Teaching a Robot to Drive

a Skill-Learning Inspired Approach

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doktor rerum naturalium" der Georg-August-Universität Göttingen

vorgelegt von
Irene Markelić aus Datteln

Göttingen, 2010

Referentin/Referent: Prof. Florentin Wörgötter
Koreferentin/Koreferent: Prof. Ulrich Parlitz
Tag der mündlichen Prüfung: 6. August 2010

Abstract

Robots can alleviate our lives by accomplishing tasks that are unpleasant or even dangerous for us. To employ them efficiently they should be autonomous, adaptive, and easy to instruct and use. Traditional white-box approaches to robot control are based on an engineer's understanding of the physical structure of a given problem from which he or she derives a possible solution which is implemented into the system. This is a powerful approach, yet also limited in several ways. The primary drawback is that such systems depend on predefined knowledge and therefore each additional behavior again requires the same expensive implementation cycle.

By contrast, humans and many other animals are not limited to innate behaviors but can acquire various skills during lifetime. In addition, for them detailed knowledge of the underlying (physical) process of a given task does not seem to be required. These features are attractive for artificial systems too.

In this thesis we therefore investigate the hypothesis that principles of human skill acquisition may lead to alternative methods for adaptive system control. In addition we postulate that an economical factor should be considered, i.e. tasks should be solved with little a priori knowledge and little processing, to arrive at efficient processing mechanisms. We investigate this for the task of autonomous driving which is a classical problem in system control with a wide range of applications. The precise task is to learn basic, anticipatory driving from a human supervisor.

After reviewing relevant aspects concerning human skill acquisition and introducing the notion of *internal models* and *chunking* we propose the application of the latter to the given task. We realize chunking with a database where human driving examples are attached to extracted descriptions of the visually perceived street trajectory. This is first realized in an indoor scenario using a robot, and later, in the course of the European DRIVSCO project, transferred to a real car. Furthermore, we investigate the learning of visual forward models which are part of internal models and their effect on control performance in the laboratory setup.

The main result of this interdisciplinary and application oriented work is a system which is able to produce action plans in response to visually perceived street trajectories, without requiring metric information for street following. The predicted actions concerning the robot setup were steer and speed, and for the real car steer and acceleration where the predictive capactiy of the system for the latter was limited. Hence, the robot learned to drive autonomously from a human teacher and the car learned to predict the human's actions. The latter was successfully demonstrated during the international review meeting of the project. The outcome of this work is interesting for applications in robot control and especially intelligent driver assistance systems.

Acknowledgements

This work originated in a diverse and international working atmosphere created by many curious and bright characters. One of them is my seemingly tireless supervisor Florentin Wörgötter who is always on the quest for his next dopamine dose. His constant support helped me to overcome the inevitably occurring critical phases during such a work and I came to very much appreciate his light-hearted, warm and encouraging guidance. With his political incorrectness, self-irony, and lack of hierarchical thinking he is not only an inspiring supervisor, but also a great company. I also thank his wife and secretary Ursula for always being very helpful and friendly.

Most practical hands-on help, especially in the first year, but also beyond that, I received from my colleague Tomas Kulvičius who I thank very much for this.

I also thank the other members of our working group, Jan-Matthias Braun, Sinan Kalkan, Babette Dellen, Poramate Manoonpong, Minija Tamosiunaite, Kejun Ning, Eren Aksoy, Alexey Abramov, and Guoliang Liu.

Furthermore, I thank Christoph Kolodziejski. Only few colleagues are thus open-minded, devoted and helpful as he is and I was allowed to learn that this is not only his attitude in his professional life.

I further want to express my appreciation for all members of the DRIVSCO project, among others Norbert Krüger and Marc van Hulle who welcomed me in their labs. Especially, I want to mention the A-Team(!), my colleagues with whom I worked in Lippstadt to finish the preparations for the final project demonstration: Karl Pauwels, Anders Kjær-Nielsen, and Lars Baunegaard With Jensen.

During the doctorate work is a very dominant part of one's life, which may be tiring and I'd like to thank my close friends Adina Grafe, Grazia Crapanzano and Tanja Brigadski, who helped me to fill the scientific breaks with joy and distraction.

Am meisten möchte ich meiner Familie danken, vor allem meinen Eltern. Ich kann nicht in Worte fassen wieviel eure Unterstützung für mich bedeutet. Ihr habt mir geholfen, wenn ich Hilfe brauchte, ihr habt mir zugehört, wenn ich jemanden zum Reden brauchte und ihr habt mich getreten, wenn ich getreten werden musste. Dafür und für alles was ihr seid danke ich euch so sehr.

Für meine Mutter und meinen Vater,
Annemarie und Momir Markelić.

Contents

Chapter 1

Introduction

”The self thus becomes aware of itself, at least in its practical action, and discovers itself as a cause among other causes and as an object subject to the same laws as other objects.“
Jean Piaget

We share our world with artificial technical devices which we build for accomplishing tasks unpleasant or even dangerous for us. Some general requirements to make efficient use of such systems are that they a) are easy and intuitive to design, build and program (instruct), b) function autonomously, i.e. without supervision, c) that they are adaptive, meaning to be able to cope with the challenges of a changing environment, and furthermore d) that they are available at affordable prices. Traditional approaches to system control, although very powerful and established, also have certain shortcomings concerning these requirements. On the other hand skilled animal behavior comprises features which are very attractive for artificial systems, too. Both are compared in the following.

1.1 The Problem of Control

The advanced kind of artificial systems which we will refer to as *robots*, is not only equipped with actuators to carry out actions, but also with sensors which can provide information about the system itself and its environment. This perceived information is used to describe the *state* of the system, which informally describes its situation (a formal definition will be given in section 3.2.1). To achieve that a robot accomplishes a given task, it must be equipped with the knowledge how to act appropriately in a given state. In control engineering this is referred to as *control* or *controller*, and in the domain of machine learning as *policy*. The rules making up the controller are usually called *control laws*. A controller thus realizes a desired *behavior* of a robot. For example, if the task is to drive a car, the sensed states might refer to the car's position and orientation within the street and a controller might specify which steering or speed commands should be applied to the car in response. The result is an observable *behavior*, in this case driving.

Hence, a controller is a function which takes a sensed state as input and maps it to a desired control output. This is only possible if the input contains enough information to unambiguously perform this mapping. In other words, the necessary information for solving the task must be contained in the state description, in which case it is said to be complete. At the same time no superfluous information should be contained to avoid unnecessary computations. For the driving example this means that it may make sense to include the current speed of the robot-car in the state description but not the color of a detected car in front.

Furthermore, since the action of the system is derived from the states, it is important that these correctly reflect reality. A robot that is instructed to move through open doors will perform poorly if sensing a closed door erroneously as open.

These problems (control, state definition, and sensing) must be tackled to achieve that a system solves a given task, i.e. to achieve system control. They are far from trivial and subject of current research.

