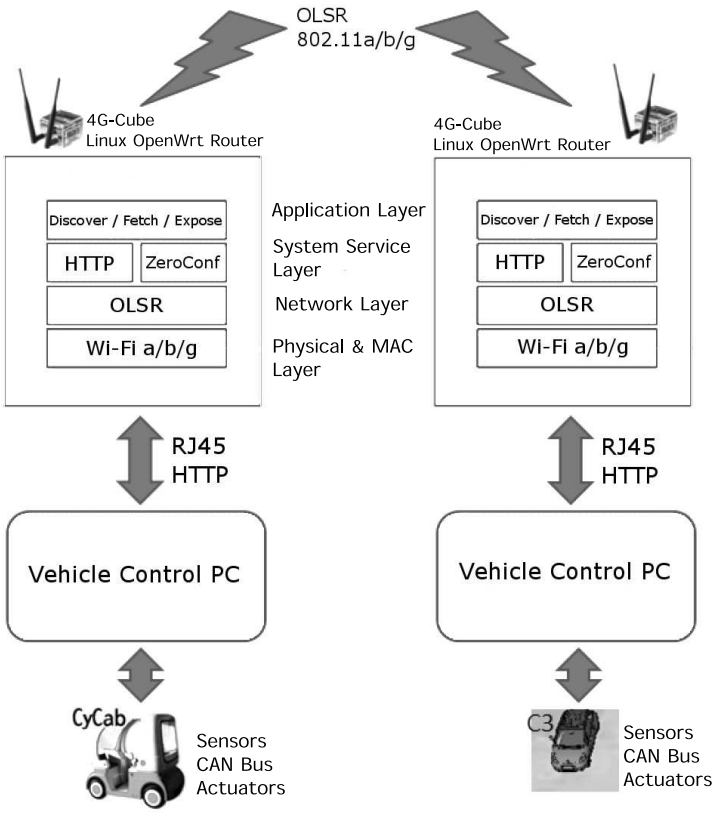


Figure 7.7: The Cybercars-2 Communication Framework architecture [74].

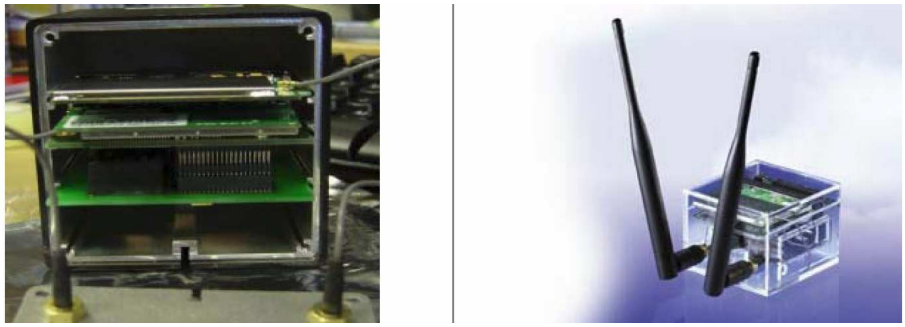


Figure 7.8: The 4G-Cube is a small MIPS-based computer with integrated Wi-Fi(b/g) and Ethernet interfaces [74].

nearby vehicles. In this highly dynamic application environment, involving moving vehicles, dynamic routing is a major requirement. For this purpose, the Optimized Link State Routing protocol (OLSR) is used. The OLSR

protocol provides the functionality required for vehicle communications. It was designed specifically for multihop networks with a strong and efficient mechanism for data flow delivery and it enables the quick creation and reconfiguration of vehicle mesh networks [77].

As part of the system service layer, the service discovery mechanism "Zero configuration" from Apple has been adopted, as it helps to improve the network establishment procedure. Multicast DNS (mDNS) is used to provide a local namespace in order to abstract the vehicle's network addresses. On top of DNS (or mDNS), Service Discovery (DNS-SD) can be used by the 4G-Cube to publish or query information about the applications or services running in the neighborhood [74].

The communication protocol is based on the HTTP 1.1 protocol and uses the HTTP GET and POST requests. The following three functions are provided:

- Discover: To list available services,
- Expose: To send data (i.e. to make data available to all network nodes),
- Fetch: To receive data from a specific network node.

The *discover* function is used to list all available services. Its main use is for retrieving the list of all communicating vehicles (network nodes) along with the types of information they are able to send.

The *expose* function is used to send communication data. The variety of information sent over the communication network is virtually unlimited. It can include for instance the vehicle's current GPS position, speed, heading, data from any on-board sensors, information about its future travel direction, etc.

The *fetch* function is used to receive communication data from a specific network node.

As demonstrated in the experiments, communication (V2V, but also V2I) is very useful for real-time decision making and for improving the road safety of autonomous city vehicles [54,78]. In the experiments, V2V communication

was used as the only source of information in order to test the decision making approach.

The data received through communication and from on-board sensors is processed and kept up-to-date in the World Model, which represents the vehicle's view of its road environment and contains all relevant information, such as near-by vehicles, their speed and travel direction. As part of the World Model, the Perception Module's purpose is to obtain data from all available information sources. Regarding only software design aspects, the origin of obtained data is not relevant. Consequently, in order to simplify the software architecture, this module processes data obtained through communication in the same way as data obtained from the vehicle's on-board sensors. However, the differentiation with respect to the origin of data is still possible, however it is a task of the higher software layer, the World Model.

Figure 7.9 shows the UML [56] class diagram of the Perception Module, along with the functionalities which obtain data from the LIDAR sensor (class Lidar), the GPS receiver (class GPS), and data received through communication over the 4G-Cube (class 4GCubeComm). The class CommPackage represents a communication data package. It includes the vehicle's ID, along with GPS data, such as the vehicle's heading, latitude, longitude, speed, and timestamp.

The class PerceptionModule, the classes which access data from on-board sensors (Lidar, GPS), and the communication class (4GCubeComm) are all executed concurrently, each in its own execution thread. Furthermore, in order to enable simultaneous communication with multiple vehicles, one execution thread is created for each communication channel (i.e. one communication thread per vehicle).

Figure 7.10 shows the interaction between the Perception Module and two vehicle communication threads. In a first step, the Perception Module creates the thread for the communication with Vehicle 1. After initializing the required parameters (method *init*), and the Perception Module initiates data reception by calling the method *receive*. The method *receive* uses the communication protocol function *fetch* to receive data. The receive method completes its execution only when data is received from the specified vehicle.

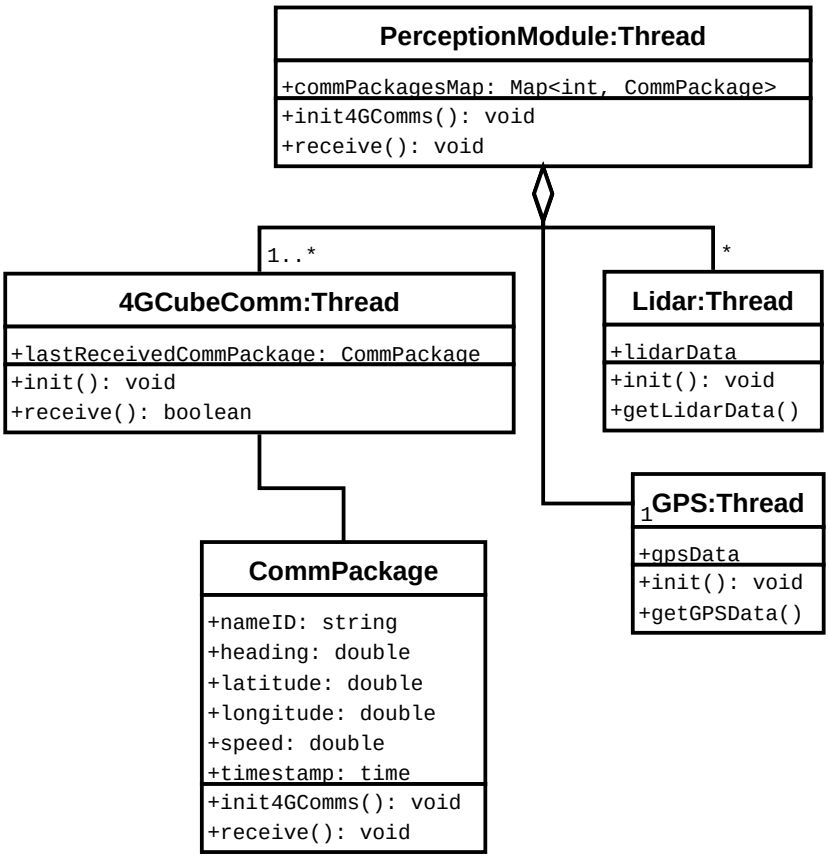


Figure 7.9: UML class diagram showing the Perception Module, the communication classes and classes used to access data from on-board sensors.

While the communication thread is waiting for data to be received, the execution of the Perception Module thread continues. This allows the creation and initialization of another communication thread for Vehicle 2 (Figure 7.10). Whenever communication data from either Vehicle 1 or Vehicle 2 is received (commPackage), the Perception Module receives and stores it. On request, the most recent communication data is provided to the World Model. A discussion about future potentials and current limitations of V2V and V2I communication networks can be found in Appendix C.

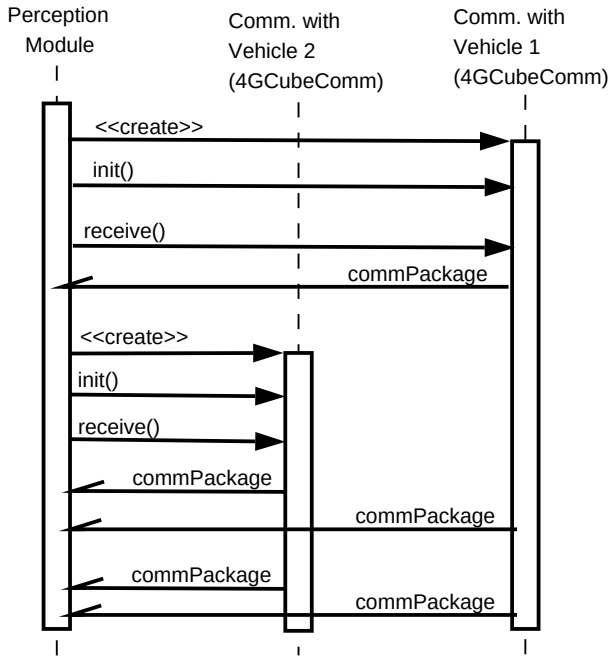


Figure 7.10: UML sequence diagram showing the interaction between Perception Module and the asynchronous communication threads (class 4GCubeComm).

7.3.2 Experiment 1

In the first traffic scenario, the autonomous vehicle approached a stopped vehicle. Safe passing was possible, and the oncoming traffic lane was free of any obstacles (Figure 7.11). In this first scenario, the autonomous vehicle immediately started the passing maneuver when it approached the stopped vehicle.

The autonomous vehicle's speed while approaching the stopped vehicle was around 1m/s, while the passing maneuver was executed at around 0.7m/s. The passing maneuver was started around 3-5 meters before the autonomous vehicle reached the stopped vehicle. A minimum safety distance of 1 meter to obstacles was defined, which triggered the activation of the emergency stop in the case that any obstacles were detected closer.

This experiment was repeated 17 times, and the correct decision was always made. However, after the activation of the driving maneuver for

passing, the software crashed[2] 8 times due to implementation errors, and other components of the control software and hardware (e.g. communication or GPS failure). Nevertheless, since the main objective of these experiments was to test the decision making process, and not the execution of driving maneuvers, the results can be regarded as successful.

Figure 7.11: Experiment 1. The autonomous vehicle passes a stopped vehicle.

7.3.3 Experiment 2

The second traffic scenario was similar to the first, however another manually driven vehicle was oncoming, making safe passing impossible (Figure 7.12). In this second scenario, the autonomous vehicle waited behind the stopped vehicle, and started passing the stopped vehicle when the oncoming traffic lane was free.

Similar to Experiment 1, the approaching speed was 1m/s, the distance to the stopped vehicle was set to 2-3 meters, and the passing maneuver was executed at around 0.7m/s.

[2]Due to the main focus on road safety, whenever software hardware problems occurred while driving in automatic mode, the vehicle always stopped safely without the need of remote emergency stopping.

This decision making experiment was successfully repeated 13 times without decision making failure or collision. However, similar to the first experiment, due to software and hardware problems, the vehicle was not able to continue driving in 5 repetitions.

Figure 7.12: Experiment 2. The autonomous vehicle does not pass the stopped car due to an oncoming vehicle. It waits behind the stopped car until the oncoming lane is free. Then it passes the stopped car.

7.3.4 Experiment 3

In the third traffic scenario, a manually driven vehicle was stopped at an intersection (Figure 7.13). The autonomous vehicle waited behind the stopped vehicle until it crossed the intersection. Then the autonomous vehicle continued driving, stopped at the imaginary stop sign before continuing across the intersection.

The approaching speed to the intersection was around 1m/s, while the intersection crossing maneuver was executed at 0.7m/s. Another relevant parameter for this experiment was the distance to the intersection, from which the passing maneuver was considered infeasible. Due to space limitations,

and the low vehicle speed, this distance was set to only 10-15 meters.

This experiment was repeated 12 times, without decision making failure. Since the problems of the previous two experiments could not be solved in the available time frame, after stopping at the intersection, the vehicle was not able to continue driving in 5 repetitions.

Figure 7.13: Experiment 3. The autonomous vehicle does not pass a vehicle which stopped at an intersection, but waits for it to proceed. It continues driving when the front vehicle clears the intersection.

7.4 Discussion

Table 7.2 lists the number of successful and failed repetitions during the conducted experiments. These results were recorded after the software and hardware systems reached a sufficient reliability for our test purposes. Although there were a number of remaining problems, such as the unreliable execution of driving maneuvers, which could not be solved in the available time frame, the results show that in all repeated experiments, the decision making subsystem was always able to avoid collisions with other vehicles and make appropriate driving decisions in real-time. Consequently, as real-time decision making was purely based on communication, the results also

show that the used communication framework proved to be sufficiently reliable, and is therefore useful for improving the safe operation of autonomous vehicles.

Table 7.2: Results of real-time decision making experiments.

Experiment	**# trials**	**Decision Making**		**Driving Maneuvers**	
		successful	failed	successful	failed
Experiment 1	17	17[a]	0	9	8[b]
Experiment 2	13	13[c]	0	7	5[de]
Experiment 3	12	12[f]	0	7	5[g]

[a]The decision to pass was made correctly and the passing maneuver was activated.
[b]The driving maneuver for passing was not correctly implemented.
[c]The decision to pass when possible was correctly made.
[d]The execution of the passing driving maneuver failed.
[e]One recorded experiment included only the decision making and no subsequent driving maneuvers.
[f]The decision to stop and wait behind the stopped vehicle was correctly made.
[g]The driving maneuver failed for continuing when the intersection was free.

The main goal of the experiments related to real-time decision making was to demonstrate that the ICSL autonomous vehicle decision making & control software, and most of all the ICSL real-time decision making approach works with real vehicles and sensors, and meets the real-time decision making requirements. The integration of the Cybercars-2 Communication Framework developed by INRIA into the ICSL control software was the first opportunity to test the decision making approach under real-world conditions, while at the same time proving the usability of the Cybercars-2 Communication Framework.

During the experimentation phase at INRIA, all experiments were repeated numerous times. Often, new software or hardware related problems were detected and solved. Consequently, these experimental results cannot be regarded as a rigorous test benchmark for the quality of the entire system. Nevertheless, these experimental tests have demonstrated the feasibility of the developed approach, and the prototype implementation.

The following Chapter 8 concludes this thesis.

Chapter 8

Conclusion

8.1 Contributions of this Thesis

This thesis has addressed and presented a solution for the problem of real-time decision making by autonomous city vehicles, which included contributions towards three main objectives, as follows:

- World Model,
- Real-Time Decision Making,
- Driving Maneuver Design Concept.

8.1.1 World Model

While approaches for modeling the environment of autonomous robots for other applications exist (e.g. for off-road applications [50]), no such solution has been published yet, which is able to include all entities required for autonomous driving in urban traffic conditions. This includes a priori known information, information obtained through communication, and information obtained from on-board sensors.

The presented World Model was developed based on standard object-oriented software engineering methods, which were applied in this new research area with overlapping engineering and computer science tasks. The

presented World Model is able to store and manage information about relevant entities, such as roads, intersections, traffic lanes, traffic signs, pedestrians, etc., as well as the relationships between them. This information is provided as input for the decision making subsystem, in an appropriate data structure and in real-time, enabling safe autonomous driving in real-world urban traffic conditions.

Therefore, the developed World Model represents a significant contribution for merging data from different sources of information, and towards achieving the main objective of autonomous driving in real-world urban traffic conditions.

The essence of the developed World Model has been published in the Proceedings of the 2010 IEEE Intelligent Vehicles Symposium (IV 2010), San Diego, California, USA [48].

8.1.2 Real-Time Decision Making

In addition, the main contribution of this thesis is the real-time decision making approach, which has been addressed with respect to the system specification, model design, and software implementation.

The decision making process has been subdivided into two consecutive stages, each focusing on different objectives. While the purpose of the first decision making stage is to determine which driving maneuvers are feasible in the current traffic situation, the second decision making stage focuses on non-safety relevant objectives, such as improving comfort and efficiency.

Taking into account the need to specify a complex operational behavior which depends on a large number of events occurring in the vehicle's surrounding traffic environment, the first stage of the decision making process uses Petri nets to enable the modeling and analysis of this stage. The second decision making stage uses Multiple Criteria Decision Making (MCDM), an established mathematical method which allows to include and take into account a large number of possibly conflicting decision objectives.

Although the decision making methods applied in this thesis are well established modeling and analysis tools often applied in the design, modeling,

and implementation of complex systems, their combined application in this new research area of autonomous city vehicles is an innovative contribution towards enabling autonomous vehicles to cope with the complexity of urban traffic situations.

The main concept of the developed Real-Time Decision Making approach are being published in:

- The Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics, San Antonio, Texas, USA, *"Towards Increased Road Safety: Real-Time Decision Making for Driverless City Vehicles"* [47];
- The Proceedings of the 7th IFAC Symposium on Intelligent Autonomous Vehicles (IAV 2010), Lecce, Italy, *"Multiple Criteria-Based Real-Time Decision Making by Autonomous City Vehicles"* [66];
- The IEEE Intelligent Transportation Systems Magazine, *"Enabling Safe Autonomous Driving in Real-World City Traffic using Multiple Criteria Decision Making"* [79].

8.1.3 Driving Maneuver Concept

The idea of subdividing the complex task of autonomous driving into subtasks, the driving maneuvers, is not new. However, so far developed solutions have focused on simplified driving requirements, and not on driving in the complex real-world urban traffic conditions. Furthermore, the published material on this topic often omits details about the modeling approaches and implemented solutions. Therefore, this topic has readdressed and presented a systematically developed solution for the modeling and implementation of driving maneuvers, based on finite automata.

Using finite automata in the presented way for the driving maneuver concept enables the modeling of driving maneuvers with arbitrary complexity and purpose, using identical data structures. This enables the decision making process to start, stop, and retrieve status information from driving maneuvers, regardless of their complexity or driving task.

Consequently, the presented driving maneuver concept enables a more structured development of complex driving maneuver algorithms, which form a basis of decision alternatives for the decision making process.

The driving maneuver design concept has been addressed in the papers in the Proceedings of:

- the 2009 IEEE International Conference on Systems, Man, and Cybernetics, San Antonio, Texas, USA, *"Towards Increased Road Safety: Real-Time Decision Making for Driverless City Vehicles"* [47].

Furthermore, the achieved experimental results are being presented in the Proceedings of:

- the 3rd IEEE International Symposium on Wireless Vehicular Communications: IEEE WiVEC 2010, Taipei, Taiwan, *"Improving Safety for Driverless City Vehicles: Real-Time Communication and Decision Making"* [54];

- the 2010 IFIP International Conference Network of the Future, Brisbane, Australia, *"The Role and Future Challenges of Wireless Communication Networks for Cooperative Autonomous City Vehicles"* [78];

- and in the Journal of Robotics and Mechatronics, *"Real-Time Decision Making for Autonomous City Vehicles" [80].*

8.2 Future Challenges

The main challenges for the future development of autonomous city vehicles consist of a combination of technical, theoretical, but also sociological and legal challenges.

One of the main technical challenges is the need for better sensors, perception, and localization systems, which are able to provide accurate, reliable, and consistent information about the vehicle's surrounding environment and its position. All safety relevant components of such vehicles need to guarantee safe operation, by including redundant components, fault tolerance and

error recovery techniques. The problems of sensor noise, sensor uncertainties, and sensor reliability need to addressed as part of the fault-tolerant system solutions.

Furthermore, more realistic simulation environments are needed, which enable the simulation and testing of various decision making & control system components in particular, but also simulations of the entire system in realistic traffic scenarios. Realistic simulation environments would enable the testing of autonomous vehicles without putting traffic participants at risk in premature on-road testing procedures.

The theoretical challenges include the development of a complete system specification for autonomous driving in real-world urban traffic. So far developed solutions rely on the developer's driving knowledge how to deal with certain conditions, and on ad-hoc tests. However, in order to be able to achieve road safety and reliability, the possibility to formally verify the system's correct operation is crucial, and this task requires a system specification.

While the prototype implementation and the presented evaluation results demonstrate that the developed decision making approach is applicable and suitable, additional work is necessary in order to advance its development towards commercial real-world applications. In order to obtain unconditional reliable decision making results, the following aspects need to be addressed by transportation system experts:

- The minimal set of traffic environment information provided by the World Model needs to be refined in the context of making safe driving decisions.
- The currently developed set of driving objectives needs to be expanded to unquestionably reflect the system specification.
- Additional research is required for the development of a complete set of driving maneuvers for urban traffic, in order to enable real-time decision making and autonomous driving in any situation.

Although there seem to be many challenges to overcome, besides the

current impulse created by military objectives, the need for safer civilian transport systems, and the pursuit of the unparalleled benefits provided by civilian autonomous city vehicles, will create the market demand for such vehicles, and will lead to the realization of this vision in the near future.

Appendix A

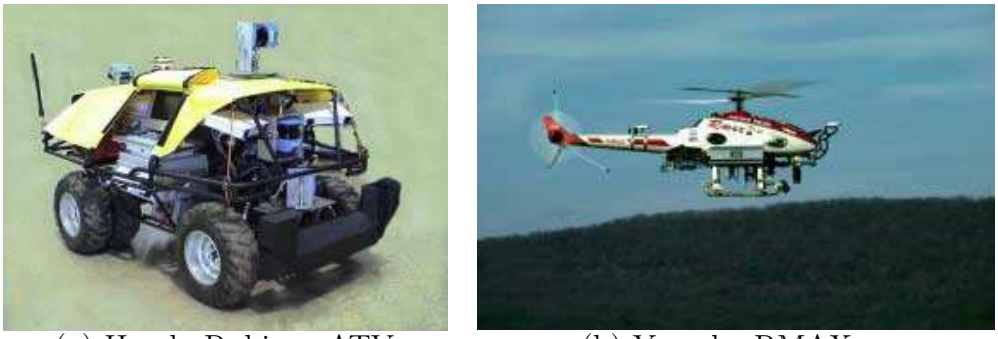
Overview of Related Autonomous Vehicle Projects for Highway and Off-Road Applications

A.1 Autonomous Off-Road and Highway Vehicle Projects

A.1.1 DARPA PerceptOR Program

The DARPA PerceptOR (Perception for Off Road Navigation) program focused on the development of autonomous off-road navigation techniques for unmanned off-road ground vehicles. The objective was the use of autonomous off-road vehicles for military applications. Field tests were conducted in the years 2001 and 2002 [23].

In order to support and improve the perception abilities of an autonomous off-road ground vehicle, a cooperating autonomous helicopter, a so-called Flying Eye (FE), supported the ground vehicle with additional video and LIDAR data [23]. The ground vehicle, a Honda Rubicon ATV (Figure A.1a), was equipped with 2D LIDAR sensors, two stereo camera pairs and several other cameras used for terrain classification. The autonomous helicopter, a Yamaha Rmax (Figure A.1b), was equipped with a LIDAR sensor and transmitted 3D data to the autonomous ground vehicle. The autonomous helicopter received navigation points from the ground vehicle. For evaluation purposes, the ground vehicle was supervised by an operator and assisted by a field team and a data analysis team (Figure A.2).



(a) Honda Rubicon ATV (b) Yamaha RMAX

Figure A.1: The PerceptOR autonomous ground vehicle and its supporting helicopter [81].

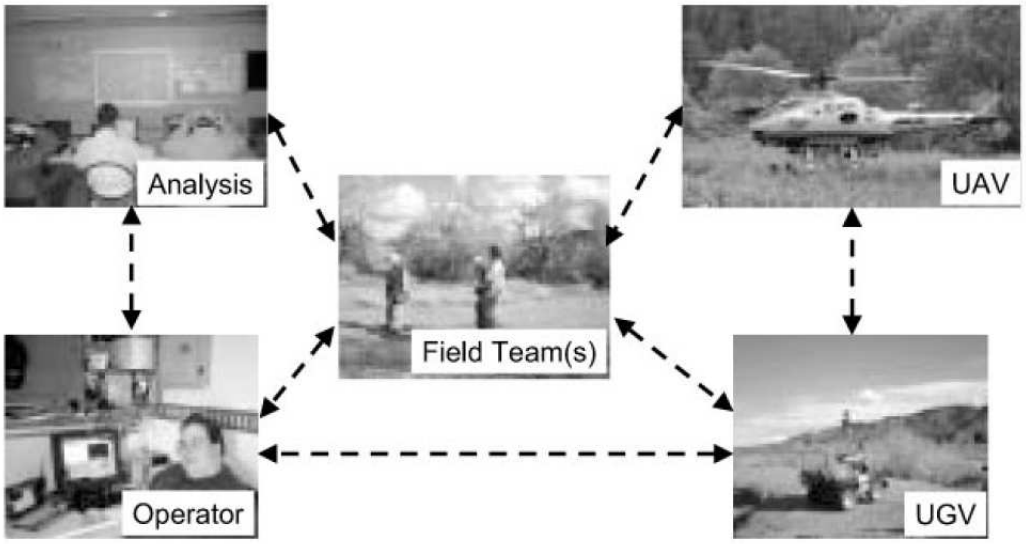


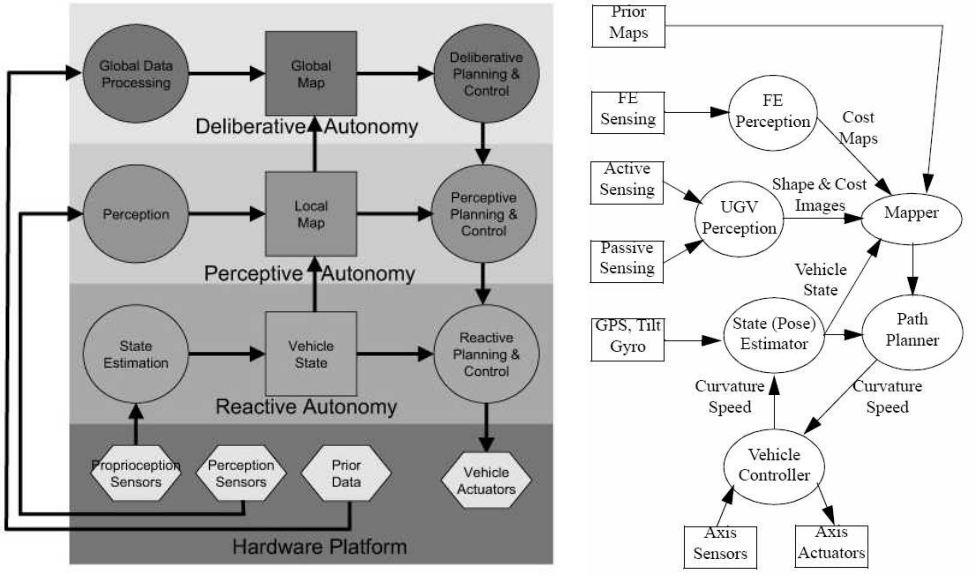
Figure A.2: PerceptOR coordination structure [82].

PerceptOR's control software architecture consisted of the following three layers, so-called autonomy layers (Figure A.3.a):

- Deliberative autonomy layer,
- Perceptive autonomy layer,
- Reactive autonomy layer.

Each lower layer executed commands sent by a higher layer. However, in order enable quick reactions to emergency situations, each lower layer was

able to ignore higher layer commands. Each layer integrated functionalities for data interpretation, a world model and vehicle control. Starting from the lowest, the reactive layer, the execution times of each layer increased along with increasing task complexity. The reactive control loop was executed at around 100Hz and performed reflex like safeguarding functions for the vehicle [81]. The reactive layer also executed trajectory following tasks, vehicle specific tasks, and was able to react to exceptions and alarms.



(a) Control software architecture. (b) Processing and flow of data.

Figure A.3: PerceptOR control software architecture and flow of data [81,82].

The middle architecture layer, the so-called perceptive autonomy layer, included image processing, the fusion of information from different sensors, and the creation of 2D and 3D maps of the environment. As the vehicle was designed to drive on any terrain, the ability to distinguish between soft vegetation and hard obstacles was crucial. Compressible and therefore driveable terrain was classified based on the colour. Green objects were classified as grass and bushes, while brown objects, such as rocks or dirt, were classified as rigid. The density of objects was measured based on the LIDAR scans, which revealed empty spaces (e.g. in grass). These two color-based classification methods worked for most cases, but not always. Problems were caused for instance by dry and therefore brown grass, or fences, which are penetrable by LIDAR, but not compressible [23, 81].

The deliberative autonomy layer was the highest layer and executed the most complex tasks regarding memory and computing costs. As part of this

layer, the motion planner used the Field D* search algorithm[1] for global path planning and a prediction algorithm for the local planner.

Running at 10Hz, the global path planner assigned terrain costs based on data received from the autonomous helicopter, if available. Otherwise, if no aerial data was available, the terrain cost function was set to an intermediate value. The search algorithm operated on an 8-connected grid map. In each run, the global path planner replanned a path from the current position to the destination point. The local path planner was executed at the same frequency as the global planner (around 10Hz) and used a prediction model based on terrain elevation data. The local path planner determined the autonomous ground vehicle speed, and was also able to stop the vehicle if a collision was predicted. In the case that all trajectories were classified unsafe, the ground vehicle was slowed down [82].

Figure A.3.b shows the data processing modules and the flow of data in PerceptOR's control software. The so-called mapper module receives its input from a priori terrain maps and online from the helicopter sensors (FE perception) and on-board sensors (UGV perception). Based on the mapper output and the vehicle's position (State Estimator), the path planner sends speed and curvature commands to the vehicle controller. The path planner controls the vehicle in a closed loop, as it receives the actual speed and curvature feedback from the vehicle controller .

The deliberative autonomy layer executed four so-called behaviors: "navigate", "lookaround", "goaround", and "planaround" [82]. The navigate behavior was active during normal driving operation. The lookaround behavior performed a complete scan of the environment while the vehicle was stopped. The goaround behavior attempted to align the vehicle to match the path planner direction. The planaround behavior was activated if the goaround behavior failed, or the vehicle could not proceed. In that case, a new path was planned. If no other path could be found, the human operator would intervene. An arbitration mechanism [82] was used to resolve disagreements arising from conflicting simultaneous goals such as obstacle avoidance and waypoint seeking.

In 2001 and 2002, the autonomous system was tested in an extensive test program in different terrain environments. Besides technical difficulties, the perception system was not reliable enough to detect all obstacles and to operate in harsh weather conditions, such as rain. The path planners (local and global) were not always able to find the solution, even though a

[1] D* is a heuristic search algorithm derived from A*. Its main advantage over A* is its flexibility in the case of changing cost parameters during the search, as in such a case it does not require a search restart like A*. D* has been first published in [83].

driveable path existed. The main reason was the perception system, which relied on heuristic cost functions when fusing data from different sensors. The developers noted that *"cost fusion ultimately proved far more subtle and complex than we had hoped"* [82]. Furthermore, there were problems related to real-time execution: *"the system often detected obstacles, but was unable to respond adequately before encountering the obstacle"* [82].

A.1.2 DEMO III Program

DEMO III was a research program focusing on the development of autonomous terrain vehicles for military applications. Different vehicles were used (Figure A.4), however all were similarly equipped and running the same control system software. Field tests were carried out in the years 2000 and 2003 [24–26].

The DEMO III program's objective was to develop an autonomous vehicle able to drive up to 40 mph on roads, 20 mph off-road by day and 10 mph off-road by night. The vehicles were equipped with LIDAR sensors, colour and infrared cameras and GPS/DGPS receivers.

(a) XUV [24] (b) JPL HMMWV [24] (c) XUV [25]

Figure A.4: Autonomous vehicles used in the DEMO III project.

The control software was based on the RCS architecture. The RCS (so-called Real-time Control System)[2] architecture has been proposed by Albus et al. [26,84]. RCS claims to be a so-called "cognitive"[3] control system architecture for autonomous vehicles, and includes higher military organizational layers, such as company or battalion organization for military applications, which integrate the use of autonomous vehicles (Figure A.5). However, for the sake of brevity, this work focuses only on the architecture layers which are relevant

[2]In this case, the abbreviation RCS is a specific name, and does not refer to a general real-time control system.

[3]Defined by Albus et al. as "the organizational structure of functional processes and knowledge representation that enable the modeling of cognitive phenomena" [26].

for autonomous vehicles, and does not address the so-called Section, Platoon, Company and Battalion layers.