1.2 White-Box System Control

Traditionally, system control is treated in (control) engineering and its subfield control theory which usually employ so-called white-box approaches, meaning that control is derived from the known underlying structure or process of a given problem. For example, if the task is to move an object from position A to position B, one could use knowledge about the well-studied field of mechanics and apply Newton's laws to determine how much force should be applied to the object to achieve the desired translation. This requires, in addition to knowing about classical mechanics, the knowledge of other parameters, e.g. the weight of the object, and the friction of the floor. This simple example already illustrates important features of such approaches: Once, all the necessary information is given, they may work very reliably, making use of a powerful and well-established physical and mathematical framework. However, all knowledge is put into the robot by the system designer. The system becomes more "intelligent" the more knowledge it is equipped with. Anything not considered a priori cannot be handled by such a robot. Consequently, for controllable environments where it is possible to consider all contingencies, such traditional white-box approaches lead to highly reliable and autonomous systems, as is the case with industrial robots. However, "real-world" environments can be unpredictable and trying to provide solutions for all eventualities can be a very challenging if not impossible task. Therefore, it can be concluded that the higher the dependence of a control strategy on a priori knowledge, the more demanding is the design of according controllers, especially for unpredictable or complex environments. Another limitation of this approach is that the process of designing and programming must be repeated for every additional behavior that a robot is supposed to be equipped with. This is acceptable for special-purpose machines but undesired for more general future systems.

Hence, alternative approaches to control not suffering these shortcomings are desired. This may be found in biology, which is the topic of the next section.

1.3 White-Box Control vs. Skill Acquisition

The artificial abilities of robots to perceive the world and to act in it are natural to living beings. Animals have little problems to perceive the world and to behave appropriately in a changing environment, thus they can be seen as systems for which the control problem has been solved, and as such may serve as role model for improving system control.

Although there are innate "programs" (instincts), most animals can acquire new skills during their lifetime. This great flexibility allows them to adapt to their particular environment and thus improves their chances of survival. Especially impressive is the capacity of us, humans, to acquire new skills: farming, learning new languages, domesticating other animals for our benefit, and programming computers are just a few examples showing in which diverse ways we master our environment. Thus, the ability of skill acquisition seems to be a key ingredient for our successful performance in the world.

In the following we will contrast skilled animal behavior with white-box control and see that the former comprises features which are desirable for artificial system control, too. We argue, that these features may result from mechanisms underlying skill learning as well as economic pressure.

Since skills are acquired during lifetime they must be achieved without provided specific a priori, (i.e. innate) knowledge about how to solve a particular task. This difference to traditional system control makes the animal flexible and adaptive.

An additional difference between skilled behavior and white-box approaches is that new skills can be acquired without explicit instruction but for example by demonstration. One type of human and animal learning that can lead to the acquisition of new skills is *imitation learning* in which the learner acquires a new skill simply by copying the behavior of a teacher [Byrne and Russon, 1998; Demiris and Meltzoff, 2008; Meltzoff, 1988; Dautenhahn and Nehaniv, 2002; Ferrari et al., 2009]. This apparent ease of acquiring a new skill contrasts strongly with the effort of programming complete processes and strategies into a robot as necessary in the control-theoretic white-box approach, and is therefore a desirable feature for artificial systems, too.

Furthermore, it is remarkable that neither for learning nor for performing a new skill we need to know the underlying structure or processes of the task. For example, we can operate a vehicle without knowing anything about its internal workings or mechanics! This "black-box" approach to control, together with economic pressure, may lead to more efficient, perception-action couplings than in traditional control. This claim is justified as follows: When control is derived from the understanding of a physical process, typical parameters that must be provided are for example absolute distances, weights, forces, friction, positions in space etc. In this case, effort must be spent for determining such values from the sensory input, an often costly process involving a lot of knowledge and computations. Knowledge must either be provided, which is expensive because it requires effort from the designer, or acquired by the system itself, which is expensive because it requires time. Also computations require time, computational power, and energy. Therefore, another control strategy solving the same task but requiring less knowledge and less computations can be seen as more economical. (Note, that our postulate here differs from the idea of an *ecological* approach which usually refers to understanding entities by considering them as being part of greater interacting systems.) For living beings the consumption of time and energy can be crucial for survival, e.g. in situations that require a fast behavior like fleeing or hunting. Therefore we believe that it is reasonable to assume that evolution favors economical processes. An example may be navigation in space:

A common approach in traditional robotics for navigation in space is to let the robot compute a 2D or 3D map of the environment from its sensory input and to let it locate obstacles and itself in this map. To move and avoid obstacles it is equipped with routines to plan paths (usually graph search methods) which are translated into the robot's action space using a dynamical/kinematic model of the robot itself[1] [Latombe, 1990; LaValle, 2006]. Thus, knowledge is required in form of given models and the information how to compute a metric map from sensory input, and furthermore computations are required, namely those to compute the map.

By contrast, animals may rely on more direct perception-action couplings, solving the same problem. For example, to avoid obstacles, it was suggested that humans while driving do not rely on absolute distances when braking but instead use the expansion rate of an obstacle mapped on the perceiver's retina, a strategy indicated to be also used by some other animals [Lee, 1976]. Another example are insects which navigate in space relying on *reactive* mechanisms that require almost no processing of sensory information which is very economical [Cartwright and Collett, 1983; Mallot and Basten, 2009; Wehner et al., 1996; Collett, 1996; Gerstmayr et al., 2008].

This example shows that complex behaviors can be achieved by simpler strategies, which may be due to economic pressure. They require less knowledge and processing, and are thus more economical. For artificial system control this means that the effort in programming and the demand of hardware (see also section 2.3) can be reduced, which in the end will lead to more affordable products.

Since simple reactive behaviors also have certain limitations the question arises if and how these can be improved, while retaining their efficiency. This question is tackled in the next section.

[1]A dynamical model captures dynamical properties of the object. E.g. how it behaves if a certain force is applied. It can also include restrictions of the object, like maximal accelerations. A kinematic model describes the motion behavior of the object not considering the force which caused the motion.

1.4 Reactive vs Deliberative Control and Internal Representations

In the previous section reactive behavior was given as example for an efficient control strategy. However, despite this capacity it has limitations. For example it requires continuous sensory input, it is slow in the sense that a re-action can only occur after(!) the perception of a stimulus, and it does not allow deliberative, i.e. planned behavior, which can be a sequence of actions to be taken in the future, i.e. an action plan.

It was suggested that deliberative behavior in animals is based on *internal representations* [Clark and Grush, 1999; Grush, 2004; Hesslow, 2002, 1994] which are considered to be mappings of experienced entities or processes into so-called *forward models*, see section 3.2.1. (The acquisition of these models is considered to be one important factor in human skill acquisition. Another is the ability to compile elementary (possibly action) entities into larger ones, called *chunking*, see section 3.2.1 and 3.2.2).

The metric environmental map together with kinematic/dynamic models from the white-box example in the previous section can be seen as an example of such a representation. In fact, animals with binocular vision are in principle able to create 3D allocentric maps (centered on aspects of the external environment [Klatzky, 1998]) similar to the metric map from the control-theoretic example. However, if such maps are indeed created is not known, and if so it is questionable whether they are used for the task of motion control [Wang and Spelke, 2002; McNamara et al., 2008; Gibson, 1979; Churchland et al., 1994; Berthenthal, 1996].

In summary, this leads to the question whether it is possible to achieve action plans without (metric) internal representations, i.e. extend the simple and efficient reactive behavior?

1.5 Conclusion and Hypothesis

The reviewed information suggests that although traditional white-box approaches to system control are powerful they are constrained while skilled animal behavior exhibits features desirable for artificial system control, too. It consequently appears promising to study in how far mechanisms underlying skill acquisition can be used for developing alternative approaches to system control. Furthermore, this should be done while also considering economical factors, since these may influence skilled behavior. This is the topic that this work is devoted to.

The hypothesis of this thesis is that applying mechanisms underlying human skill learning to a real-world application while taking economical factors into account is beneficial in three distinct ways:

I) By using human skill learning as paragon while taking an economical stance alternative methods for system control can be developed. Not necessarily with the demand to replace tried and tested approaches but with the aim to improve and supplement available methods with respect to the stated requirements for system control (autonomous, adaptive, easy to instruct, and affordable).