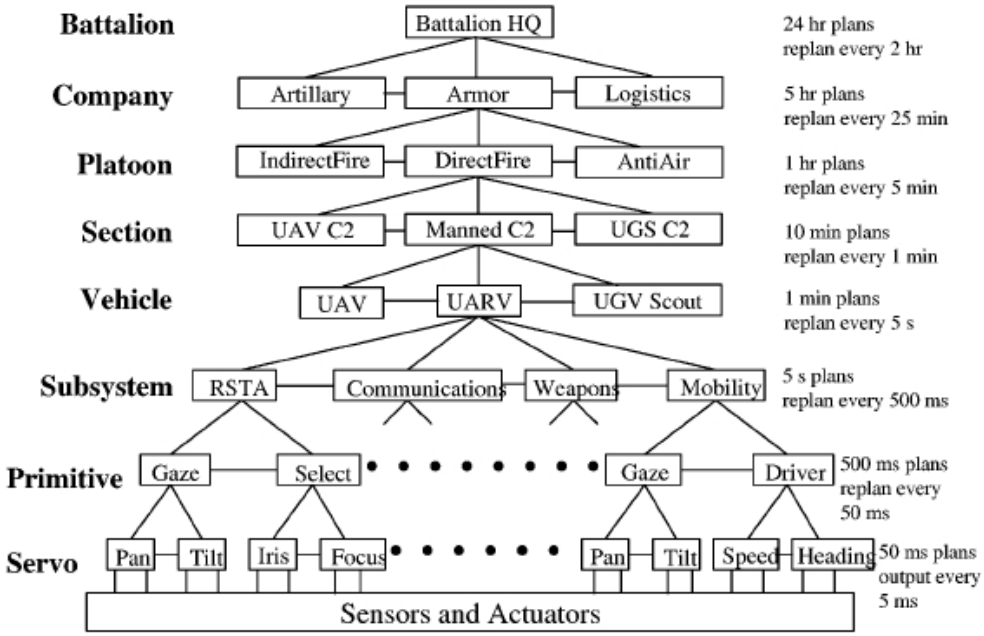


Figure A.5: The RCS hierarchy levels for a combat systems integrating autonomous vehicles [84]

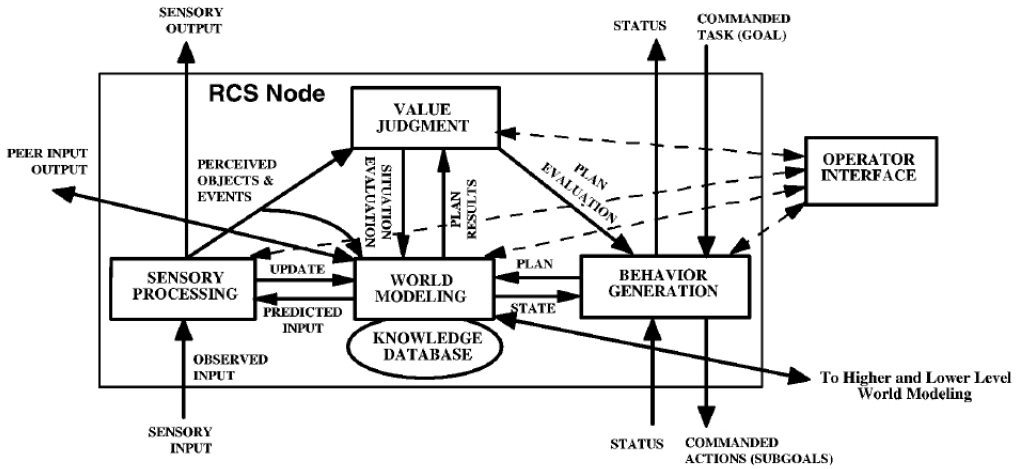


Figure A.6: The RCS node structure [84]

The RCS architecture is hierarchically structured and is based on so-called processing nodes. Each processing node contains the following modules (Figure A.6):

- Behavior Generation,
- World Modeling,
- Sensory Processing,
- Value Judgement,
- Knowledge Database.

Each behavior generation module includes a planner and a set of so-called executors. The purpose of the planner is to decompose tasks into subtasks, which can be solved by the executors. Executors carry out commands, but can also react to emergency situations. By including planning and reactive behavior in each node, deliberative plans and reactive behaviors are combined.

The world modeling module is used to create maps of the environment based on input from the sensory processing modules. The purpose of the value judgement module is to "evaluate expected results of tentative plans" and to assign "confidence and worth to entities, events and situations" [26].

In addition to the RCS architecture, Albus and Barbera [26] proposed a top-down methodology for the design of a control system for "autonomous on-road driving under everyday traffic conditions" based on RCS. The proposed methodology consists of the following six steps: single

1. Analyse the domain, create a decomposition tree for tasks *covering every traffic scenario.*

2. Define a structure of organizational units that will execute the tasks defined in step one.

3. Specify the algorithms and processing of each unit defined in stop two. The algorithms should be based on state automata.

4. Identify relationships between entities, events and states.

5. Identify mechanism to detect the relationships identified in step 4.

6. Define sensor requirements.

The DEMO III autonomous off-road vehicles have been tested in different environments, including urban areas. However, none of the conducted tests included moving obstacles, on-coming traffic, pedestrians or any other vehicles [26].

A.1.3 ARGO

ARGO was a research program conducted by the University of Parma, Italy [22, 28, 85–87]. The program focused on the development of an active safety system and an autonomous pilot for highway traffic. The vehicle was a standard car equipped with two cameras, a standard PC and a steering angle controller. The vehicle's steering angle was automatically controlled, while the speed was manually controlled by a human driver in a conventional manner. Autonomous vehicle demonstrations were carried out in the years 1998-2001.

Figure A.7: The autonomous vehicle ARGO [22].

The only sensors used were two cameras and a speedometer. The automatic driving mode allowed the selection of two functionalities: road following, and platooning. The road following mode controlled the steering angle of the vehicle in order to follow a lane detected using cameras. Platooning mode included the detection of other vehicles in front and controlled the vehicle's steering angle [86].

The vehicle's control software named GOLD ("Generic Obstacles and Lane Detection") consisted of around ten thousand lines of "C" source code and included computer vision algorithms, which were able to detect the lane markings and other vehicles in front. However, it was assumed that the road in front of the vehicle was flat and that all vehicles are symmetric [85, 87, 88].

In 1998, the ARGO vehicle was able to drive 2000 km on Italian highways, "mostly in autonomous" mode. However, difficulties arose in specific traffic situations such as tunnels, when the cameras failed to provide useful data.

A.1.4 VITA II

VITA II (Vision Technology Application) was an autonomous vehicle developed in the early 1990s by Daimler-Benz as part of the European research program PROMETHEUS[4]. The program's objective was "automatic vehicle guidance" on motorways [9], aiming to solve problems caused by increasing traffic on European highways. The VITA II objective was mainly research on the field of computer vision for autonomous highway vehicles, but also the realization of a prototype vehicle [9].

Figure A.8: The VITA II vehicle [9].

The vehicle VITA II was a modified conventional car (Figure A.8), equipped with drive-by-wire technology, a transputer[5] system with 70 microprocessors and 18 cameras.

In automatic mode, VITA II was able to follow a lane, to change traffic lanes, to overtake, to keep the safety distance while following other vehicles, and to recognize and comply with highway traffic signs and rules. A human vehicle operator was able to specify the vehicle's nominal velocity using the vehicle's conventional cruise control. Attempting to keep the specified speed, the vehicle overtook autonomously if necessary and possible, or slowed down behind slower vehicles otherwise.

The vehicle was equipped with three computing units, each unit integrating a transputer system. A so-called application computing unit executed computer vision and behavior control tasks. A second computing unit was used for basic vehicle control, such as throttle, breaks, and steering. The third computing unit controlled the movement of the camera platforms.

[4]Programme for European Traffic with Highest Efficiency and Unprecedented Safety. PROMETHEUS was founded 1985 as a consortium of 15 European car manufacturers, 70 research groups and 200 subcontractors [9].

[5]A microprocessor with extensions for parallel execution.

Distances to obstacles were calculated using camera images. Besides cameras and sensors measuring the vehicle's state (e.g. speed, steering angle), no other sensors were available. All obstacle detection and collision avoidance algorithms were based on computer vision. In order to reduce the computation times, these algorithms were reduced to a minimum. As a result, the only detectable obstacles were cars and trucks, while other obstacles, such as motorbikes, pedestrians or bicycles remained undetected. Furthermore, in order to reduce the computation time and meet real-time requirements, the number of obstacles to be avoided was reduced to a maximum number of fifteen or even eight. These restrictions enabled the vehicle controller to achieve a system reaction time of 40ms.

The control software consisted of 12 modules , which were categorized into three main layers (Figure A.9):

- Vehicle Control Layer (containing the VC module),
- Sensor Modules Layer, and
- Decision Making Layer, misleadingly called "Control System" in the original publication [9]. This layer contained a Situation Assessment (SA) and Behaviour Control (BC) module.

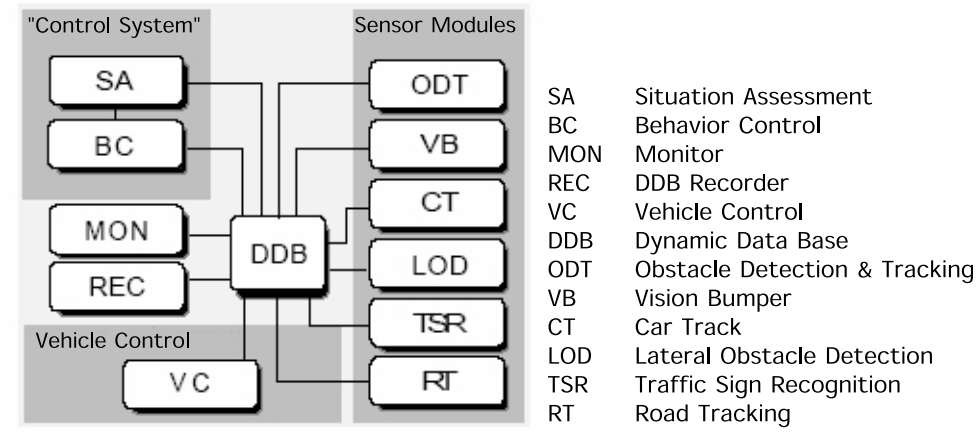


Figure A.9: The VITA II control software architecture (adopted from [9] and translated).

The exchange of data between modules was implemented using a shared data module, the so-called Dynamic Data Base (DDB). The modules MON and REC monitored and recorded data for evaluation and analysis purposes.

The required number of CPUs (Transputer nodes) was allocated for each module, guaranteeing the parallel execution under real-time constraints. For instance, the road detection module, which kept the vehicle in the within lane limits, was executed on 11 CPUs and had a cycle time of 40ms.

The situation assessment (SA) module extracted information from sensors, while the so-called behavior[6] control (BC) module was responsible for controlling the vehicle's speed and steering angle. Algorithms for error detection and fault tolerance were implemented in order to achieve a high reliability.

The lane following and obstacle avoidance algorithms were based on the potential field method. The potential field method models the autonomous vehicle as a particle, which is exposed to imaginary forces representing the environment. A more detailed explanation of this method can be found in [89].

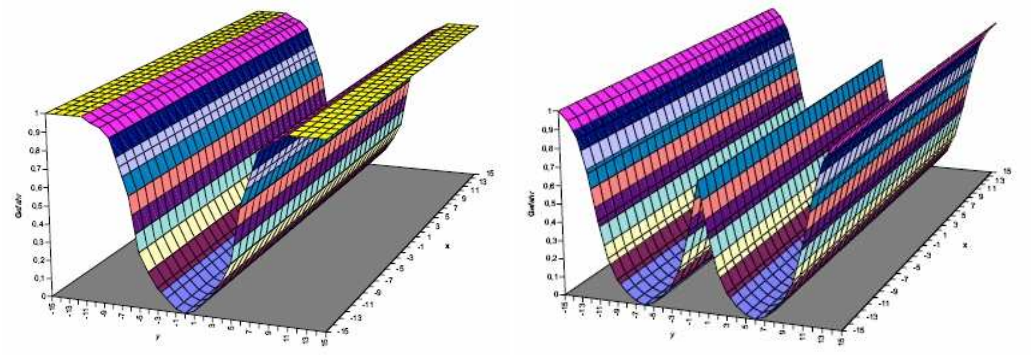
While driving, VITA II created maps of the environment, which classified surrounding vehicle areas according to estimated risk and assigned each area a so-called danger value. Areas representing a danger for the vehicle were assigned a higher danger value, resulting in so-called danger hills. Obstacle free, and therefore considered as not dangerous areas, were assigned a low value, resulting in not elevated map areas. Using such maps with estimated danger values, the vehicle tried to follow non-dangerous areas.

Besides obstacles, the detected lane boundaries were classified as "dangerous" and were assigned a higher danger value. Figure A.10 shows two obstacle free maps for a one and two-lane road respectively. On the one lane map, only the boundaries are elevated, meaning a higher danger value. The two lane map contains an elevated middle line, which is however lower than the road boundaries. This keeps the vehicle preferably within one lane, while still enabling a lane change if an obstacle is detected.

In addition to obstacles and traffic lanes, the potential field method was used to represent traffic rules and human operator intentions. Reichardt [9] argues that traffic rules imply a restriction which can lead to a danger situation if not respected. Therefore, traffic rules were modeled as danger hills, similar to obstacles.

The operator's intentions (i.e. the specified nominal vehicle speed) were modeled on a potential field map by inclining the map plane. The map was inclined forward if the actual velocity was lower than the nominal value, creating a danger hill behind the vehicle; respectively, the map was inclined backward if the actual velocity was higher than the nominal velocity.

[6]Reichardt [9] used the term "behavior", while this work uses the term driving maneuvers.



(a) One lane model. (b) Two lane model.

Figure A.10: Lane models using the potential field method [9].

The different maps modeling obstacles, traffic rules and human operator intentions were fused into a single map, which was used by the decision making module (i.e. the so-called behavior control module (BC)). The map fusion algorithm allowed the specification of preferences, defining a weight factor for each specific map. For instance, the obstacle avoidance map was given a higher weight factor compared to the lane boundary map, resulting in the vehicle's preference to avoid an obstacle instead of colliding with an obstacle in order to avoid crossing the road boundaries.

However, Reichardt [9] admits that using inadequate weight factors for the fusion of maps can lead to conflicting and even dangerous situations. Figure A.11 shows the fusion of a two lane road map with an obstacle map. The weight factor of 3:2 for the maps (a) and (b) leads to map (c), which will lead the behavior control to prefer avoiding the obstacle, even if the vehicle has to cross the road boundary. However, using a weight factor of 2:3 leads to map (d), which leads to a crash instead of crossing the road boundary. Similar dangerous situations can be created by fusing other potential field maps with inadequate weight factors.
Reichardt [9] developed the so-called behavior pattern approach[7] as a behavior based architecture, defining three groups of behaviors:

- Basic Behavior Patterns,
- Schematized actions,
- Emergency actions.

Basic behavior patterns include low-level vehicle control tasks, such as keeping the vehicle within the lane or controlling the distance to the front vehicle.

[7]Original German name: "Verhaltensmusterkonzept".

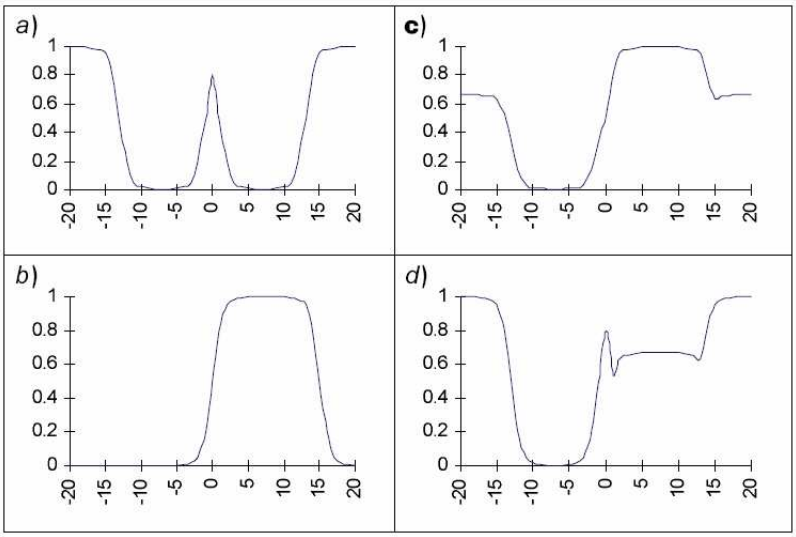


Figure A.11: Fusion of a road map with an obstacle map [9].

Schematized actions integrate higher-level control tasks, such as overtaking. The purpose of emergency actions is to avoid vehicle damages if the other control system levels fail.

A "behavior pattern", which corresponds in fact to a driving maneuver, is defined as a quadruple which contains a type name, a set of objects describing the traffic situation and restrictions, a state object and an algorithm. Each "behavior pattern" was implemented as an automaton and was assigned a priority of execution.

The connection between the so-called behavior patterns was realized using an inhibition network, which modelled the behavior patterns as edges and assigned each state transition an inhibition factor. For instance, the activation of a behavior pattern for overtaking inhibited the activation of the pattern used for following the right traffic lane. The purpose of the inhibition network was to solve conflicts between behavior patterns and enable their connection, which is in fact an approach for decision making.

After a successful simulation, the system was tested on public roads. The autonomous vehicle was however supervised by a human operator, who was able to switch to manual mode, for instance in dangerous situations. The VITA II vehicle drove autonomously about ten thousand kilometers on highways in Germany and France.

A.2 Theoretical Decision Making Proposals

While the previous sections addressed fully developed concepts of autonomous vehicles, this section gives a brief overview of mostly theoretical proposals for or related to decision making for autonomous city vehicles.

A.2.1 Situation Assessment using Pattern Matching

The decision making process for autonomous vehicles represents the system's response to a traffic situation. However, before being able to make any driving decision, the traffic situation needs to be assessed based on perceived information. Therefore, the problem of situation assessment is a first subtasks in a decision making approach.

Lattner et al. [90,91] proposed an approach for traffic situation assessment using on pattern matching. However, their approach focused only on the identification of risky or dangerous traffic situations.

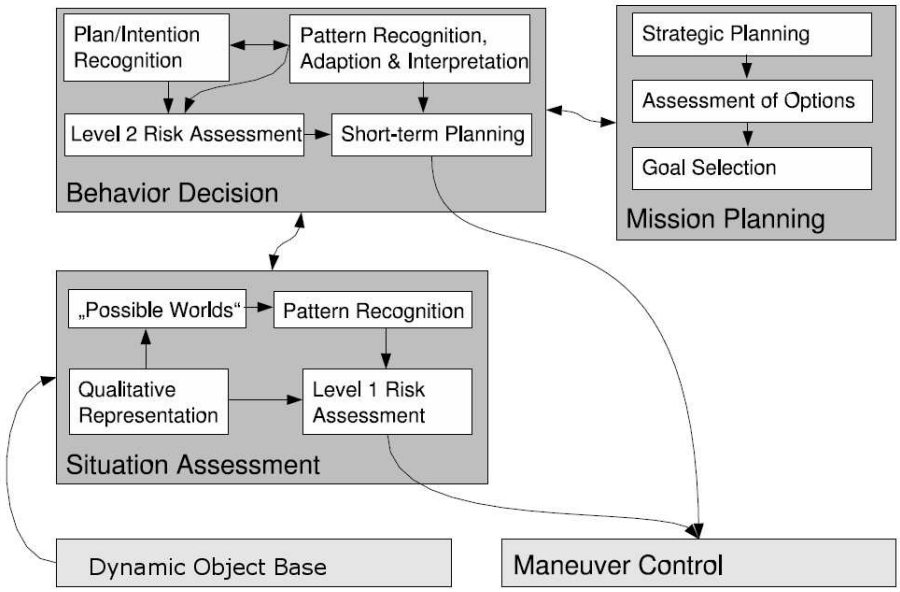


Figure A.12: Situation assessment architecture using pattern matching [90].

The architecture of the so-called knowledge-based system contains three modules (Figure A.12):

- Situation Assessment,
- Behavior Decision,

- Mission Planner.

The Situation Assessment module uses information about the environment, which is provided by the so-called Dynamic Object Base (DOB). On a lower, faster responding layer, the situation assessment analyses the traffic situation. If an emergency situation is recognized, the Situation Assessment module is able to respond by sending commands to the maneuver control module (Level 1 Risk Assessment).

The Possible Worlds module in the Situation Assessment contains a model of the perceived traffic situation as well as predicted future situations. The "Behavior Decision" module decides about the following vehicle maneuver based on pattern matching algorithms. The Mission Planner represents the long-term level of decision and planning.

The main idea of this proposal is the Situation Assessment module, which contains the Pattern Recognition module. The Pattern Recognition module uses a knowledge base containing a set of traffic situation patterns. In a pattern matching process, the currently perceived traffic situation is compared to the known patterns stored the knowledge base.

The information stored in the knowledge base describes a traffic situation in the following terms: object classes, topological data, spatial relations between objects, speed and distance information, road network data, traffic signs, background knowledge (e.g. one-way street). In this approach, recognized objects are stored in so-called object classes. Each object description contains a class name (e.g. truck) and its properties. Topological data describes the position of dynamic objects; spatial relationships between them can be, for instance, before/behind. Information about the road infrastructure, such as intersections or lanes, is stored in the road network data. The background knowledge includes traffic rules for specific lanes, one-way street definitions, etc.

Lattner et al. [91] define risk patterns as "abstract descriptions of situations where certain conditions hold". An example of such a pattern describing dynamic objects ahead of the autonomous vehicle in medium distance is:

$$\begin{aligned} &\text{maxDistHolds} (\mathit{medium\_distance}, \mathit{Actor}, \mathit{iv}, S, E) , \\ &\text{relationHolds} (\mathit{ahead}, \mathit{Actor}, \mathit{iv}, S, E) , \\ &\text{isMemberOfClass} (\mathit{Actor}, \mathit{dynamic\_object}) , \end{aligned} \tag{A.1}$$

where *Actor* is a variable for a dynamic object, S and E are start and end points of time, and iv denotes the intelligent vehicle.

Each pattern can be described formally using Allen's interval algebra[8], and complex patterns are composed of simpler ones. Lattner et al. claim

[8]Calculus for temporal reasoning.

that the abstract description of traffic situations using patterns allows the use of pattern matching algorithms to compare the current situation with predefined known patterns, in order to find dangerous situations. The proposed risk-assessment approach has been successfully tested in a simulation, using only a few standard traffic situations.

However, Lattner et. al. [90] also admit a problem which their approach is not able to solve:

> "*For non-standard or complex traffic situations (e.g. in cities) it is usually hard to formulate all possible aspects relevant for risk assessment.*"

A.2.2 EMS-Vision: Decision Making using Fuzzy Logic

The perception system EMS-Vision (Expectation-based Multi-focal Saccadic) has been developed at the University of the German Armed Forces, Munich. In addition to the perception functionality, the system also integrated a situation analysis and a decision making subsystem [29, 92, 93].

The hardware of the EMS-Vision system consisted of a multi-camera system mounted on a rotating platform. The rotating camera platform allowed to keep objects of interest in the field of view. In addition to controlling the camera platform, the system was able to make driving decisions.

The EMS-Vision system architecture contained three modules (Figure A.13):

- behavior decision for gaze and attention,
- behavior decision for locomotion (vehicle movement control),
- a central decision unit.

The purpose of the central decision unit was to activate and deactivate driving maneuvers[9], such as follow lane, change lane, enter lane.

The Central Decision module (CD) was based on a fuzzy logic algorithm using IF-THEN-rules which were stored in a central database. The Central Decision module accessed a so-called situation analysis unit and selected the IF-THEN-rules for the traffic situation accordingly. So-called situation aspects, containing fuzzy sets, information and processing logic of objects, obstacles, etc., were managed by the situation analysis unit.

The system was tested on the VaMoRs vehicle, an autonomous vehicle originally developed in 1987 for highway traffic. The vehicle was almost

[9]The term "capabilities" was used in the original publication [29].

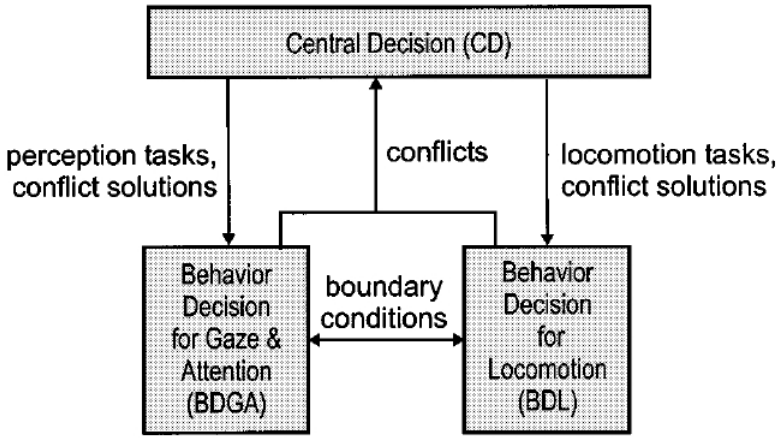


Figure A.13: The EMS decision making units [29].

identical to the vehicle VITA II, which is addressed in this chapter in section A.1.4.

A.2.3 Decision Making based on Optimal Control

Kelly and Stentz [94] propose an approach for solving conflicting decision situations based on optimal control. Assuming two simultaneously executing behaviors[10], for instance following a trajectory and avoiding obstacles, the simultaneous execution of both could lead to conflicting commands. In such a situation, they propose an arbitration algorithm which decides which behavior should be executed, or whether to execute a combination of both behaviors.

The decision making module is called "Optimal Control Goal Arbiter" (Figure A.14). Its role is to coordinate the execution of two sets of possibly conflicting behaviors: a set of hazard avoidance behaviors and a set of goal seeking behaviors.
The decision making problem is modeled as an optimal control problem, in which the goal seeking behavior is represented by a utility function under constraints. Constraints can be for instance hazard avoidance. In order to estimate the hazard constraint values, each point along a given trajectory is assessed with respect to the vehicle's safety.

The optimal control problem is defined as follows [94]:
Let $\vec{u}_i(t)$ be a candidate command, $\vec{x}_i(t)$ the response trajectory generated

[10]This thesis uses the term driving maneuvers instead, however following the usual definition that only one driving maneuver can be active at a time.

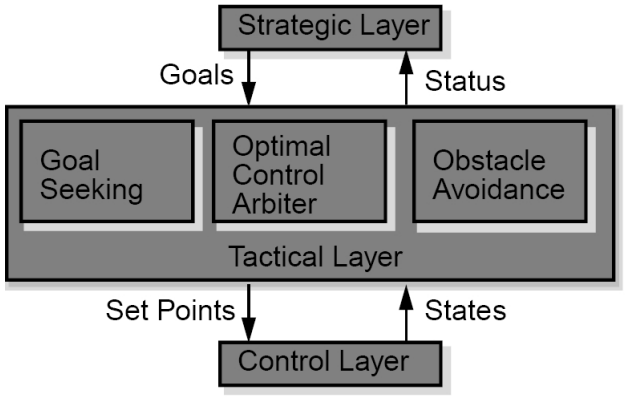


Figure A.14: Decision making architecture proposed by Kelly and Stentz [94].

from the nonlinear system dynamics. The system is modelled as:

$$\begin{array}{ll} \vec{x}' = f(\vec{x}, \vec{u}) & \text{System Dynamics} \\ \vec{g}(\vec{x}) = 0 & \text{Terrain Following} \end{array} \tag{A.2}$$

A hazard function h assesses every terrain point on a trajectory a hazard value for a specific hazard. This results in a hazard vector $\vec{y} = h(\vec{x})$. A terrain point is considered safe if its hazard vector is below a defined threshold: $\mid \vec{y} \mid < \vec{y}_{safe}$.
Based on the definitions above, the optimal control problem is defined as follows [94]:

$$\begin{array}{lll} \text{Minimize:} & L[\vec{x}_i(t)] = ||\vec{x}_i(t) - \vec{x}_{goal}(t)|| & \text{Goal Proximity functional} \\ \text{Subject to:} & \vec{x}' = f(\vec{x}, \vec{u}) & \text{System Dynamics} \\ & \vec{g}(\vec{x}) = 0 & \text{Terrain Following} \\ & \vec{y} = h(\vec{x}) & \text{Hazard Kinematics} \\ & |\vec{y}| < |\vec{y}_{safe}| & \text{Safety Constraint,} \end{array} \tag{A.3}$$

where the functional $L[\vec{x}_i(t)]$ specifies how close a trajectory follows the goal trajectory.

The proposed decision making approach has been tested using autonomous terrain vehicles and planetary exploration robots [94]. However, as the computation of every possible trajectory was too time consuming, only a defined region called detection zone was used for evaluation. This zone was restricted to the area in front of the vehicle, and even further restricted by leaving out a small region directly in front of it, where the vehicle was already committed to go. As a further restriction, the arbitration algorithm evaluated only ten steering angle commands per execution cycle and decided based on the

solution of the optimal control problem. During the tests, the autonomous vehicles were able to achieve speeds of up to 15km/h [94].

A.3 The DARPA Challenges

A.3.1 DARPA Grand Challenges 2004 and 2005

The US Defense Advanced Research Projects Agency (DARPA) organized the first DARPA Grand Challenge in 2004. With regard to the US Congress mandate to operate one third of the ground combat vehicles unmanned by the year 2015, DARPA organized the Grand Challenge 2004 race and offered US$1 million prize for the first vehicle able to travel autonomously 142 miles in desert terrain [27, 95].

However, at the first DARPA Challenge in 2004, none of the fifteen participating vehicles were able to finish the race. The longest travel distance was achieved by the Red Team (CMU) using a Hummer-based autonomous vehicle called Sandstorm. Sandstorm went off-course after 7.4 miles and spinning tires caught fire while trying to drive back on track[11] [95]. Many of the remaining vehicles left the track and got stuck soon after the start line or could not continue due to other technical problems.

The Grand Challenge race was repeated in 2005 with rules similar to those from the previous year 2004. This time, the desert track was 175 miles long and had to be completed in less than 10 hours. The prize was doubled to US$2 million.

Before the race, DARPA provided the teams nearly 3000 navigation points (i.e. global coordinates in longitudes and latitudes), the track boundaries and speed limits. Therefore, global path planning was done off-line before the beginning of the race [71]. During the race, only static obstacles had to be recognized and the vehicles were not required to overtake autonomously. If an autonomous vehicle was slower than another, the slower vehicle was manually stopped and later, after it was overtaken, it was restarted by supervising personnel.

Only 5 of the 23 vehicles were able to finish the DARPA Grand Challenge 2005 (Figure A.15). 6 These autonomous vehicles are further addressed in the following subsections.

[11] The same vehicle achieved the second place in the following DARPA Challenge in 2005.

(a) Stanley crossing the finish line [71]

(b)

(c) Sandstorm (2nd) [96]

(d) H1ghlander (3rd) [97]

(e) Kat-5 (4th) [98]

(f) TerraMax (5th) [99]

Figure A.15: The five autonomous vehicles which were able to complete the DARPA Grand Challenge 2005.

Stanley (1st place)

Stanford University was the leader in the development of the autonomous vehicle Stanley (Figure A.15.a, b), which has won the DARPA Grand Challenge

2005 [71, 100, 101].

The vehicle was based on a Volkswagen Touareg 4WD, and was equipped with a camera, five SICK LIDAR sensors, a GPS receiver and an inertial measurement unit. The main vehicle control functions, such as throttle, brakes, steering and gear shifting were electronically controllable. Sensors measured and transmitted the vehicle's speed and steering angle to computers over a CAN bus interface. The five LIDAR sensors scanned the front area of the vehicle horizontally at distances of up to 25 meters. An initial plan to use two additional RADAR sensors was rejected a short time before the race due to technical difficulties [71].

The computing system integrated six Pentium M computers running Linux, which were connected over Ethernet. However, only three computers were used during the race for the control software, the fourth was used for logging data, the remaining two were idle. One of the three computers running the control software was dedicated to video processing alone. Stanley's sensors were polled at frequencies up to 100Hz, and the vehicle control, such as brakes, steering, and throttle, was executed at frequencies up to 20Hz [71].

Stanley's control software and consisted of approximately 30 software processes, divided into the following six layers (Figure A.16):

- Sensor Interface layer,
- Perception layer,
- Planning & Control layer,
- User Interface layer,
- Vehicle Interface layer,
- Global Services layer.

The developers [71] state that all software processes were executed in parallel[12], each at its own execution cycle frequency, without inter-process synchronization. The reason for the asynchronous approach was to reduce the risk of deadlocks and undesired processing delays[13].

Stanley's sensor interface layer received and time stamped sensor data. Time stamps were used in order to merge data from different sensors without synchronizing the concurrent processes. The idea was to allow each process to run at its own execution cycle time without delays at synchronization points. As each sensor data package contained a timestamp, data from multiple

[12]Most likely they were executed concurrently and not really in parallel.

[13]Deadlocks occur when two or more concurrent processes wait for each other to finish or to release an exclusively accessible, shared resource. In such a situation both processes are blocked. Deadlocks can be prevented by careful resource allocation, avoided using suitable algorithms, or detected at runtime.

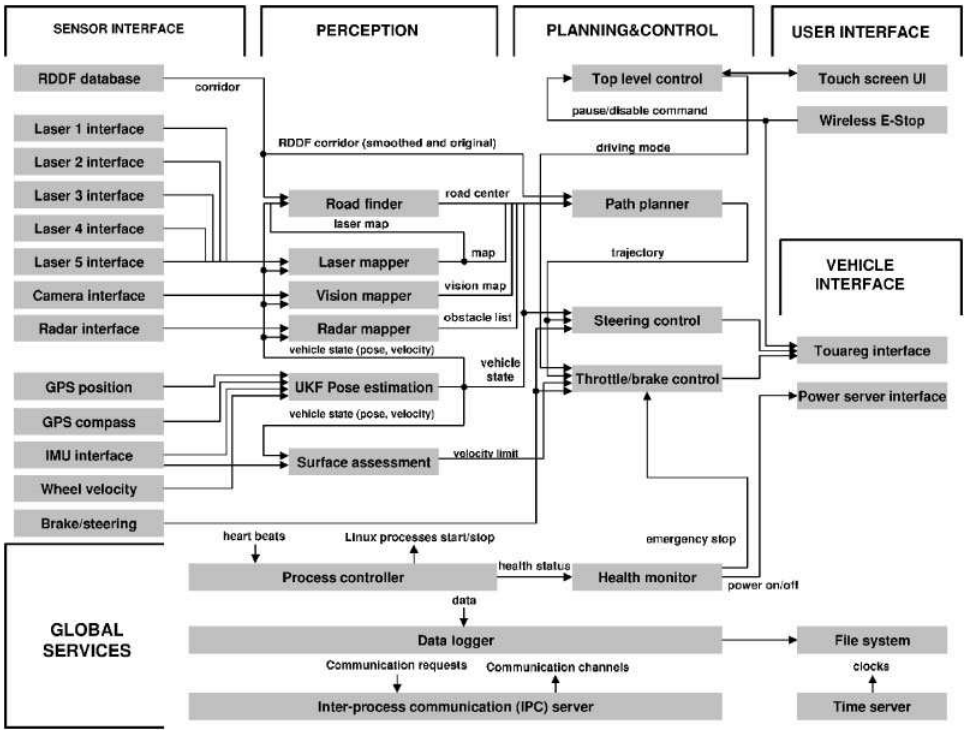


Figure A.16: Stanley's control software architecture [71].

sensors could be merged, although it was received and processed at different cycle times.