II) By using a real-world application instead of a toy environment (simulation or laboratory only) the idea is borrowed that "the world is its own best model" [Brooks, 1990, p. 5]. This way it is ensured that hard constraints are kept and the end product is applicable under realistic conditions.

III) Since the "mechanisms of human skill learning" are a subject of research themselves it is not possible to resort to completely worked-out and well-defined function descriptions. Instead, often various more or less supported hypotheses, sometimes even contradictory, are encountered, appearing all reasonable and sound. According to the idea of "understanding by synthesis", i.e. by synthesizing control, based on such proposed ideas[2] regarding skill learning, possible flaws and lacks of elaborateness can be revealed. Thus, the approach taken in this thesis may contribute by helping to better evaluate theories concerning (human) skill learning.

The stated hypothesis has two consequences:

a) Due to the focus on a real-world application this work is strongly application-oriented.

b) Since methods from different research areas are combined this work is very interdisciplinary, merging ideas from classic control theory, psychology and neurobiology, and finally machine learning, computer vision, and computer science.

1.6 Task Specification

After having defined the goals of this thesis now the real-world application that is to be realized is described in detail. The task tackled is that of *autonomous driving* which is a classical research field in autonomous system control. It is of high scientific interest, desired for military applications, rescue systems, and of great economic importance since it has a direct use for advanced driver assistant systems (ADAS), where it can help to increase safety by reducing the risk of accidents.

In the course of the EU funded DRIVSCO project during which this thesis was prepared, a working driver assistant system was to be realized and demonstrated in a *real* car. One of its features, which was contributed by this thesis, was to detect and warn the driver when he or she deviated from his or her normal driving behavior. This was achieved based on solving the following task:

Create a system that learns basic, vision-based, proactive driving from a human teacher

[2]Note, that the aim is not to rebuild a biological processes, but instead to realize the effect of a suggested process appropriate for a computer application.

while minimizing the amount of required a priori knowledge.
For economic and safety reasons methods for solving this task were first tested in an indoor scenario using a robot and then ported and adapted to the needs of the real-car setup. The individual parts of the task specification are motivated and explained in the following:

Basic driving refers to the generation of lateral and longitudinal, i.e. steer and speed, control for street- or lane-following, i.e. not considering intersections, lane changes, traffic handling etc. This is the most basic task which must be solved by all approaches to driving. More advanced behaviors may be placed on top of it.

Vision is the primary source of information to humans during driving [Sivak, 1996]. Similar to the human eye cameras are passive, distal sensors which map 3D scenes onto 2D, and thus a camera was chosen to provide the sensory input for this task.

Proactive driving is used in diffuse ways in the literature, from meaning to anticipate the behavior of other drivers up to car-maintenance. The term proactive is defined in Merriam-Webster's dictionary as "acting in anticipation of future problems, needs, or changes". Here, proactive driving is used to refer to the generation of *action plans*. Note, that this implies less focus on anticipation but rather the ability to *plan ahead*. The ability to generate sequences of actions plays an important role in driving, for example it can reduce reaction time and help to cope with short lacks of sensory input. For a thorough list of the advantages of action sequence generation in general, see [Sun and Sessions, 2000].

Learning the behavior **from a teacher** is motivated by the previously stated requirement for autonomous systems to be easy to instruct. Imitation learning is well suited for this task requiring the human only to demonstrate the desired behavior, resulting in a very simple human computer interface (HCI).

Using as little a priori knowledge as possible introduces an economic factor which is supposed to enforce the developed control to be efficient without requiring much pre-coded information. This will reduce the effort of behavior design and system instruction on one hand and on the other reduce the demands on necessary hardware. Both factors lead to a decrease of the costs of the end product.

Finally, based on the specified task the following experimental setup is assumed which is also illustrated in Fig. 1.1: A human is operating a vehicle and the system has access to the vehicle's input, i.e. the control actions conducted by the driver. At the same time a camera capturing the scene ahead is providing visual sensory input. It is assumed that the human who is demonstrating the task is not a novice, but an experienced driver.

1.7 Overview and Structure

According to the hypothesis, one aim is to use techniques of human skill learning to solve the

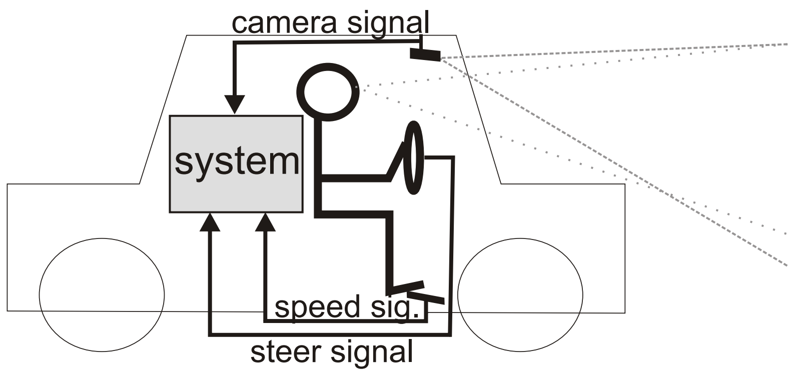


Figure 1.1: Sketch of the task setup: The system has access to the visual sensor input and the actions conducted by an experienced driver.

given task of learning driving from a human teacher and to test if existing approaches can be improved. Therefore, in chapter 2 the topic of autonomous driving is introduced consisting of a short history of milestones, followed by a listing of prominent approaches with an identification of their advantages and disadvantages. Then a short review of major tenets in human skill acquisition as far as concerning this work is given in chapter 3, highlighting crucial aspects, like open questions, and which mechanisms appear suited for the task. The chapter is concluded with stating which of the presented information can be applied to the given problem. This suggested framework is then formalized and realized in chapter 4, which is subdivided into two major sections, one devoted to perception 4.2, and one to action 4.3. The gained experience from this work will be that the proposed methods work well, however, also do contain certain shortcomings, mainly due to the explicit split between perception and action. The investigation of these problems under explicit consideration of the interdependencies between the two processes is the topic of chapter 5. This part is conducted in the laboratory environment using a robot instead of the real car. Finally, chapter 6 is devoted to the porting of the developed methods to the real-car system. Necessarily, this is analogous in many aspects to chapter 4. The work is concluded with a discussion of the experiences with the proposed methods with respect to the stated hypothesis in chapter 7.

1.8 Original Contributions

The main contributions of this work are:

- A system was developed able to generate action plans *without* requiring a metric world model by means of chunking and imitation learning for learning basic driving from a teacher, described in chapter 4 and published in [Markelic et al., 2008].
- The yet unpublished work of chapter 5 describes the *learning* of an internal representation by the system from experience, i.e. by explicitly considering the interdependence of action and

perception, and shows how it considerably improves system performance. This functionality is directly adapted from suggested skill learning mechanisms and has the potential to contribute to very adaptive and affordable driver assistance systems. To the knowledge of the author this is the first time that this has been applied to the task of driving.

- The developed methods for learning driving were adapted and ported to a real car and trained with real-driving data. The resulting behavior for driver assistance was successfully demonstrated during the final review meeting of the DRIVSCO project, observed by three independent international reviewers, see [EU-Newsletter, 2010]. This work is described in chapter 6 and submitted for publication [Markelić et al., 2010]. This shows that the developed methods work under realistic conditions and are usable for a real-world application.