The perception layer received data from the sensor interface and generated "internal models". The UKF Pose estimation module determined the vehicle's position, speed and orientation using an Unscented Kalman Filter (UKF) technique[14]. This estimation was required for areas where GPS localization was not possible, for instance under bridges or in tunnels. For the position estimation, data from four sensors was processed: GPS, GPS compass, inertial measurement unit (IMU) and the wheel velocity. Without GPS localization, the UKF Pose estimation module was able to correctly estimate the vehicle's position for up to 2 minutes. During GPS failures, the vehicle's

[14]The Kalman filter [102] is a method widely used for tracking and estimation of linear models. The filter consists of two sets of equations: time update (predictor) equations and measurement (corrector) equations. The time update equations are used to obtain a priori estimates of the next time step, the measurement equations are used to incorporate new measurements into the a priori estimates [103]. The advantages of the unscented Kalman filter are a higher accuracy and a simpler implementation compared to an Extended Kalman filter, which performs a linearization before applying the traditional Kalman filter.

speed was reduced to 10 mph.

Basic vehicle control components, such as steering, brakes, and throttle control, were part of the Planning&Control layer, which also included a local path planner. The purpose of the local path planner was to keep the vehicle on track while avoiding static obstacles. The global path was generated offline, before the beginning of the race.

The vehicle's speed was determined based on "recommendations" from the path planner, the health monitor and the velocity recommender modules. For maximum safety, the actual velocity was set to the minimum of the three inputs. The health monitor module decreased the recommended velocity in the case of critical system or GPS failures. The velocity recommender module recommended a velocity based on the terrain slope and vibrations caused by the terrain roughness. After detecting high vehicle vibrations (a maximum shock), the speed was linearly reduced, and again linearly increased at low vibrations. The maximum shock parameter and the linear acceleration rate were determined by comparison to a human driving profile.

Stanley's top level control in the planning and control layer executed manual user commands triggered by a touch screen or an emergency stop remote control (so-called E-stop). The vehicle interface implemented the commands for the basic vehicle functions such as brakes, steering angle and throttle. Electrical power management, monitoring of subsystems, a time server, and data logging functions, were integrated into the global services layer.

The obstacle detection distance for the laser sensors was limited to 22 meters, which enabled a maximum safe speed of 25mph. In order to be able to increase the vehicle's maximum speed above 25mph, camera images were used for the classification of farther terrain. By projecting camera image areas of laser-analyzed driveable surfaces onto unknown terrain image areas, unknown regions exceeding the maximum laser distance were analyzed and classified as driveable or not driveable. However, computer vision alone was not sufficient to control the steering, as possible changes of colour of the terrain surface led to erroneous terrain classification [71].

Sandstorm, H1ghlander (2nd and 3rd place)

The Teams "Red" and "Red Too" developed two autonomous vehicles, Sandstorm (Figure A.15.c) and H1ghlander (Figure A.15.d). Both vehicles were similarly equipped, were running the same control software, and were based on the HMMWV (Hummer), an off-road vehicle designed for military purposes. The Sandstorm vehicle was a 1986 AM General M998 HMMWV, while H1ghlander was a modified 1999 GM H1 Hummer Sport Utility Truck [96,97].

The vehicles were equipped with 11 computers running Linux, 7 LIDAR sensors (1 sensor for 150m range, 6 sensors for 50m range), a 360 degrees RADAR, a video camera, a GPS/INS sensor and a drive-by-wire system. Using a gimbal, the long range LIDAR sensor could be swept 180 degrees, which (including the 60 degrees field of the sensor) resulted in a 240 degrees field of view.

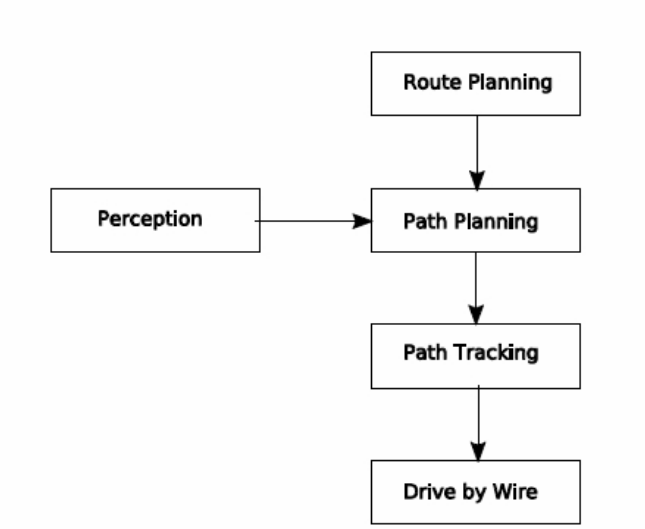


Figure A.17: Sandstorm's and H1ghlander's control software architecture [97].

The vehicle control system architecture consisted of five modules (Figure A.17): Perception, Route Planning, Path Planning, Path Tracking, and Drive by Wire.

The global route was created by the Route Planner, however offline, before the race. The Red Team used additional satellite images and other topography data to generate the global path. The local path planner used the A* search algorithm[15] in order to find the optimal path around obstacles. The path planner used for H1ghlander and Sandstorm created a set of alternative paths on an internal map by fusing RADAR and LIDAR data.

[15] A* is a heuristic graph search algorithm. It requires a heuristic function which assigns weights (costs) to the graph edges. The algorithm is complete, meaning it always finds a solution if one exists. The direction of search, and therefore also the algorithm's performance, highly depend on the chosen heuristic function. In a worst case, the complexity is exponential. A* and its variations has been widely used in complex, but static search applications, such as path planning for industrial manipulators with many degrees of freedom [104, 105]. In dynamic applications, the algorithm has a major drawback, as it requires a restart from the beginning in the case that replanning is required.

The A* heuristic function was defined based on terrain costs, assigning each point a value which reflected its relative height to the neighboring points. Lower located terrain areas were therefore preferred by the search algorithm to higher located areas.

The vehicle's velocity and steering were controlled using classic PID (proportional-integral-derivative) or PI (proportional-integral) controllers derived from Simulink[16] models.

In the case of getting stuck in the terrain (detected by wheel speed differences), or if the heading deviation exceeded 30 degrees, the vehicle reversed 10m along the path or in an arc. A so-called "Vehicle Health Management" system monitored the processes and the eleven computers. Each process sent a health message containing its start time, last run time, cycle time, initialization time and a heart beat counter. Using this information, the health monitor could detect blocked processes restart them if necessary. Process messages were sent using the UDP protocol[17]. The computers were monitored using the Ping protocol[18] In the case that one of the computers did not respond, the vehicle was stopped and the computer was restarted [96, 97].

Kat-5 (4th place)

"Team Gray", achieved the 4th place in the Grand Challenge 2005 with their autonomous vehicle Kat-5 (Figure A.15.e) [98, 100, 106].

Kat-5 was a standard 2005 Ford Escape Hybrid equipped with a drive-by-wire system, four LIDAR sensors, two GPS/INS units and stereographic cameras. One LIDAR sensor ranged up to 80 meters, two ranged up to 50 meters, and the last LIDAR was used for terrain analysis. The drive-by-wire system was originally designed for conventional cars to help handicapped drivers. It consisted of actuators attached to the conventional steering, brakes, throttle pedals and transmission lever.

For localization, the team used an "Oxford Technical Solutions RT3000" GPS/INS unit, which integrated a GPS receiver and an INS (Inertial Navigation System). Using a Kalman filter and inputs from the wheel speed sensors, the INS sensor was able to estimate the position in areas where GPS localization was not possible (e.g. tunnels, under bridges, etc.). Four computers executed the control system software tasks: two marine PCs running Linux and two Mac Mini computers running OS X. The Mac Mini computers operated redundantly and were dedicated to path planning tasks. One PC

[16]Matlab modeling library.

[17]Universal Datagram Protocol.

[18]Ping is a simple network protocol to check the reachability of a computer over the network.

was processing sensor data, the second PC was used for navigation. The control system software was implemented in Java.

Figure A.18 shows a block diagram of Kat-5's control software architecture. Several redundant hardware units were integrated, allowing the vehicle to continue the race in the case of hardware failures. The Mac Mini computers operated redundantly, as well as the GPS units, the obstacle avoidance approach and the drive-by-wire system.

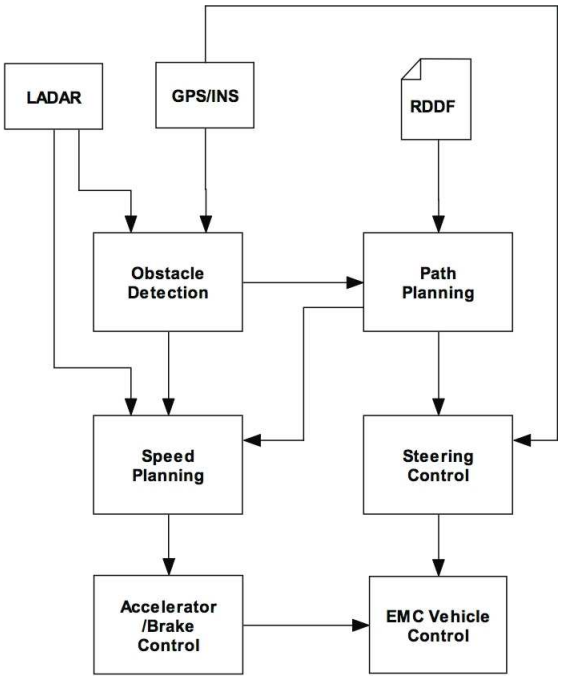


Figure A.18: Kat-5's control software architecture [106].

The path planner was based on cubic B-splines[19], which, in a first step, created a smooth path trajectory in the middle of the given RDDF corridor. In following steps, the path planner repeatedly adjusted this original trajectory, eventually resulting in a driveable and obstacle-free path. The developers took into account that *"the path planning algorithms might occasionally time out"* and reduced the vehicle's speed to *"3 mph for safety reasons until the algorithms had a chance to recover"* [98]. During the race, a programming error related to path planner timeouts unnecessarily slowed down the vehicle on wide, and easy to drive path segments.

The vehicle's speed was defined for each trajectory point based on the trajectory curvature in the current and following navigation points. The

[19]Mathematical functions defined piecewise by polynomials.

obstacle detection method was mainly based on LIDAR sensor data, but integrated stereographic camera information as well. After fusing information from both systems, so-called "confidence levels" were generated for each obstacle. Confidence levels indicated the probability that a detected obstacle was real. Similar to the vehicle Stanley, Kat-5 used timestamps for the LIDAR data, however for other reasons than Stanley. Kat-5's LIDAR sensors occasionally delivered erroneous data, leading to the detection of not existing obstacles. In order to avoid such errors, the LIDAR scans were kept in memory only for a defined period of time. Timestamps were used to find such expired data in the memory [98].

TerraMax (5th place)

The autonomous vehicle TerraMax (Figure A.15.f) was the last of the five vehicles able to complete the Grand Challenge in 2005 [99]. Although the vehicle needed 12h 51min, exceeding the DARPA requirement to complete the race in less than 10 hours, it was awarded the 5th place in the 2005 Grand Challenge [100].

The vehicle TerraMax was an off-road Oshkosh truck equipped with a vision system, a single-plane LIDAR, a multi-plane LIDAR, 2 Oxford Technical Solutions GPS/INS RT3100 sensors, Trimble GPS sensors, and a drive-by-wire system. The vision system consisted of 3 cameras, mounted on the front of the vehicle. Vision was used for obstacle detection and the detection of the track [107].

TerraMax was equipped with two redundant GPS/INS sensors. In normal operation, data from both sensors was averaged, but in the case that one failed, the second sensor was used alone. Averaging the position of both sensors was an attempt to improve the positioning accuracy. One GPS/INS sensor was configured to use DGPS (Differential GPS), the other used WAAS (Wide Area Augmentation System). DGPS and WAAS use fixed ground stations to improve the GPS accuracy. In the case of GPS failure, the INS (Inertial Navigation System) part of the GPS/INS sensor continued to deliver position estimates based on wheel speed inputs.

TerraMax's control system, called "Intelligent Vehicle Management System (iVMS)", consisted of the following independent applications communicating over Ethernet (Figure A.19):

- Vehicle Control,
- Real Time Path Planner,
- Obstacle Detection,
- Behavior Management,
- Navigation.

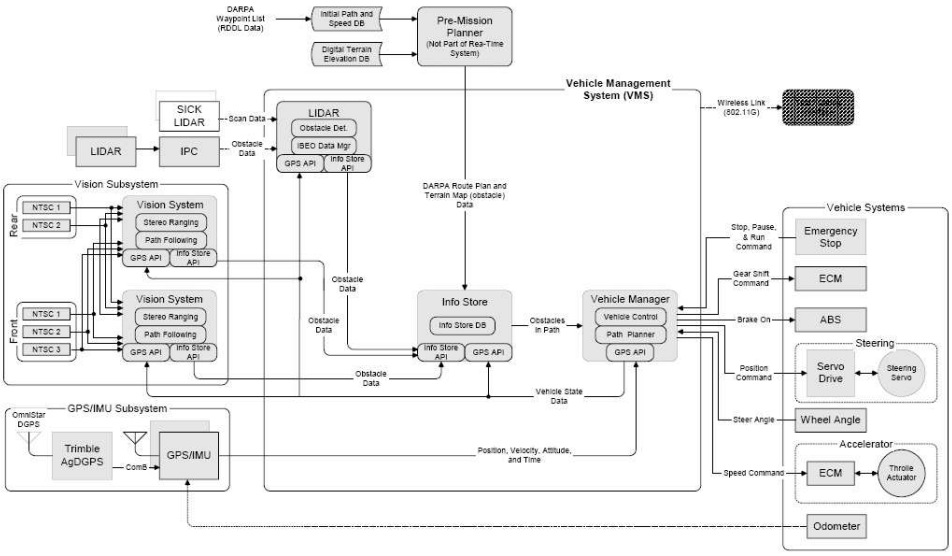


Figure A.19: TerraMax's control software architecture [72].

The Vehicle Control application controlled the vehicle's throttle, brakes, steering and transmission. The Real Time Path Planner application computed an optimal path using "a tree-based algorithm" [99]. In each planning cycle 2000 path candidates were generated, the optimum path regarding distance to the route centre, route curvature or detected obstacles, was chosen.

The Obstacle Detection application used LIDAR sensor data and the vision system. The vehicle's position was determined by the Navigation application using GPS data.

The Behavior Management application decided about the so-called "mode" of the vehicle. Each behavior was modeled as a state machine reacting to events. The behavior management application's input came from the path planner, the obstacle database, the navigation sensors and the vehicle interface. As a response to defined condition changes, different behaviors were executed. The behavior management detected and responded to the following conditions:

- E-stop (emergency stop),
- No valid path ahead,
- Obstacle behind while backing up,
- Narrow tunnel,
- Large course change requiring a backup maneuver,
- Stuck between obstacles.

The E-stop was a wireless emergency stop remote controller used to stop and restart the autonomous vehicles during the race. The "no valid path ahead" condition stopped the vehicle, waited and eventually backed up if no valid path was found. The "obstacle behind" condition led to a forward movement of the vehicle, the "large course change" condition turned the vehicle around. The "narrow tunnel" condition disabled the vision system and used LIDAR data alone. The "stuck between obstacles" condition tried to find a way out, but eventually ignored any obstacles and moved forward.

One of the objectives was, besides winning the Grand Challenge race, the development of a vehicle for military purposes in desert environments. This objective is apparent in the behavior implementation: in the case of getting stuck between obstacles, the vehicle ignored any obstacles after several unsuccessful attempts to avoid them. During the race, the vehicle collided with a tunnel barrier and still continued the race, although one of the sensors was displaced [99].

The System Management function, a module of the control software, was responsible for stopping and restarting control applications in the case of abnormal operation. This function led to 52 automatic vehicle stops during the race.

Appendix B

Petri Nets

B.1 Petri Net Basics

The following definition of a Petri net is adopted from [65]:

Definition B.1.1. *Petri Net*
A *Petri net structure* is a 4-tuple $C = (P,\ T,\ I,\ O)$, where

$P = \{p_1, p_2, ..., p_n\},\ n \geq 0$ is a finite set of *places*,

$T = \{t_1, t_2, ..., t_m\},\ m \geq 0$ is a finite set of *transitions*,

$P \cap T = \varnothing$ (P and T are disjoint),

$I : T \rightarrow P^{\infty}$ is the *input function*, and

$O : T \rightarrow P^{\infty}$ is the *output function*.

The input functions I and O as defined in definition B.1.1 map transitions to bags[1] of places. A place p_i is an *output place* of a transition t_j if $p_i \in O(t_j)$. A place p_i is an *input place* of a transition t_j if $p_i \in I(t_j)$.

Definition B.1.2. *Multiplicity* $\#(p_i, I(t_j)$, $\#(p_i, O(t_j)$.
The *multiplicity* $\#(p_i, I(t_j)$ of an *input* place p_i for a transition t_j is the number of occurrences of the place in the transition's input place. The *multiplicity* $\#(p_i, O(t_j)$ of an *output* place p_i for a transition t_j is the number of occurrences of the place in the transition's output place [65].

Based on the multiplicity definition B.1.2, the input and output functions I and O are extended to [65]:

[1] Bags are extensions of sets, allowing *multiple* occurrences of elements [65].

$$I : P \to T^{\infty}$$
$$O : P \to T^{\infty}$$

such that

$$\#(t_j, I(p_i) = \#(p_i, O(t_j)$$
$$\#(t_j, O(p_i) = \#(p_i, I(t_j)$$

A Petri net is graphically represented as a bipartite[2] directed multigraph[3], where the places are represented by circles and transitions are represented by bars. Multiple inputs or outputs are represented by multiple edges (therefore multigraph). Petri net graphs are bipartite, as they can be divided into a set of places and a set of transitions, such that each edge is directed from one set to the other.