Chapter 2

Autonomous Driving

"Any sufficiently advanced technology is indistinguishable from magic.“
Arthur C. Clarke

This chapter gives an overview on the field of autonmous driving and sets this thesis apart from other reported approaches. For this we first provide a list of milestones in autonomous driving and then identify typical approaches by means of prominent and representative examples. This is followed by a discussion of their advantages and disadvantages by means of which we place this thesis into the provided context.

2.1 Milestones in Autonomous Driving

The history of autonomous driving comprises a number of outstanding accomplishments which are described in this section and summarized in table 2.1.

Related to the field of autonomous driving is earlier work in vision based navigation, e.g. the robot Shakey [Nilsson, 1969], or the Stanford Cart [Moravec, 1982, 1990]. But autonomous driving, which has been tackled largely independently in Japan, Europe, and the USA, can be considered to have started at the end of the 1970s with the Tsukuba Intelligent Vehicle, developed in Japan [Tsugawa, 1993]. Since then, heavily funded long term programs and initiatives, e.g. the European project Prometheus (PROgramm for a European Traffic with Highest Efficiency und Unprecedented Safety) [Braess and Reichart, 1995], the NavLab project from Carnegie Mellon University, CMU [Thorpe et al., 1991] the ARGO project [Broggi et al., 1999] from Universita di Parma in Italy, or the DARPA challenges [DARPA, 2010] funded by the US government propelled advances in the field. During the Prometheus project (1987-1994), which involved 19 countries and European vehicle manufacturers, several highlights were achieved. For example, in 1987 the VaMoRs test vehicle (Versuchsfahrzeug für autonome Mobilität und Rechnersehen) developed at Universität der Bundeswehr München, UBM, under E. D. Dickmanns, was demonstrated of being able to drive autonomously for 20 km at speeds of up to 96 km/h. In 1994, the successor vehicles VITA-2 and VaMP (VaMoRs Passenger car) drove autonomously in normally dense three-lane traffic on a highway close to Charles de Gaulle Airport in Paris at speeds of 130 km/h. In 1995 followed a 1600 km trip with VaMP from Munich in Germany to Odense in Denmark and back. Speeds up to 180 km/h with only 5% human intervention were reached. Lateral (steer) and longitudinal (velocity) control were achieved based on black and white vision sensors [Dickmanns, 2002].

The NavLab project, lead by Chuck Thorpe at CMU, is still active. Since 1984 it has been dedicated to research in autonomous driving, using various test cars, called NavLab vehicles. The 11th and current vehicle is the NavLab 11, a robot Jeep Wrangler. In the same year in which VaMP drove from Munich to Odense, the vehicle NavLab 5 from CMU achieved 98.2% autonomous driving on a 5000 km trip, called "No Hands Across America" [Pomerleau, 1995]. Longitudinal control was not included, but handled by a human operator.

The ARGO project, an Italian initiative successive to Prometheus, proved its potential in the "Mille-Miglia in Automatico" in 1998, a journey of 2000 km over six days on the motorways of northern Italy, during which an average speed of 90 km/h was achieved and the car was in automatic mode 94% of the time.

The US-American Defense Advanced Research Projects Agency (DARPA) launched a competition to push forward the development in unmanned land vehicles. In 2004 and 2005 the competition was a 142 mile course on unstructured rough terrain, termed the Grand Challenge. During the first competition no team succeeded in finishing. In 2005 a robot car, Stanley, developed by a group from Stanford University, won the challenge. In 2007 the focus was shifted from driving on unstructured terrain to inner city driving and referred to as Urban Challenge. It was won by the robot car Boss from CMU.

Thus, autonomous driving has been pursued now for approximately 3 decades following different paradigms which are reviewed in the following section.

2.2 Approaches to Autonomous Driving

When scanning the literature on autonomous driving it becomes clear that two major tenets dominate the field. One is control engineering, i.e. applying methods from control theory (e.g. [Dickmanns and Graefe, 1988; Turk et al., 1988; Thrun et al., 2006; Urmson et al., 2007]) and the other machine learning (e.g. [Pomerleau, 1989; Pasquier and Oentaryo, 2007; Partouche et al., 2007; Kwasnicka and Dudala, 2002; Riedmiller et al., 2007; Togelius and Lucas, 2006]). Of course, much work is not exclusively employing one or the other strategy but using both to different extents. We will illustrate the characteristics of both by means of two especially clear and prominent examples from

Table 2.1: Milestones in autonomous driving

When	What	Where / Who	Description	Source
Early Work in Vision Based Navigation				
1966	Shakey	Artificial Intelligence Laboratory, SRI (Stanford Research Institute), group led by Charles Rosen	wheeled platform with planning and automated navigation capabilities using a camera, ultrasonic-, and touch sensors.	[Nilsson, 1969]
1970	Stanford Cart	Stanford University, Hans Moravec	wheeled platform with a TV-camera, could plan a path through cluttered environment at a speed of 1m/15 min. Lead to development of the CMU Rover in 1981	[Moravec, 1982, 1990]
Autonomously Driving Vehicles				
1977	The Intelligent Vehicle	Tsukuba Mechanical Engineering Lab, Japan. Developers were Tsugawa, Hirose and Yatabe	tracked lane markers using stereo vision, speeds of 30 km/h	[Tsugawa, 1993]
1987	VaMoRs vehicle	UBM, group led by E. D. Dickmanns	20 km at speeds up to 96 km/h, highway entring maneuvers from acceleration strip	Dickmanns [1998]
1995	VITA II vehicle and twin VaMP	UBM, group led by E. D. Dickmanns	driving on normal three-lane highway traffic, 130 km/h, lane changing, convoy driving, overtaking.	[Dickmanns, 1998]
1995	VaMP vehicle	UBM, group led by E. D. Dickmanns	1600 km at speeds up to 180 km/h, only 5% human intervention, using black and white cameras only	[Dickmanns, 2002]
1995	NavLab 5 vehicle, ”No Hands Across America”	CMU, Dean Pomerleau and Todd Jochem	5000 km, 1.8% human intervention, lateral control only	[Pomerleau, 1995]
1998	ARGO vehicle, ”Mille Miglia in Automatico”	University di Parma, Bertozzi, Broggi, Fascioli	2000 km at speeds up to 90km/h, 6% human intervention, using stereovision	[Broggi et al., 1999]
2004	DARPA Grand Challenge	no winner		
2005	Stanley robot wins DARPA Grand Challenge	Stanford, group led by Sebastian Thrun	140 miles rough terrain, featuring various sensors including GPS and laser range finders	[Thrun et al., 2006]
2007	Boss robot wins DARPA Urban Challenge	CMU and General Motors, group led by William ”Red” Whittaker	road driving, merging into moving traffic, intersections, parking	[Urmson et al., 2007]

either field, and one using methods from both. The first example is “The 4D Approach to Vision” which is based on control engineering, the second is “Stanley” the winner of the Great Challenge which employs methods from both areas, and finally the third example is “ALVINN”, representing the machine learning approach. The sections describing these examples first give a short general overview of the particular example, then describe crucial mechanisms, and finally conclude with an enumeration of most distinct features.

2.2.1 The 4D Approach to Vision

In the course of the heavily funded European program Prometheus (funded with one billion dollars) from 1987-1994 [Braess and Reichart, 1995; Dickmanns, 2007] major advances in the field of autonomous driving were accomplished and demonstrated by the VaMP vehicle. The system could sense road curvature, lane width, number of lanes, and lane markings, using two cameras, one in the front and one in the rear with different focal lengths at a resolution of 320 x 240 pixels. It could determine its own position and attitude relative to the lane, and furthermore it could track the relative state of up to ten other vehicles including their velocity components. In addition it could decide when to execute a lane change. Images were processed at a rate of 25 frames per second [Dickmanns, 1998].

A crucial mechanism in the system was the so called ”4D Approach to Vision”, e.g. [Dickmanns and Graefe, 1988], referring to object recognition by synthesis. This means that 3D shape models contained in a predefined database were perspectively projected into the vehicle’s view and matched to the observed scene (requiring a model of the sensors, i.e. the camera(s)). A known object was detected if the visual input correlated with the projection of a particular shape model. This way, 3D information was obtained top-down, in contrast to bottom-up approaches in which objects are reconstructed based on extracted image features. In addition to shape, also descriptions of the object’s dynamics were provided in form of dynamic models. These could be used to relate the object’s position to time, and thus the mechanism was termed the 4D Approach to Vision, where time was the fourth dimension in addition to position (which consists of 3 coordinates in a 3D Cartesian coordinate system).

By exploiting the database containing shape and dynamic object models, the system could set up and maintain an internal representation (Dickmanns called it ”world 2”) concerning its own state (position and orientation on the street), and the state of other objects (including heading, velocity, and position). Action commands were derived from this internal representation (a 3D Cartesian map) using designed control laws. It was also possible to predict future states of the plant (the system to be controlled) and other recognized objects using the map, the control laws and the provided shape and dynamic models by explicitly taking the plant’s own actions into account! This allowed predicting image features which led to the introduction of window based tracking, which enabled real time processing with very limited computing power at that time. For that, Dickmanns and his group applied recursive state estimators [Simon, 2006], in this case an extended version of the Kalman Filter [Kalman, 1960], and elegantly closed the loop between perception and action.

Many of the described mechanisms of this pioneering work are now used as standard techniques in several fields and applications, for example in visual servo control [Hutchinson and Corke, 1996].

Below we summarize the prominent properties of this approach:

- model based, i.e. heavy use of predefined shape-, dynamic-, sensor-models
- recognition by synthesis, thus no 3D scene-reconstruction required, but restricted to known shape models
- internal representation in 3D Euclidean space
- actions based on designed control laws
- mainly vision based, additional sensors were speedometers and/or odometers
- explicit coupling of actions and sensory input for prediction and recognition, using recursive state estimators
- introduction of window-based tracking

2.2.2 Stanley

The Grand Challenge was won in 2005 (no participant finished in 2004) by the robot Stanley[1], developed by a group from Stanford University lead by Sebastian Thrun. Stanley used methods from classic control-engineering but also methods from artificial intelligence (AI) research. The course of the race was provided two hours before the start, via a file in a DARPA specific format (RDDF). It described the course by 2,935 way points, associated speed limits, longitudes, latitudes, and corridor widths. "As a result, the race was primarily a test of high-speed road finding, obstacle detection, and avoidance in desert terrain" [Thrun et al., 2006].

Important issues concerning Stanley's control were the use of the unscented Kalman filter [Julier and Uhlmann, 1997] for state estimation and many (mostly active) sensors for building a 2D environmental map and the use of a vehicle model. Stanley was equipped with five laser range finders, GPS (global positioning system), two radar sensors, an inertial measurement unit, and one color camera. The system was realized based on the three-layer architecture [Gat, 1997], using asynchronous communication, i.e. each module was processed at its own pace. Probabilistic methods were used for handling errors in sensor recordings, e.g. Markov models [Rabiner, 1989] were applied to model noise in laser recordings for surface state assessment. Machine learning

[1]Although Stanley comprised many sensors of which vision only played a subordinate role, its description was included in this review, mainly due to the popularization of the DARPA challenges in the media, causing the impression that autonomous driving was a solved problem.

algorithms were employed, e.g. for discriminating drivable and non-drivable terrain based on driving examples from a human.

Control actions were obtained by computing a 2D map in absolute coordinates from sensory input, containing the plant's position and orientation as well as detected obstacles and the detected street. This map was used for path planning together with a base trajectory computed from the RDDF file. By varying the lateral offset of this base trajectory from the street several candidate trajectories were derived. By introducing constraints and costs an optimal trajectory was chosen by the planning module (based on a graph search algorithm). The final path was then processed by a closed-loop controller, using a predefined control-law, which worked by maintaining the lateral offset specified by the computed trajectory. Velocity was determined based on the slope of the terrain, detected obstacles, vertical shock, given limits contained in the RDDF file, and limits based on the dynamic constraints of the car. Human driving examples were used to relate velocity to vertical shock. It is reported that a limited look-ahead of sensor readings caused the system to conduct actions later than a human [Montemerlo et al., 2006].

In summary, its main features are as given below:

- various, foremost active sensors to acquire environmental information and the plant's state
- probabilistic approaches for handling erroneous sensor recordings
- focus on software architecture and asynchronous communication
- Unscented Kalman Filter for state estimation
- use of path planner
- limited look-ahead of sensor readings caused actions to be conducted later than by human
- steering control was based on a (nonlinear) control law and a vehicle model
- use of human driving examples to fine-tune control and perception

2.2.3 ALVINN

Quasi opposed to the control-engineering strategies are many machine-learning approaches to autonomous driving. A prominent example is ALVINN (Autonomous Land Vehicle In a Neural Network) [Pomerleau, 1989, 1991, 1990, 1999], which was developed at CMU to control the NavLab, Carnegie Mellon's test vehicle. It was reported in 1996 to have steered the NavLab car at distances up to 22 miles with a processing speed of 15 frames per second at a velocity of 55 miles per hour (88.5 km/h).

ALVINN worked totally different from the previously two described approaches. Instead of relying on predefined models, it learned driving from observing human behavior by means of an artificial neural network [Haykin, 1998] which served to associate observed human actions with concurrent visual input from a camera. It was a fully connected feed-forward network with a 30×32 input layer, four hidden units and an output layer with 30 units. Its input were the pixel values of the (downscaled) camera image and the output was a distribution over 30 possible steering angles. Speed control was handled by a human. By mapping camera frames directly to motor commands no transformation of the visual input into another representation was required. Indeed, no processing of sensory information (besides downscaling the image) was necessary. This means that no feature extraction or reconstruction of a 2D- or 3D scene was necessary.

Since the system learned from observing the driver it received positive examples only. This led to the problem that the system had no information on how to handle situations not covered by the provided examples, e.g. how to correct its course after having drifted off the street to some degree. This was solved by constructing two more views from the recorded image (by means of pixel coordinate transformations) showing the street as if driving to the left or the right from it, i.e. being off course and adding action information on how to return to the center of the road. This synthesized information was added to the training data.

Since the network's processing was covert (subsymbolic) it was not known which image features it used and how they contributed to the synthesized driving behavior.

In summary ALVINN's properties were:

- no need to pre-program explicit knowledge into the system in form of models, control laws, or for scene reconstruction purposes
- no need for an internal representation (other then the network state), and thus no computations necessary for its construction
- could be trained on any type of road
- worked with a single camera
- no insight into the subsymbolic process carried out by the network (opaque processing)
- actions were generated in a reactive fashion, i.e. no planning or prediction of actions
- need for special treatment of unknown cases (bad driving situations) due to training the system on positive examples only

2.3 Conclusion

Examples for different lines of research (control-theory, machine learning, and a hybrid approach) for tackling the task of autonomous driving were presented. In the following the strengths and weaknesses of both strategies are discussed and compared.