Example B.1.1. *Figure B.1 shows a very simple example of a Petri net graph. This Petri net is represented as a 4-tuple* $C = (P,\ T,\ I,\ O)$ *where:*

- $P = \{p_1, p_2\}$ *(places)*
- $T = \{t_1\}$ *(transition)*
- $I(t_1) = \{p_1\}$, $O(t_1) = \{p_2\}$

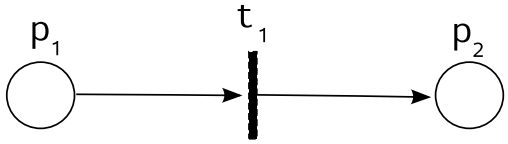


Figure B.1: A simple Petri net graph.

Definition B.1.3. *Marking* μ *[65]*
A Petri net *marking* μ is a function $\mu : P \to \mathbb{N}$ which maps the set of places to the set of nonnegative integers. The Marking μ can be represented as a vector $\mu = (\mu_1, \mu_2, ..., \mu_n)$, where n is the number of places ($n = |P|$) and $\mu_i \in \mathbb{N}, i = 1, 2, ..., n$. As a function, the vector representation corresponds to: $\mu(p_i) = \mu_i$.

Definition B.1.4. *Marked Petri net*
A *marked* Petri net $M = (C, \mu)$ is a Petri net $C = (P, T, I, O)$ and a marking μ [65].

[2]The set of vertices V can be partitioned into two sets V_1 and V_2 such that all edges go between V_1 and V_2 [108].

[3]A multigraph can have multiple edges and self-loops [108]

Definition B.1.5. *Enabled Transition [65]*
A transition $t_j \in T$ in marked Petri net $C = (P, T, I, O)$ is *enabled* if for all $p_i \in P$:

$$\mu(p_i) \geq \#(p_i, I(t_j))$$

Definition B.1.6. *Execution rules (Firing Transition) [65]*
A transition $t_j \in T$ in marked Petri net $C = (P, T, I, O)$ with marking μ may fire whenever it is *enabled*. The new marking μ' after a transition t_j fires is:

$$\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j) + \#(p_i, O(t_j))$$

A firing transition removes all of its enabling tokens from the input places and inserts a marking for each arc in the output places.

B.2 Modeling of Systems using Petri Nets

A large variety of systems can be modeled using Petri nets, for example computer hardware systems, chemical systems, traffic control systems, but also, as applied in this work, software systems. While Petri nets are not limited to modeling of certain system types or system features, this technique is most adequate for the design and analysis of *discrete* systems with interacting components. The original work of Carl Adam Petri focused on "*the description of causal relationships between events*" [65].
The most important benefits of Petri nets for the application in decision making for autonomous city vehicles are [109]:

- the ability to model *concurrency*,
- the ability to model systems at multiple *levels of abstraction*,
- the possibility to *verify* the system.

The first step for this approach is to identify the events and conditions in the system. *Events* in this context are "*actions which take place in the system*", while *conditions* are "*predicates or logical descriptions of the state of the system*" [65]. Conditions are further divided into *preconditions* and *postconditions*. Preconditions are necessary for an event, while postconditions hold after the occurrence of an event. In a Petri net model, conditions are modeled as places and events as transitions. The firing of a transition represents the occurrence of an event.

Appendix C

Future Potentials and Current Limitations of V2V and V2I Communication Networks

One of the major remaining challenges for autonomous city vehicles is the accurate and reliable vehicle localization in urban environments. The currently used Differential GPS (DGPS) technology offers very accurate information, however only when the vehicle is able to receive GPS satellite signals, and, additionally, the position correction signals from a stationary beacon. In urban environments, for example between high buildings or under bridges, the direct reception of satellite signals is not reliable. Therefore, Inertial Navigation Systems (INS) are often used in combination with DGPS, which allow the estimation of position and heading based on inertial measurements and vehicle velocity. However, the accuracy of INS-based estimations decrease within a very short period of time, making such systems useful only for minutes after the GPS satellite reception is lost.

Future communication networks could provide a solution to the localization problem by enabling the vehicles to receive their position on the road from the road infrastructure. Road infrastructure sensors could be used, which detect autonomous vehicles, and inform them about their current position. Additionally, autonomous vehicles could receive other close-by vehicles' position information which have GPS reception, and, knowing the distance to these vehicles from on-board sensors, calculate their own position.

Another major challenge is the autonomous vehicle's ability to reliably recognize relevant traffic features, such as traffic signs, intersections, other vehicles, and pedestrians. While currently used LIDAR and RADAR sensors are able to provide very accurate information regarding the distance and velocity of obstacles, they are not able to recognize the type of obstacle. On

the other side, in their current state of development, computer vision systems do not seem to provide the required accuracy and reliability in bad weather and light conditions.

As demonstrated in the experimental tests, V2V communication can be used to improve the autonomous vehicles' perception capabilities of communication enabled vehicles. In the same way, the road infrastructure network could provide relevant information, for example about traffic signs and intersections.

Since today the majority of people already carry communication devices, such as mobile phones, future networks could use communication with such devices to improve, in addition to vehicle's on-board sensors, the recognition of pedestrians.

While the currently available wireless communication technology satisfies the needs for mobile non-safety-critical applications such as speech, email, web surfing, entertainment, etc., the currently available wireless networks are not sufficient for safety-critical applications, such as V2V and V2I communication for autonomous vehicles.

The most critical current limitations for wireless V2V and V2I communication networks are:

- Low communication reliability,
- Unsatisfactory network reachability,
- Unsatisfactory real-time performance,
- Inadequate network security.

Autonomous city vehicles are safety-critical systems. Therefore, if communication is used for purposes which can affect their safe operation, such networks need to guarantee reliability, reachability, fulfillment of real-time requirements, as well as network security requirements.

While occasional communication dropouts are acceptable for non-safety-critical applications, V2V/V2I networks require a very high level of communication reliability. Furthermore, such networks need to guarantee reachability everywhere within the network area, even close to high-voltage power lines, or in tunnels. The worst-case communication times need to be guaranteed within specified real-time limits, regardless of the number of communicating vehicles, amount of transmitted data, or network load.

Network security is another major challenge for future autonomous vehicle communication networks. On such networks, security breaches could have a devastating impact, causing major traffic delays, or, in worst-case scenarios, enable network intruders to take over control over autonomous vehicles.

This section has been published in the Proceedings of the 3rd IEEE International Symposium on Wireless Vehicular Communications: IEEE WiVEC 2010 [54], and in the Proceedings of the IFIP International Conference Network of the Future [78].

Appendix D

Implementation Details

D.1 Graphical User Interfaces

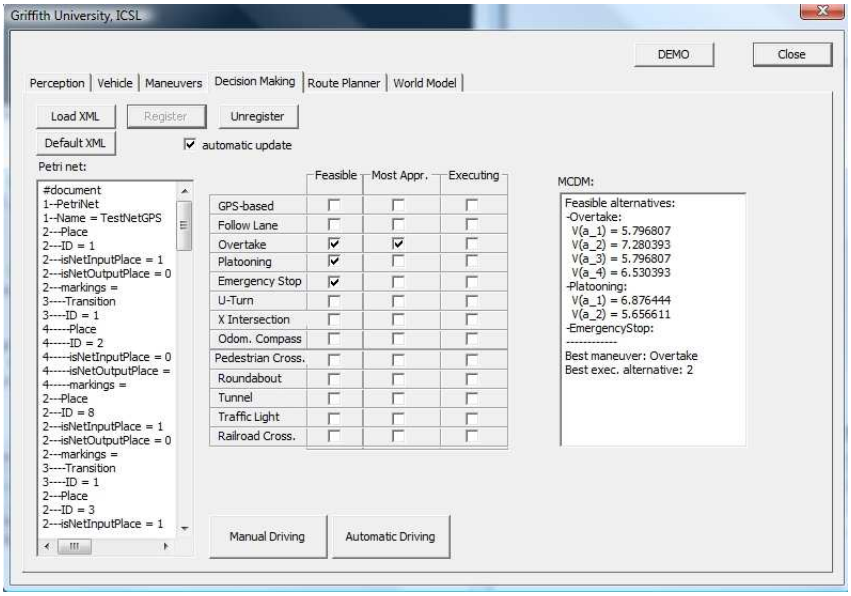


Figure D.1: The Decision Making GUI provides the functionality for loading the decision making Petri net (Stage 1), for registering the Decision Making process as an observer of the World Model, for starting and stopping autonomous driving, as well as supervising in real-time the current decision making result.

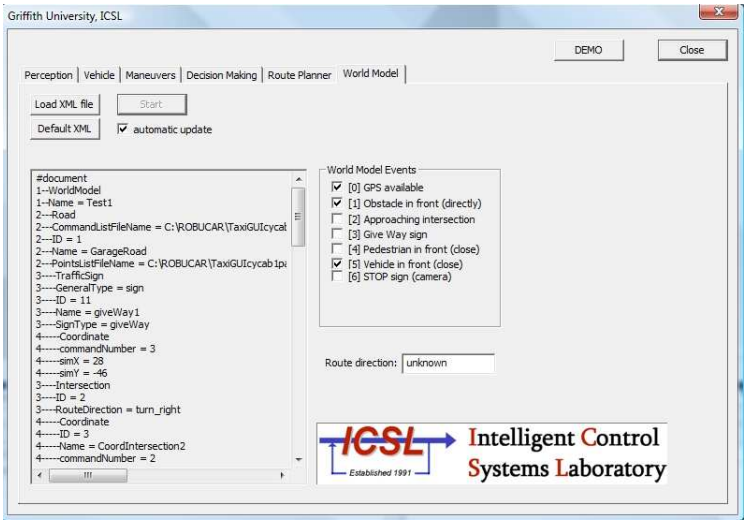


Figure D.2: The World Model GUI provides the functionality for loading the a priori information from an XML file, for starting and stopping the World Model updating process, and for analysing in real-time the status of World Model Events. This GUI also displays the path planner indication.

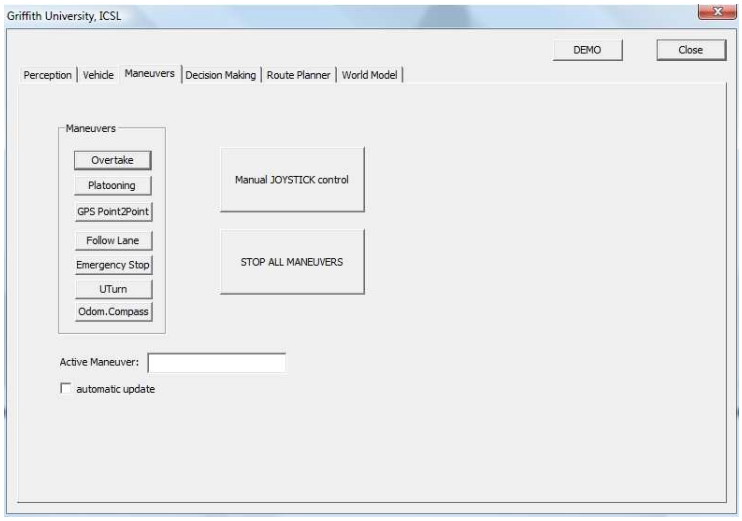


Figure D.3: The Driving Maneuvers GUI provides the functionality for manually starting and stopping driving maneuvers. Each driving maneuver opens its own graphical user interface, which allows to specify driving maneuver parameters. This GUI can also be used to drive the vehicle manually using a USB joystick.

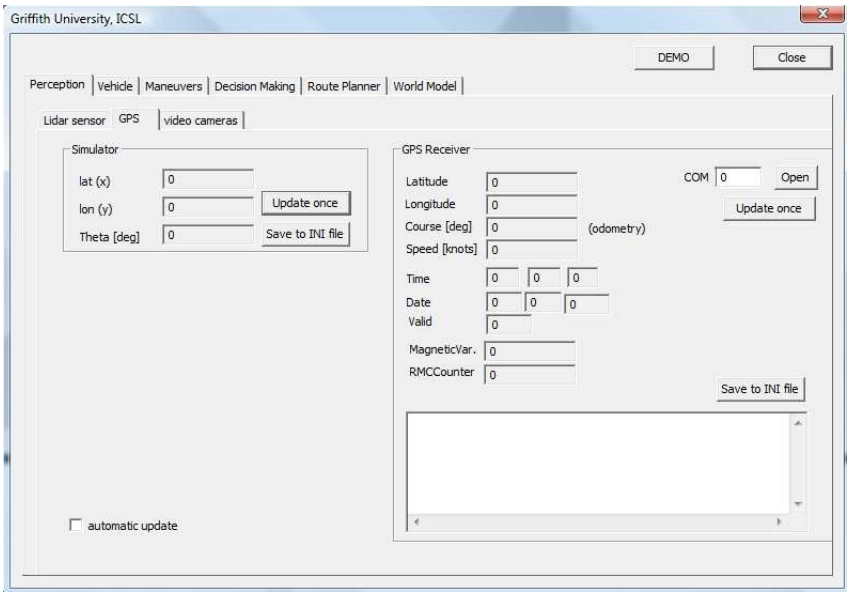


Figure D.4: The GPS receiver GUI displays the currently received GPS data. The GUI contains fields for both the simulated data, and for the real GPS receiver.

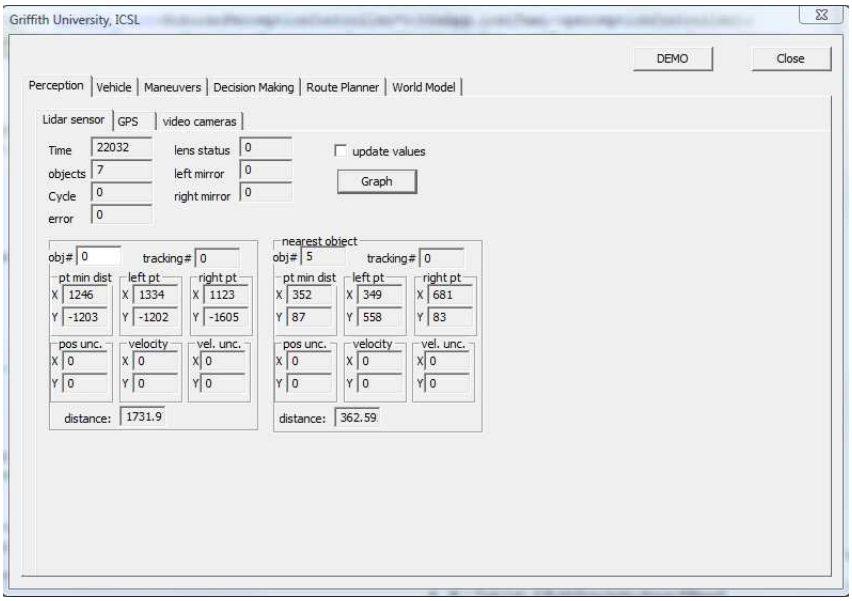


Figure D.5: The LIDAR sensor GUI shows in real-time data obtained from the LIDAR sensor. The same GUI is used for both simulator and real sensor.

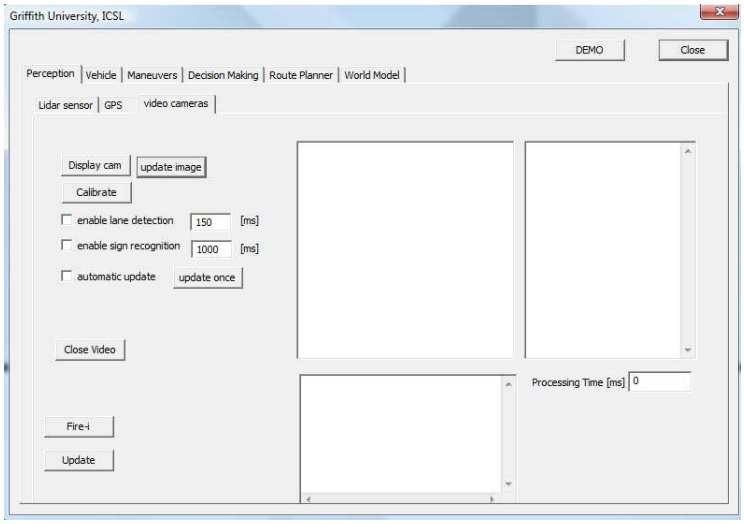


Figure D.6: The Video camera GUI shows in real-time the data obtained from the two video cameras, and, if enabled, the results of the traffic lane recognition algorithms. This GUI is only available for the real vehicle, since the simulator does not provide video camera simulation.

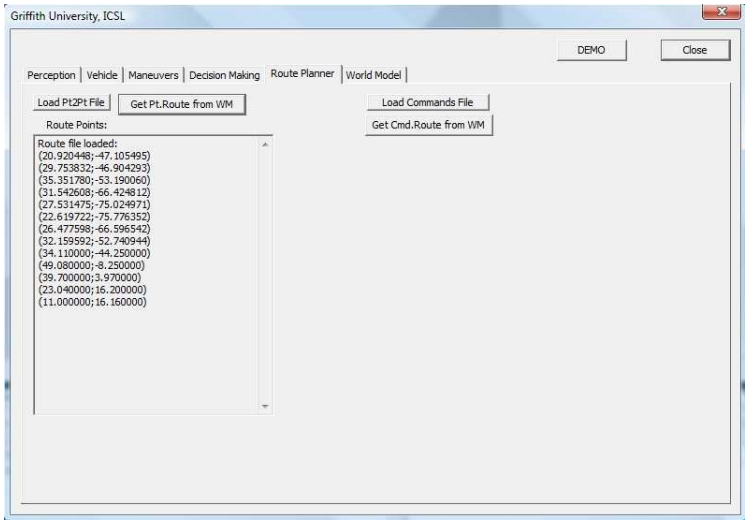


Figure D.7: The Route Planner GUI provides the functionality for loading GPS navigation points. The list of points can be loaded from both World Model (included in the a priori information), or from an additional file containing only GPS navigation points.