The characteristics of the control-theoretic approach are as already sketched in the Introduction. The system performances of the 4D approach and Stanley (concerning its control-engineered aspects) were highly autonomous. However, this performance rooted in the provided knowledge programmed into the systems, i.e. the models (shape-, dynamic-, and sensor models), the know-how to construct environmental maps, and design control laws which mapped sensed states to actions to achieve a desired behavior. As already explained, relying on predefined knowledge limits the system control in various ways and Dickmanns himself remarked 2002: “In the long run, even autonomous learning seems to be essential since not all knowledge should have to be programmed into the system by human developers.” In this tenor the conventional control of Stanley was extended by combining it with (advanced probabilistic) machine learning techniques, even using human driving examples to improve the system behavior.

Both approaches relied on internal representations in from of metric maps and given models. These equipped the systems with the ability to make predictions about changes concerning their own state and those of other entities with respect to their environment. For example, if the position and orientation within a metric map of a plant are known, it can be predicted how these will change in response to given action commands by using a given dynamic model of the plant. This ability to make predictions is useful in several ways (compare also section 3.2.1):

- By predicting the plant’s own or other object’s states, also the plant’s sensory input can be predicted. This can be used to filter noise in the perceived sensor input by comparing it to the expectation, i.e. the prediction (which lead to the introduction of window-based filtering by the group of Dickmanns, a standard technique today). The filtering improves the processing of sensory information substantially by reducing the rate of false positives, making it thus more reliable, and furthermore making it more efficient by decreasing the necessary amount of information to be processed. For example, in the case of visual input it suffices to consider small image-windows instead of entire frames.

- It enables the plant to generate actions even in case of (short) sensor outages. This is possible by generating action commands based on a determined state, then predicting the resulting next state and again determining an appropriate action for this predicted state and so on.

- If the determined actions are not executed, then the same principle from the item above can be used for the simulation of actions. This can be used to generate action plans on one hand and on the other hand to achieve forms of reasoning as discussed in section 3.2.1.

Despite all these advantages, such internal representations in form of metric maps and given models are expensive concerning the necessary given knowledge and the required computations for the map constructing. For example, during the mapping of 3D scenes to 2D by a camera, in the case of visual input, the depth information is lost and the perspective effect introduced. The latter describes that distances and object sizes in the projected image decrease the further they are away from the camera. This prevents a simple determination of absolute distances, (curved) street segment lengths, or street curvatures etc. But this information is necessary for white-box approaches describing motion based on kinematics and dynamics. To obtain these measurements two approaches are commonly applied: One is to recover the depth information directly, usually by using a stereo camera setup (requiring a second camera), and acquiring a 3D map from the resulting 3D scene reconstruction. The second approach is to re-project image pixels to a known, flat ground plane, which results in a top view of the recorded scene with the perspective effect eliminated from which a 2D map can be acquired. In the domain of street detection this coordinate transformation is often referred to as inverse perspective mapping (IPM) [Bertozzi and Broggi, 1998; Mallot et al., 1991]. Both techniques (3D reconstruction using stereo information and IPM) require the developer to be familiar with the field of projective geometry and the knowledge of so-called extrinsic and intrinsic camera parameters. The former relate the cameras to some global coordinate system and the latter refer to specific camera information, e.g. pixel size, focal length, and distortion parameters. Camera calibration can be a tedious process and already small perturbations, for example vertical shock, can affect a calibrated setup, i.e. change the values of the ex- and intrinsic parameters. Then the quality of the reconstruction results decrease and with it the system's performance, resulting in the need to recalibrate which can be inconvenient for long-term practical use.

In summary, systems based on traditional control-theoretic approaches to driving can perform highly autonomously but at the cost of being limited to specific knowledge provided externally by experts. The use of a 2D- or 3D environmental map built from visual sensory input allows relating measurements to control laws, which are based on physical relationships, but requires the elimination of the perspective effect in the recorded image frames and/or to recover the lost depth information. This is expensive in terms of necessary a priori knowledge (determination of in- and extrinsic camera parameters, developer trained in projective geometry), computation time (needed for realizing the coordinate transformation of IPM or the depth reconstruction from stereo, longer development cycle, need for camera (re-)calibration), and money (due to the longer development cycle and also increased hardware prices in the case of using a second camera).

By contrast, ALVINN, which was presented as representative for machine learning based approaches to driving, performed well, too, without requiring large amounts of given knowledge. The system learned the skill of driving simply by observing a human teacher! Actions were mapped to visual situations by means of a neural network, which is an example of a nonlinear system identificator [Chen et al., 1990]. This example meets several requirements suggested for effective system control. It is autononmous, adaptive and easy to instruct. In addition it makes very few requirements concerning hardware and pre-coded knowledge, leading to low costs. In particular it shows that efficient mapping between sensory input and action output does not necessarily require knowledge of an underlying structure, which was pointed out earlier to be an important feature in skilled animal behavior, too.

One problem related to the learning from a teacher in the described setup was that only positive examples could be given. To provide negative examples, the driver had to first bring the car into a bad situation, e.g. drive off the track and then show the system how to return. However, in this case the system would also learn the action to drive off the street, which is certainly undesired. In addition, it would be awkward if a (commercial) system had to be taught like this.

Another important feature of ALVINN is that it functioned without an internal representation about its environment (other then the current network state). However, this also limited the system, planning and/or predictions usable for various purposes leading to an improved overall system performance as described above, were not possible.

Therefore, augmenting a reactive black-box system like ALVINN with the capacity of prediction making and planning would constitute a considerable improvement in autonomous driving control. If and how this can be achieved using knowledge about skill learning is treated in the next chapter. Thus, this thesis is to be placed into the field of machine learning approaches to autonomous driving.

Chapter 3

Learning the Skill of Driving

”The only source of knowledge is experience.“
Albert Einstein

Driving a car is a skill that we humans are obviously not born with but can acquire during our lifetime. This ability to learn new skills enables us to dynamically adapt to new or changed situations and environments which is also a desired property for autonomous systems. Identifying the relevant mechanisms realizing skill acquisition is thus a first step towards investigating their application to the task at hand, i.e. learning anticipatory driving from a human supervisor. In this chapter theories and experimental results related to human skill learning are reviewed and yet unresolved issues related to the work in this thesis are emphasized. The chapter is concluded by stating which of the reviewed information can be applied to the stated tasks of this thesis.

3.1 Skills

Before tackling skill acquisition we review what skills are and by which features they are characterized.

A skill is an ability that is not contingent, but acquired during lifetime. Rosenbaum 1983 defines a skill as: ”...an ability that allows a goal to be achieved within some domain with increasing likelihood as a result of practice”. The literature distinguishes between different kinds of skills, where the broadest distinction is made between intellectual (also called cognitive- or higher-cognitive) and perceptual-motor (or lower-cognitive) skills. Here, we will use the terms intellectual and (perceptual-) motor skills. Examples of intellectual skills are: playing chess, programming computers, solving mathematical problems, etc. Examples of perceptual motor skills are: playing instruments, doing sports, grasping objects, etc. Intellectual skills operate on declarative knowledge (knowledge about *facts*, i.e. *what*) and motor skills require procedural knowledge (knowledge about

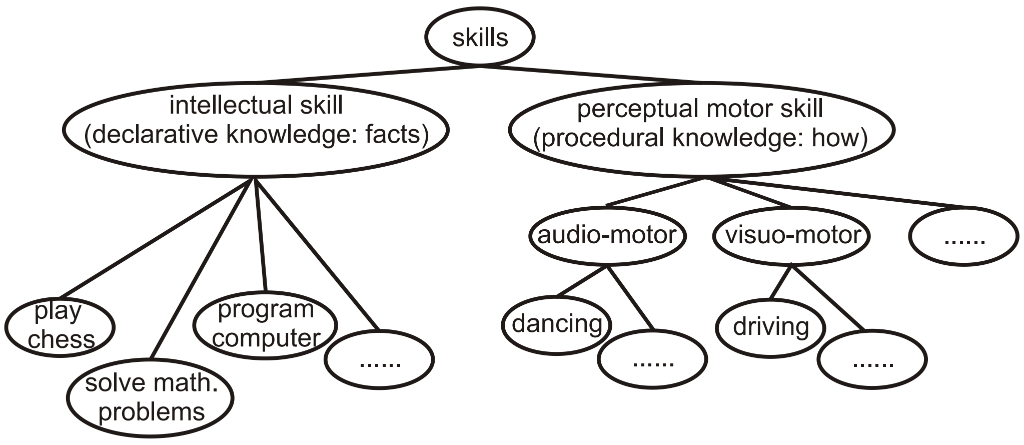


Figure 3.1: A taxonomy of skills.

how actions are executed). Perceptual-motor skills are further distinguished according to the type of sensor providing the necessary information for accomplishing a task. For example, visuo-motor skills are those in which visual information is guiding the action like in reaching, grasping, and visual locomotion. Also the task of driving can be considered to be mainly a visuo-motor skill (assuming that other sensory inputs such as from the vestibular system are negligible). A sketch of this presented taxonomy is given in Fig. 3.1. Furthermore it is known that skilled behavior is domain specific and does not transfer from one domain to another [Ericsson and Lehmann, 1996].

Intellectual skills, a traditional area of AI research, are often regarded as an indicator of an individual's intelligence while motor skills seem to be considered as somewhat simpler - someone with the capacity of solving a difficult mathematical problem is usually ascribed more intelligence than someone with the capacity of riding a bicycle. Despite this rather subjective impression, it turned out to be very challenging to equip robots with motor skills which reflects the difficulty of (formally) describing procedural knowledge. Also, the diversity between these two presumably different types of skills was questioned as it was indicated that both are acquired involving the same types of mechanisms [Rosenbaum et al., 2001]. From the developmental sciences it is known that both types of skills are closely related. It has been even suggested that the learning of motor skills forms the basis for all intellectual skills, which comprises the cognitive processes reasoning and planning. For example, in 1952 Piaget proposed that "intelligence sensori-motoric" precedes and provides the foundation for "intelligence intellectualle". Today this is sometimes termed *motor cognition*, [Jeannerod, 2006]. A further discussion on motor learning is given in Halsband and Lange 2001.

In summary, skills comprise motor and intellectual skills, they are learned, domain specific, and it is hypothesized that the acquisition of motor skills forms the basis for important cognitive processes. In the following we will focus on the question *how* (visuo-)motor skills are acquired, and what the proposed requirements of the underlying mechanisms are.

3.2 Human Motor Skill Acquisition

The process by which we acquire skills is investigated by the cognitive sciences involving several disciplines tackling the problem at various levels using different techniques. In psychology behavioral data, (e.g. reaction times, response to stimuli, eye tracking, etc.) is collected and analyzed, while in neuroscience the underlying neural mechanisms are identified by means of neurophysiological studies and imaging techniques, and finally, using computational modeling, our understanding of proposed processes is formalized making use of computers and mathematics.

In the following we will review the theory underpinned by contributions from all of the said disciplines that **motor skill learning is a staged process comprising a) an early phase were learning is slow due to the establishment of so called *internal models* and b) a late phase were learning is automated due to a process known as *chunking***.

According to the classic theories based on behavioral data postulated by Fitts and Posner in 1967 and also by Bernstein in 1996, motor skill acquisition comprises three phases or stages:

1. **The cognitive or early phase**: During this stage the subject understands the requirements of the task, e.g. on the basis of verbal or written instructions, or by observation. He or she discovers the correct sequence of actions for accomplishing the task and a gross motor plan is generated. This requires the analysis of the consequences of executed actions and thus the learner's attention. The improvement in performance is slow and the behavior typically uncoordinated and jerky.

2. **The associative or intermediate phase**: During this stage the learner practices what was learned during the cognitive phase and associations between specific cues and required movements are established. Feedback control is used to detect and eliminate errors and gradually anticipation develops. The performance becomes more stable and smooth.

3. **The autonomous or late phase**: Reaching this stage requires persistent physical practice and leads to an automation of the required behavior. It is characterized by little increase in the subject's performance and in contrast to the early stage the learner requires no, or only little, conscious effort for accomplishing the task. Once automated, the acquired procedural knowledge is available for very long periods of time indicating efficient storage mechanisms.

Very much in compliance with this behavioral theory are findings from neuroscience, which suggest a division of motor skill learning into an **initial**, an **intermediate**, and an **advanced phase**[1] where the former two are jointly referred to as **early** and the latter as **late phase**. A comprehensive review of human brain structures and their role during the individual phases of skill learning based on collected results from functional imaging, electrophysiological, and cellular/molecular studies is

[1]Finer classifications also include a consolidation and retention phase.

early phase	**late phase**
attention and conscious control necessary	no conscious control necessary
closed-loop	open-loop
variability in performance	stable performance
fast learning (within session)	slow learning, requires continued practice
slow perception (reaction times)	fast perception
suggested mechanism: internal models	**suggested mechanism: chunking**

Table 3.1: Distinction between early and late phase in motor skill learning and the proposed mechanism realizing the stages.

given in [Halsband and Lange, 2006] and focusing on more specific aspects in Hikosaka et al. 2002 and Bastian 2006.

In short, motor skill learning can be subdivided into different phases characterized by an increase in: precision of the performance, persistance (meaning the time periods over which the learned behavior is remembered), the speed of execution; and a decrease in: conscious attention, variability in the performance, and inference with other skills (meaning that a newly acquired skill during early learning can be disturbed by other, already obtained skills) [Luft and Buitrago, 2005; Karni et al., 1998; Ericsson and Lehmann, 1996]. This is summarized in table 3.1.

There are strong hints that the slow learning in the early phase is caused by the establishment of so-called internal models, i.e. mappings between sensory and motor information which can be acquired via feedback learning [Kawato, 1999; Wolpert et al., 1995; Flanagan and Wing, 1997]. The idea of internal models originated in control theory [Francis and Wonham, 1976] and will be treated in section 3.2.1 in detail.

The transition from slow moment-to-moment into fast and automated control during the late phase is usually ascribed to a mechanism called chunking during which bits of information are grouped into larger entities, the chunks. This will be treated in section 3.2.2.

3.2.1 The Early Phase and Internal Models

The characteristics of the early phase in motor learning (compare table 3.1) can be explained by the acquisition of internal models [Kawato, 1999; Wolpert et al., 1995; Flanagan and Wing, 1997], a concept initially used in control engineering [Francis and Wonham, 1976]. Internal models consist of two distinct types: *forward* and *inverse models*. Their formal definition requires some central concepts which are defined in the following:

A dynamical system, i.e. a system which changes over time, is said to have, or to be in, a certain *state* at a certain moment in time which is characterized by a set of *state variables*. To predict a future state of the system its current state must be known and in addition other external factors influencing it. For example if the system of interest is a knee joint, the state could be defined by its angular configuration and the external factor to be considered, also called the *control input*, might be a torque applied to the joint. The prediction of the future state can then be described by the *next-state* or *state-transition equation* given in Eq. 3.1,

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{next state equation,} \tag{3.1}$$

where the vector $\mathbf{x}_t$ denotes the state of the system at time[2] t, and the vector $\mathbf{u}_t$ denotes the control input. Together with the *output equation*, given in Eq. 3.2, which models the output that the system might yield, e.g. some resulting angular velocity in case of the knee-example,

$$\mathbf{y}_{t+1} = g(\mathbf{x}_t) \qquad \text{output equation,} \tag{3.2}$$

one obtains a formal description (model) of a dynamical system (as used in control theory).