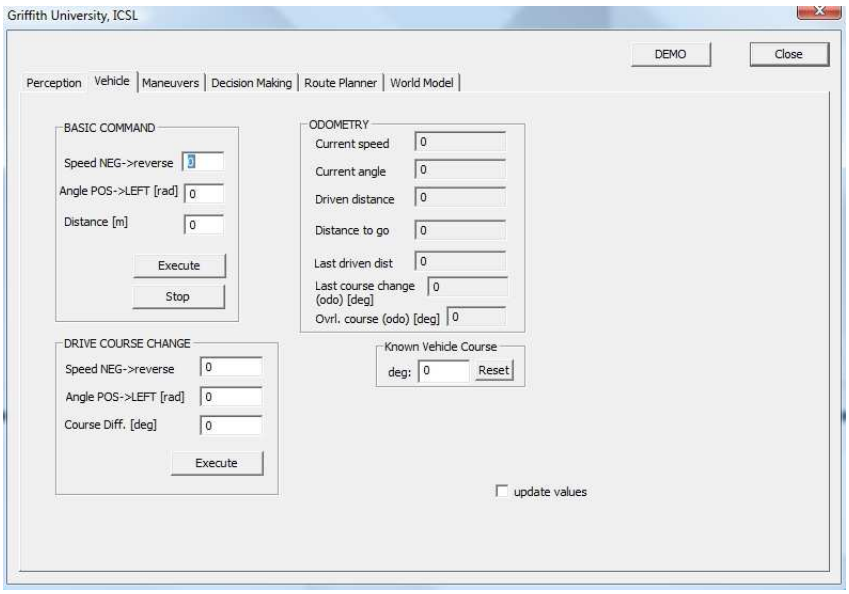


Figure D.8: The Vehicle control GUI provides the functionality for basic vehicle movements (e.g. driving a specified distance at a specified speed and steering angle, and for displaying the odometry data in real-time. This GUI is used for testing and debugging purposes.

Appendix E

Decision Making Computational Steps

The following steps are required for the initialization of the decision making subsystem:

1. Specification of driving maneuvers and execution alternatives:

$$\begin{aligned} M^1 &= \{M_1^1, M_2^1, .., M_j^1\} \\ M^2 &= \{M_1^2, M_2^2, .., M_k^2\} \\ &\vdots \\ M^n &= \{M_1^n, M_2^n, .., M_l^n\}, \end{aligned} \tag{E.1}$$

   where n denotes the number of driving maneuvers, and j, k, l the number of execution alternatives for the maneuvers M^1, M^2, and M^n respectively.

   The set of alternatives A contains all execution alternatives of all n driving maneuvers:

$$A = \bigcup_{m=1}^{n} M^m = \{M_1^1, M_2^1, .., M_j^1, M_1^2, .., M_k^2, .., M_1^n, ..M_l^n\} \tag{E.2}$$

2. Specification of the Petri net structure, which determines the feasible driving maneuvers with respect to the World Model events and route planner indication.

3. Specification of measurable attributes for each objective on the lowest hierarchy level:

$$\{attr_1, attr_2, .., attr_p\}, p \in \mathbb{N} \tag{E.3}$$

4. Specification of attribute *weights* $w_j \in \mathbb{R}$, which assign each attribute a level of importance.

Assuming the traffic situation in Figure E.1, the following computational steps are performed in each decision making cycle:

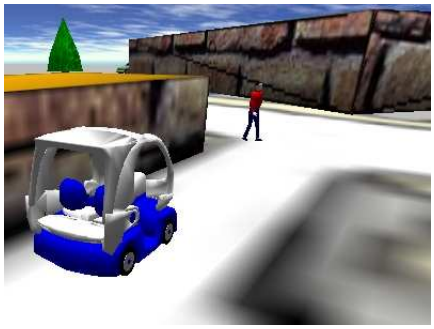


Figure E.1: Approaching an intersection (repetition of Sit. 2 in Table 7.1).

1. The Petri net input places of the first decision making stage are marked according to the World Model Events and Route Planner, the Petri net is executed, and the feasible driving maneuvers are obtained (Algorithm E.1):

$$f_{feasible} : M_{all} \times W_{events} \times D_{route} \rightarrow \{0, 1\} \tag{E.4}$$

The set of feasible alternatives is:

$$A = \{a_1, a_2, .., a_q\}, \ (q = j + k + .. + l) \tag{E.5}$$

Algorithm E.1 calculateFeasibleDrivingManeuvers

Require: array of World Model Events
Require: path planner direction

1: clear Petri net places
2: mark Petri net with World Model Events
3: mark Petri net with path planner direction
4: feasibleManeuvers ← Petri net output markings
5: **return** feasibleManeuvers

In this example, it is assumed that the World Model notifies the Real-Time Decision Making subsystem about the occurrence of the following events:

- approaching intersection
- pedestrian detected

Based on these World Model events, the following driving maneuvers are determined to be feasible (Algorithm E.1):

- a1 = GPS_Point2Point fast
- a2 = GPS_Point2Point medium speed
- a3 = GPS_Point2Point slow
- a4 = Approach intersection at high speed
- a5 = Approach intersection at slow speed

2. The second decision making stage calculates the most appropriate decision alternative using MCDM. We define the *value* of an alternative a_i as follows (Algorithm E.2):

$$V(a_i) := \sum_{j=1}^{p} w_j f_j(a_i), \tag{E.6}$$

where p denotes the number of attributes.

Algorithm E.2 calculateMostAppropriateDrivingManeuver

Require: feasible driving maneuvers (algorithm E.1)
Require: MCDM utility functions $f_j(a_i)$, attribute weights w_j

1: feasibleManeuvers ← calculateFeasibleDrivingManeuvers
2: **for** each feasibleManeuver a_i **do**
3: calculate value function $V(a_i) := \sum_{j=1}^{p} w_j f_j(a_i)$ (equation E.6)
4: **end for**
5: maximumValueManeuver ← $max(V(a_i))$
6: **return** maximumValueManeuver

The following results are obtained for the second stage, when applying the Attribute Weight Set 1 (5.0;4.5;4.0;3.5;3.0;2.5;3.0;3.5;4.0;4.5;5.0) (Fig. 7.5):

$$
\begin{aligned}
V(a_1) &= 5.0 * 0.5 + 4.5 * 0.5 + 4.0 * 0.131 * 0.25 \\
&\quad +3.5 * 0.131 * 0.25 + 3.0 * 0.131 * 0.25 \\
&\quad +2.5 * 0.131 * 0.25 + 3.0 * 0.25 + 3.5 * 0.5 \\
&\quad +4.0 * 1.0 + 4.5 * 1.0 + 5.0 * 1.0 \\
&= 21.18
\end{aligned}
$$

$$
\begin{aligned}
V(a_2) &= 5.0 * 0.5 + 4.5 * 0.5 + 4.0 * 0.131 * 0.75 \\
&\quad +3.5 * 0.131 * 0.75 + 3.0 * 0.131 * 0.75 \\
&\quad +2.5 * 0.131 * 0.75 + 3.0 * 0.5 + 3.5 * 0.5 \\
&\quad +4.0 * 0.5 + 4.5 * 0.5 + 5.0 * 0.5 \\
&= 16.03
\end{aligned}
$$

$$
\begin{aligned}
V(a_3) &= 5.0 * 0.5 + 4.5 * 0.5 + 4.0 * 0.131 * 1.0 \\
&\quad +3.5 * 0.131 * 1.0 + 3.0 * 0.131 * 1.0 \\
&\quad +2.5 * 0.131 * 0.5 + 3.0 * 0.5 + 3.5 * 1.0 \\
&\quad +4.0 * 0.5 + 4.5 * 0.25 + 5.0 * 0.25 \\
&= 15.66
\end{aligned}
$$

$$
\begin{aligned}
V(a_4) &= 5.0 * 0.5 + 4.5 * 0.5 + 4.0 * 0.131 * 0.5 \\
&\quad +3.5 * 0.131 * 0.5 + 3.0 * 0.131 * 0.5 \\
&\quad +2.5 * 0.131 * 0.5 + 3.0 * 0.5 + 3.5 * 0.5 \\
&\quad +4.0 * 0.5 + 4.5 * 0.5 + 5.0 * 1.0 \\
&= 18.10
\end{aligned}
$$

$$
\begin{aligned}
V(a_5) &= 5.0 * 0.5 + 4.5 * 0.5 + 4.0 * 0.131 * 0.25 \\
&\quad +3.5 * 0.131 * 0.25 + 3.0 * 0.131 * 0.25 \\
&\quad +2.5 * 0.131 * 0.5 + 3.0 * 0.5 + 3.5 * 1.0 \\
&\quad +4 * 0.5 + 4.5 * 1.0 + 5.0 * 1.0 \\
&= 21.76
\end{aligned}
$$

3. The decision alternative with the highest value $V(ai)$ is chosen for execution. In this example, the decision alternative with the highest

value among all alternatives is a_5, with $V(a_5) = 21.76$. Therefore, the decision is made to approach the intersection at slow speed (Figure E.3). Figure E.3 summarizes the entire procedure.

In the current prototype implementation, the weight factors are assigned with the objective of minimizing waiting times by keeping the vehicle moving towards its destination, whenever it is safe to do so. Therefore, in the shown example, the decision making subsystem does not stop the vehicle when the pedestrian is detected, but continues driving towards the intersection at slow speed, as long as the pedestrian is at a safe distance from the vehicle. However, when closer obstacles are detected, the vehicle is stopped immediately (e.g. Situation 4 in Table 7.1).

As mentioned in Section 8.2, the problem of defining a set of driving objectives and attributes, as well as appropriate attribute weight factors, remains to be addressed in the future by transportation system experts.

Having made the decision to activate the driving maneuver for crossing the intersection at slow speed, the execution of this driving maneuver is started, which then sends the appropriate speed and steering angle commands to the Low-Level Vehicle Control subsystem (Figure E.2).

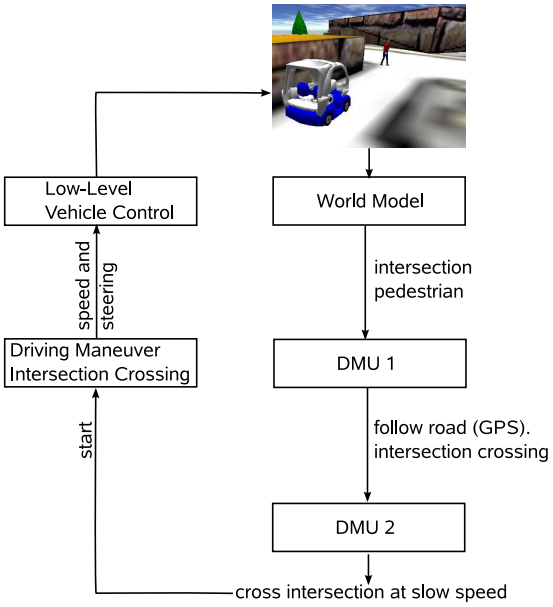


Figure E.2: Decision steps for the intersection crossing situation.

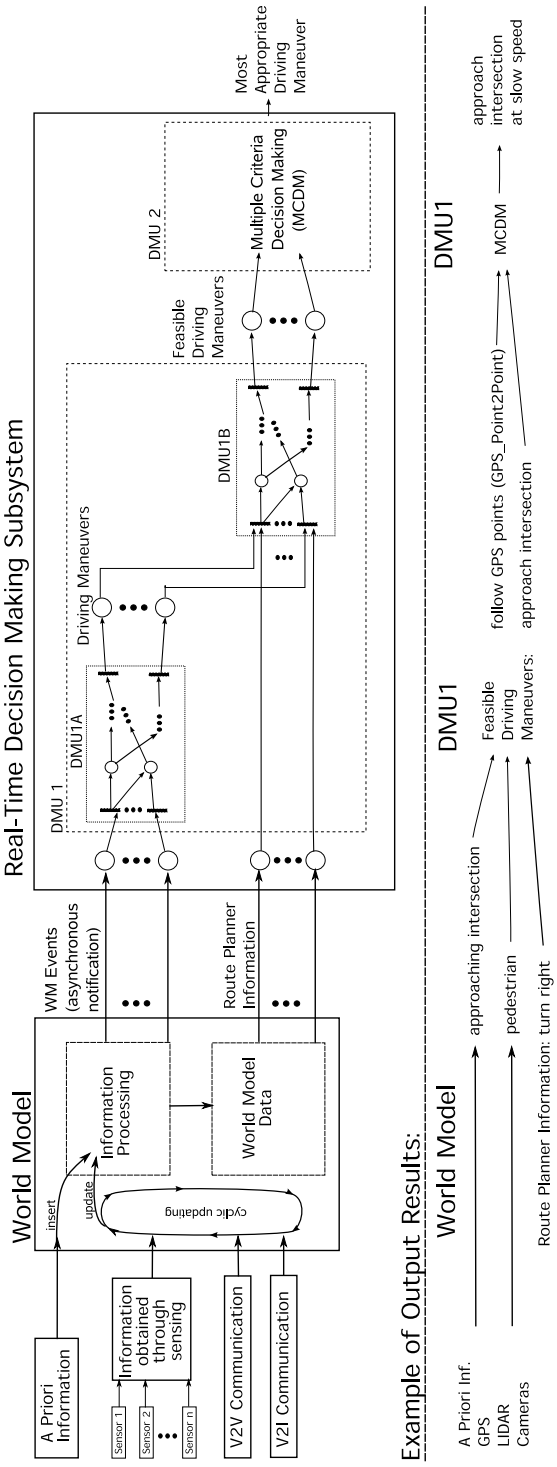


Figure E.3: Overview on the World Model's and the Real-Time Decision Making Subsystem's operational dynamics and example of intermediate results.

Bibliography

[1] J. Baber, J. Kolodko, T. Noel, M. Parent, and L. Vlacic, "Cooperative autonomous driving - intelligent vehicles sharing city roads," *IEEE Robotics & Automation Magazine*, March 2005 2005.

[2] P. L. McCarney, "World Urban Forum III - An International UN-HABITAT Event On Urban Sustainability, Background paper," World Urban Forum, Tech. Rep., 2006.

[3] M. Parent, "New technologies for sustainable urban transportation in europe," in *4th Asia Pacific Conference on Transportation and the Environment*, Xi'An, China, 2005.

[4] D. Schrank and T. Lomax, "The 2005 urban mobility report," Texas Transportations Institute, The Texas A&M University System, Tech. Rep., 2005.

[5] D. Schrank and T. Lomax, "2009 urban mobility report," Texas Transportation Institute, The Texas A&M University System, Tech. Rep., 2009.

[6] M. McIntyre, M. Rosenberg, and L. Hayes, "The global road safety crisis: We should do much more," UN the General Assembly meeting, Tech. Rep., 2004.

[7] Global Innovation Centre and Griffith University, "Background research on conduct of national/international market analysis - cybernetic transportation systems development & demonstration projects," Tech. Rep., 2007.

[8] M. Parent and G. Gallais, "Intelligent transportation in cities with cts," in *ITS World Congress*, Chicago, USA, 2002.

[9] D. Reichardt, "Kontinuierliche Verhaltenssteuerung eines autonomen Fahrzeugs in dynamischer Umgebung," Ph.D. dissertation, Universitaet Kaiserslautern, 1996.

[10] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, no. 9, 2008.

[11] I. Miller, M. Campbell, D. Huttenlocher, and F.-R. Kline, "Team Cornells Skynet: Robust Perception and Planning in an Urban Environment," *Journal of Field Robotics*, vol. 25, no. 8, 2008.

[12] Y.-L. Chen, V. Sundareswaran, C. Anderson, P. G. Alberto Broggi, P. P. Porta, P. Zani, and J. Beck, "TerraMax: Team Oshkosh Urban Robot," *Journal of Field Robotics*, vol. 25, no. 10, pp. 841–860, 2008.

[13] J. R. McBride, J. C. Ivan, D. S. Rhode, J. D. Rupp, M. Y. Rupp, J. D. Higgins, D. D. Turner, and R. M. Eustice, "A perspective on emerging automotive safety applications, derived from lessons learned through participation in the darpa grand challenges," *Journal of Field Robotics*, vol. 25, no. 10, pp. 808–840, 2008.

[14] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, and M. Walter, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[15] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield, "Little Ben: The Ben Franklin Racing Teams Entry in the 2007 DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, 2008.

[16] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, and D. Haehnel, "Junior: The Stanford Entry in the Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, 2008.

[17] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schroeder, M. Thuy, M. Goebl, F. v. Hundelshausen, O. Pink, C. Frese, and C. Stiller, "Team AnnieWAYs Autonomous System for the 2007 DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 616–639, 2008.

[18] B. J. Patz, Y. Papelis, R. Pillat, G. Stein, and D. Harper, "A Practical Approach to Robotic Design for the DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, 2008.

[19] A. Bacha, C. Bauman, R. Faruque, M. Fleming, and C. Terwelp, "Odin: Team VictorTangos Entry in the DARPA Urban Challenge," *Journal of Field Robotics*, vol. 25, no. 8, 2008.

[20] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. . Whittaker, Z. Wolkowicki, and J. Ziglar, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, 2008.

[21] P. Ridao, J. Battle, J. Amat, and G. Roberts, "Recent trends in control architectures for autonomous underwater vehicles," *International Journal of Systems Science*, vol. 30, no. 9, pp. 1033–1056, 1999.

[22] A. Broggi, M. Bertozzi, C. G. L. Bianco, A. Fascioli, and A. Piazzi, "The ARGO autonomous vehicle's vision and control systems," *International Journal of Intelligent Control and Systems*, vol. 3, no. 4, pp. 409–441, 1999.

[23] N. Vandapel, R. Donamukkala, and M. Hebert, "Unmanned ground vehicle navigation using aerial ladar data," *The International Journal of Robotics Research*, vol. 25, no. 1, pp. 31–51, 2006.

[24] P. Bellutta, R. Manduchi, L. Matthies, K. Owens, and A. Rankin, "Terrain perception for DEMO III," in *Intelligent Vehicles Conference*, 2000.

[25] A. Lacaze, K. Murphy, and M. DelGiorno, "Autonomous mobility for the DEMO III experimental unmanned vehicle," *Association for Unmanned Vehicle Systems– Unmanned Vehicle 2002*, 2002.

[26] J. S. Albus and A. J. Barbera, "RCS: A Cognitive Architecture for Intelligent Multi-Agent Systems," in *Proceedings of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles, IAV 2004*, 2004.

[27] DARPA, "Urban challenge rules," 2006. [Online]. Available: http://www.darpa.mil/grandchallenge/docs/Urban_Challenge_Rules_121106.pdf

[28] G. Conte, A. Broggi, M. Bertozzi, and A. Fascioli, "ARGO Home Page," 1998. [Online]. Available: http://www.argo.ce.unipr.it/ [Last accessed: 08.07.2009]

[29] R. Gregor, M. Luetzeler, M. Pellkofer, K. H. Siedersberger, and E. D. Dickmanns, "EMS-Vision: A Perceptual System for Autonomous Vehicles," *IEEE Transaction on Intelligent Transportation Systems*, vol. 3, no. 1, 2002.

[30] J. Kolodko and L. Vlacic, "Cooperative autonomous driving at the Intelligent Control Systems Laboratory," *IEEE Intelligent Systems*, vol. 18, no. 4, pp. 8–11, 2003.

[31] R. Kocik and Y. Sorel, "A methodology to design and prototype optimized embedded robotic systems," in *Computational Engineering in Systems Applications CESA'98.* Tunisia: In Proc. of the Computational Engineering in Systems Applications CESA'98, Tunisia, April 1998, 1998.

[32] T. Grandpierre, C. Lavarenne, and Y. Sorel, "Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors," in *CODES'99 International Workshop on Hardware/Software Co-Design.* Rome: CODES'99 7th International Workshop on Hardware/Software Co-Design, Rome, May 1999, 1999.

[33] M. Parent and G. Gallais, "Cybercars: Review of first projects," in *Ninth International Conference on Automated People Movers*, Singapore, 2003.

[34] M. Yang, C. Wang, R. Yang, and M. Parent, "CyberC3: Cybercars Automated Vehicles in China," in *Transportation Research Board Annual Meeting 2006*, Washington DC, US, 2006.

[35] IBEO Automobile Sensor GmbH, "Ladar digital a 3p operating manual," Tech. Rep., 2000.

[36] CyberC3, "CyberC3 official web site," 2007. [Online]. Available: http://cyberc3.sjtu.edu.cn/en/index.htm [Last accessed: 06.02.2007]

[37] C. Mentzer, G. McWilliams, and K. Kozak, "Dynamic autonomous ground vehicle re-routing in an urban environment using a priori map information and lidar fusion," in *2009 Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*, Troy, MI, USA, 2009.

[38] VisLab Press Release, "A journey from Rome to Shanghai while nobody is driving," 2009. [Online]. Available: http://vislab.it/press [Last accessed: 25.01.2011]

[39] VisLab Press Release, "VisLab's challenge successfully accomplished: the autonomous vehicles reached Shanghai Expo," 2010. [Online]. Available: http://vislab.it/press [Last accessed: 25.01.2011]

[40] J. M. Wille, F. Saust, and M. Maurer, "Stadtpilot: Driving Autonomously on Braunschweig's Inner Ring Road," in *2010 IEEE Intelligent Vehicles Symposium*, San Diego, CA, USA, 2010.

[41] S. Thrun, "What we're driving at," 2010. [Online]. Available: http://googleblog.blogspot.com/2010/10/what-were-driving-at.html [Last accessed: 25.01.2011]

[42] M. Buehler, K. Iagnemma, and S. Singh, "Editorial," *Journal of Field Robotics*, vol. 25, no. 10, pp. 725–726, 2008.

[43] DARPA, "Darpa announces third grand challenge - urban challenge moves to the city," 2006. [Online]. Available: http://www.darpa.mil/grandchallenge/docs/PR_UC_Announce_Update_12_06.pdf, [Last accessed: 19.07.2009]

[44] F. W. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher, and J.-M. Wille, "Caroline: An Autonomously Driving Vehicle for Urban Environments," *Journal of Field Robotics*, vol. 25, no. 9, 2008.

[45] M. Campbell, M. Egerstedt, J. P. How, and R. M. Murray, "Autonomous driving in urban environments: approaches, lessons and challenges," *Philosophical Transactions of the Royal Society*, vol. 368, no. 1928, pp. 4649–4672, 2010.

[46] W. Stadler, Ed., *Multicriteria optimization in engineering and in the sciences*. Plenum Press, New York, 1988.

[47] A. Furda and L. Vlacic, "Towards increased road safety: Real-time decision making for driverless city vehicles," in *2009 IEEE International Conference on Systems, Man, and Cybernetics*, San Antonio, TX, USA, 2009.

[48] A. Furda and L. Vlacic, "An object-oriented design of a world model for driverless city vehicles," in *2010 IEEE Intelligent Vehicles Symposium (IV 2010)*, San Diego, California, USA, 2010.

[49] U. Zimmer and E. v. Puttkamer, "Comparing world-modelling strategies for autonomous mobile robots," in *IWK '94*, Ilmenau, Germany, 1994.

[50] T. Hong, M. Abrams, T. Chang, and M. Shneier, "An intelligent world model for autonomous off-road driving," *Computer Vision and Image Understanding, 2000*, 2000.

[51] R. Benenson, S. Petti, T. Fraichard, and M. Parent, "Integrating perception and planning for autonomous navigation of urban vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, 2006.

[52] SAFESPOT, "Safespot integrated project," 2010. [Online]. Available: http://www.safespot-eu.org [Last accessed: 13.05.2010]

[53] DARPA, "Sample route network definition file (rndf)," 2007. [Online]. Available: http://www.darpa.mil/GRANDCHALLENGE/docs/Sample_RNDF(v1.5).txt [Last accessed: 29.07.2008]

[54] A. Furda, L. Bouraoui, M. Parent, and L. Vlacic, "Improving safety for driverless city vehicles: Real-time communication and decision making," in *3rd IEEE International Symposium on Wireless Vehicular Communications: IEEE WiVEC 2010*, Taipei, Taiwan, 2010.

[55] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. Elsevier, Inc., 2007.

[56] M. Fowler, *UML Distilled*, 3rd ed. Addison-Wesley, 2004.

[57] K. J. Astrom and T. Hagglund, *Advanced PID Control*. ISA - Instrumentation, Systems, and Automation Society, 2006.

[58] B. Evjen, K. Sharkey, T. Thangarathinam, M. Kay, A. Vernet, and S. Ferguson, *Professional XML*. Wiley Publishing, Inc., 2007.

[59] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*, 2nd ed. Springer, 2008.

[60] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1994.

[61] S. Boisse, R. Benenson, L. Bouraoui, M. Parent, and L. Vlacic, "Cybernetic transportation systems design and development: Simulation software," in *IEEE International Conference on Robotics and Automation - ICRA 2007. Roma, Italy.*, 2007.

[62] K. R. Apt and E.-R. Olderog, *Verification of sequential and concurrent programs.* Springer-Verlag, 1997.

[63] P. A. Laplante, *Real-time systems design and analysis.* IEEE Press, IEEE Computer Society Press, 1997.

[64] C. Heitmeyer and D. Mandrioli, Eds., *Formal methods for real-time computing.* John Wiley & Sons, 1996.

[65] J. L. Peterson, *Petri net theory and the modeling of systems.* Prentice-Hall, Inc., 1981.

[66] A. Furda and L. Vlacic, "Multiple criteria-based real-time decision making by autonomous city vehicles," in *7th IFAC Symposium on Intelligent Autonomous Vehicles (IAV 2010)*, Lecce, Italy, 2010.

[67] V. Chankong and Y. Y. Haimes, *Multiobjective Decision Making.* Elsevier Science Publishing Co., Inc., 1983.

[68] K. P. Yoon and C.-L. Hwang, *Multiple attribute decision making.* SAGE Publications Inc., 1995.

[69] S. Bennett, *Real-time computer control: an introduction.* Prentice Hall Europe, 1994.

[70] D. J. White, *Fundamentals of Decision Theory.* American Elsevier Publishing Co., Inc., 1976.

[71] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aaron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, and P. Stang, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[72] Team TerraMax, "Team TerraMax DARPA Grand Challenge 2005 - Technical Paper," 2005. [Online]. Available: http://www.darpa.mil/grandchallenge05/TechPapers/TeamTerraMax.pdf, [Last accessed: 18.07.2009]

[73] W. S. Levine, Ed., *The control handbook.* CRC Press, Inc., 1996.

[74] B. Molinete, L. Bouraui, E. Naranjo, H. Kostense, F. Hendriks, J. Alonso, R. Lobrino, and L. Isasi, *CyberCars-2: Close Communications for Cooperation between CyberCars.* Technical Report Project No IST-2004-0228062, 2009.

[75] G. Baille, P. Garnier, H. Mathieu, and P.-G. R., "Le Cycab de l'INRIA Rhone-Alpes," Intitut National de Recherce en Informatique et en Automatique (INRIA), France, Tech. Rep. 0229, 1999.

[76] C. Pradalier, J. Hermosillo, C. Koike, C. Braillon, P. Bessire, and C. Laugier, "The cycab: a car-like robot navigating autonomously and safely among pedestrians," *Robotics and Autonomous Systems*, vol. 50, no. 1, pp. 51–68, 2005.

[77] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum, and L. Viennot, "Optimized link state routing protocol," in *IEEE INMIC*, 2001.

[78] A. Furda, L. Bouraoui, M. Parent, and L. Vlacic, "The role and future challenges of wireless communication networks for cooperative autonomous city vehicles," in *IFIP International Conference Network of the Future*, Brisbane, Australia, 2010.

[79] A. Furda and L. Vlacic, "Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making," *IEEE Intelligent Transportation Systems Magazine*, 2010, accepted for publication.

[80] A. Furda and L. Vlacic, "Real-time decision making for autonomous city vehicles," *Journal of Robotics and Mechatronics*, vol. 22, no. 6, 2010.

[81] A. Stentz, A. Kelly, P. Rander, H. Herman, O. Amidi, R. Mandelbaum, G. Salgian, and J. Pedersen, "Real-time, multi-perspective perception for unmanned ground vehicles," *AUVSI*, 2003.

[82] A. Kelly, A. Stentz, O. Amidi, M. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, and H. Herman, "Toward reliable off road autonomous vehicles operating in callenging environments," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 449–483, 2006.

[83] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *IEEE International Conference on Robotics and Automation*, 1994.

[84] J. Albus, H.-M. Huang, E. Messina, K. Murphy, M. Juberts, A. Lacaze, S. Balakirsky, M. Schneier, T. Hong, H. Scott, F. Proctor, W. Shackleford, J. Michaloski, A. Wavering, T. Kramer, N. Dagalakis, W. Rippey, K. Stouffer, S. Legowik, J. Evans, R. Bostelman, R. Norcross, A. Jacoff, S. Szabo, J. Falco, R. Bunch, J. Gilsinn, T. Chang, T.-M. Tsai,

A. Meystel, A. Barbera, M. L. Fitzgerald, M. d. Giorno, and R. Finkelstein, "4D/RCS: A Reference Model Architecture For Unmanned Vehicle Systems Version 2.0," National Institute of Standards and Technology, Tech. Rep., 2002.

[85] A. Bensrhair, M. Bertozzi, A. Broggi, P. Miche, S. Mousset, and G. Toulminet, "A cooperative approach to vision-based vehicle detection," *In Procs. IEEE Intl. Conf. on Intelligent Transportation Systems 2001*, pp. 209–214, 2001.

[86] A. Piazzi, C. G. L. Bianco, M. Bertozzi, A. Fascioli, and A. Broggi, "Quintic G2-splines for the Iterative Steering of Vision-based Autonomous Vehicles," *IEEE Trans. on Intelligent Transportation Systems*, vol. 3, no. 1, pp. 27–36, 2002.

[87] M. Bertozzi, A. Broggi, A. Fascioli, and A. Tibaldi, "An evolutionary approach to lane markings detection in road environments," *In Atti del 6 Convegno dell'Associazione Italiana per l'Intelligenza Artificiale*, pp. 627–636, 2002.

[88] L. Vlacic, M. Parent, and F. Harashima, Eds., *Intelligent Vehicle Technologies*. Butterworth-Heinemann, 2001.

[89] Y. Koren, J. Borenstein, and A. Arbor, "Potential field methods and their inherent limitations for mobile robot navigation," in *IEEE Conference on Robotics and Automation*, Sacramento, California, 1991, pp. 1398–1404.

[90] A. D. Lattner, I. J. Timm, M. Lorenz, and O. Herzog, "Knowledge-based risk assessment for intelligent vehicles," in *International Conference on Integration of Knowledge Intensive Multi-Agent Systems KIMAS*, Massachusetts, USA, 2005.

[91] A. D. Lattner, J. D. Gehrke, I. J. Timm, and O. Herzog, "A knowledge-based approach to behavior decision in intelligent vehicles," Center of Computing Technologies, Universitaet Bremen, Germany, 2006.

[92] M. Pellkofer and E. D. Dickmanns, "Behavior decision in autonomous vehicles," in *IEEE Intelligent Vehicle Symposium*, vol. 2, Versailles, France, 2002, pp. 495–500.

[93] E. D. Dickmanns, "An advanced vision system for ground vehicles," in *1st International Workshop on In-Vehicle Cognitive Computer Vision Systems*, Graz, Austria, 2003.

[94] A. Kelly and A. Stentz, "An approach to rough terrain autonomous mobility," in *International Conference on Mobile Planetary Robots*, Santa Monica, California, 1997.

[95] J. Fulton and J. Pransky, "DARPA Grand Challenge - a pioneering event for autonomous robotic ground vehicles," *Industrial Robot: An International Journal*, vol. 31, no. 5, pp. 414–422, 2004.

[96] W. Whittaker and DARPA, "Red Team DARPA Grand Challenge 2005 Technical Paper," no. 01.02.2007, 24.08.2005 2005.

[97] K. Peterson and DARPA, "Red Team Too DARPA Grand Challenge 2005 Technical Paper," no. 01.02.07, 2005.

[98] P. G. Trepagnier, J. E. Nagel, P. M. Kinney, C. Koutsougeras, and M. T. Dooner, "Kat-5: Robust Systems for Autonomous Vehicle Navigation in Challenging and Unknown Terrain," *Journal of Field Robotics*, vol. 23, no. 8, pp. 509–526, 2006.

[99] D. Braid, A. Broggi, and G. Schmiedel, "The TerraMax autonomous vehicle concludes the 2005 DARPA Grand Challenge," *In Procs. IEEE Intelligent Vehicles Symposium 2006*, pp. 534–539, 2006.

[100] DARPA, "News release - a huge leap forward for robotics r&d," 2005. [Online]. Available: http://www.darpa.mil/grandchallenge05/ GC05winnerFINALwTerraM.pdf [Last accessed: 17.07.2009]

[101] S. R. Team, "Stanford Racing Team's Entry In The 2005 DARPA Grand Challenge," DARPA, Tech. Rep., 2005. [Online]. Available: http://ai.stanford.edu/ dstav [Last accessed: 17.07.2009]

[102] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME-Journal of Basic Engineering*, vol. 82 (Series D), pp. 35–45, 1960.

[103] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," Department of Computer Science, University of North Carolina at Chapel Hill, 2006.

[104] B. Hein, *Automatische offline Programmierung von Industrierobotern in der virtuellen Welt, Dissertation Universitaet Karlsruhe.* GCA-Verlag, 2003.

[105] A. Furda, "Implementierung und vergleich von bahnplanungsverfahren fuer industrieroboter: das BB-Verfahren und hierarchischer A*," Thesis (Diplomarbeit), University of Karlsruhe, Germany, Institute for Process Control and Robotics (IPR), 2004.

[106] P. G. Trepagnier, P. M. Kinney, J. E. Nagel, M. T. Dooner, and J. S. Pearce, "Team Gray Technical Paper DARPA Grand Challenge 2005," 2005. [Online]. Available: http://www.darpa.mil/grandchallenge05/TechPapers/GreyTeam.pdf, [Last accessed: 18.07.2009]

[107] A. Broggi, C. Caraffi, R. I. Fedriga, and P. Grisleri, "Obstacle detection with stereo vision for off-road vehicle navigation," *In IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Intl. IEEE Wks. on Machine Vision for Intelligent Vehicles*, vol. 3, pp. 65–72, 2005.

[108] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. The MIT Press, 1998.

[109] W. Reisig, *Petri nets*. Springer Verlag, 1985.

Index