A forward model is basically the next-state equation, i.e. it predicts the next state of a system based on the current state and some control input:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) \qquad \text{forward model,} \tag{3.3}$$

An inverse model takes a current state and a desired next state and returns the required control input which will transfer the system to the desired next state, as formalized in Eq. 3.4

$$\mathbf{u}_t = h(\mathbf{x}_t, \mathbf{x}_{t+1}) \qquad \text{inverse model.} \tag{3.4}$$

In the case of motor skill learning a forward model is considered to be a process in the central nervous system, [Wolpert et al., 1995]. In this case the state of the system corresponds to the perceived sensory cues by the human and the control input corresponds to the conducted action, or rather the neural activation encoding the action. Since a forward model of a certain process predicts how a conducted action transforms a current state it can be used to simulate this process. Concerning skill learning, such an internally modeled process could for example be one's own musculoskeletal dynamics, in other cases it could be a known person, or an abstract mental action. In short, an internal model captures information about the plant, and/or processes or entities in its environment. (Note the similarity to dynamic and kinematic models of white-box approaches to system control.) The ability of forward models to predict the outcome of a particular motor command, the next state, can explain four problems in motor learning [Wolpert et al., 1995]:

- The predicted next state of a system given by the forward model can be used by an inverse model to determine the next adequate control before the next state is actually perceived. This can explain why movements can be carried out rapidly despite known large delays in the sensorimotor loop [Desmurget and Grafton, 2000].

[2]Note, time can be continuous or discrete. Here, we use discrete time.

- A prediction of the sensory input caused by self-induced effects is also referred to as *reafference*. By subtracting sensory input (*afference*) from the reafference it is possible to obtain only those effects caused by external stimuli, *exafference*, [von Holst and Mittelstaedt, 1950; Gallistel, 1980; Blakemore et al., 2000]. For example, if we move our eyes by a few degrees the efference copy, i.e. a copy of a motor signal to the periphery, allows to predict that the next perceived image should also shift by these few degrees (reafference). If this shift is subtracted from the actually perceived image (afference), then the real shift is zero and the image appears stable (exafference). For example, this can explain why we perceive a visually stable image of the outer world although our eyes move. (This is easy to demonstrate by carefully moving one's eyeball with a finger. In this case the perceived image is not stable because no exafference was computed.)

- The discrepancy between the predicted and actually perceived sensory state can be used for motor learning, e.g. feedback error learning [Kawato et al., 1987].

- The prediction can be used for mental practice to learn the evaluation of actions for later use such that best actions can be chosen to achieve a goal. This can be used for action-selection, decision making and planning [Sutton and Barto, 1981].

Furthermore, a forward model is necessary for state estimation, see section A.16, and yet another far-reaching hypothesis concerning the capacity of internal simulations by means of forward models is that they form the basis for an internal representation of the world required for higher cognitive processes like action planning, thought, and speech, [Hesslow, 1994; Clark and Grush, 1999; Hesslow, 2002; Grush, 2004].

An inverse model on the other hand can be used for control, i.e. for identifying the necessary motor commands for obtaining a desired state or output. In fact, an inverse model *is* a predictive controller as used in control theory [Jordan, 1996].

Internal models can be acquired by the subject by (possibly randomly) conducting actions. In the case of early development this is referred to as *motor-* or *body babbling* (e.g. [Meltzoff and Moore, 1997]). During this phase the subject observes which neural activation (action) causes the transition from which sensory state to another (perception) and the associations between input state, control input, and output state are encoded in memory which allows the learning of arbitrary sensor-motor maps. Such a map can be used as a forward model, and given a desired behavior is known, i.e. a desired next state, it can also be used as an inverse model. The more experience the subject collects during practice, the more precise and robust against interference are the models, thus the interaction of the individual with its environment, called the *perception-action cycle*, and the ability to memorize experienced transitions, is of utmost importance.

Concerning computational modeling suggested computational learning algorithms for the acquisition of internal models based on supervised learning are: direct inverse modeling, [Widrow and Stearns, 1985], distal supervised learning, [Jordan and Rumelhart, 1992], and feedback error learning [Kawato et al., 1987].

It is also important to note that the associative learning used to establish the sensor-motor map allows the capturing of a system in a black-box fashion, i.e. the behavior of the system can be simulated without the need to rebuild its underlying structure e.g. the trajectory of a propelled object can be estimated without the need of integrating the equations of motion [Jordan, 1996]. (Note, that this is opposed to white-box approaches to system control.)

In summary, forward models capture the behavior of a system and can thus serve as an internal description of it, allowing to "simulate" the system. To acquire a forward model of a system it must be possible to observe the system's input and also its response. In addition, this observed information, the experience, must be encoded in memory. Note, that the theory just reviewed does not explain how this encoding is realized nor what exactly is encoded. From the viewpoint of this thesis, i.e. with a practical application in mind, observed information is limited (see the problem of learning from positive examples only, as occurred with ALVINN), i.e. a forward model must be able to inter- and possibly extrapolate from its experience.

Inverse models can be used as controllers which determine the necessary action for the transition from a current state into a goal state. Although such information can in principle be learned from data it is important to note that a goal state must be known! The information given above does not capture the important aspect of how this information can be derived!

3.2.2 The Late Phase and Chunking

The characteristics of the late phase in skill learning (compare table 3.1) can be explained by a mechansim called chunking. Chunking is considered to be involved in both – intellectual and motor skill learning and in its very basic form says that bits of entities are re-encoded, or chunked, into a few larger entities – the chunks. During intellectual skill learning bits of information [DeGroot, 1978; Simon and Chase, 1973], and during (sequential) motor skill learning primitive actions [Rosenbaum et al., 1983; Koch and Hoffmann, 2000] are chained together.

The term chunking was introduced by Miller 1956 when investigating the limitations of human working memory, and concerning intellectual skill learning it was later further elaborated to refer to a general theory of expertise acquisition by Simon and Chase 1973. This research was heavily influenced by early theoretical and experimental work of DeGroot 1949; 1978 who studied the effect of practice in intellectual skill learning (by means of chess). Also a computational model (MAPP) was developed that implemented basic aspects of the chunking theory in Chase and Simon, and Simon and Gilmartin 1973; 1973 revealing some shortcomings which were overcome by introducing a modified version of chunking, called templates [Gobet and Simon, 1996; Gobet and Chassy, 2009]. The theory aimed at explaining behavioral data concerning expert performance[3].

[3]Experts are considered to outperform most other people at a domain-specific task, which is usually accomplished

Research in expertise showed that expert performance differs systematically from that of non-experts in both intellectual and motor skill related domains: e.g. chess, physics, mathematics, medical skills, and baseball and golf, (compare [DeGroot, 1949, 1978; Simon and Chase, 1973; Gobet and Simon, 1996; Ericsson and Lehmann, 1996; Reingold et al., 2001a; Charness et al., 2001; Reingold et al., 2001b]). In short, experts can – in comparison to non-experts:

- react faster to presented stimuli
- scan presented stimuli in a distinct way
- handle a much greater volume of domain specific information (not other)
- show much higher memory capacity
- perform better, i.e. faster and more reliably

These observations lead to the following conclusions: Since the capacity of the working memory (a theoretical structure suggested for the temporary manipulation and storage of information [Baddeley, 2002]) is known to be limited [4] and since the amount of information handled by experts exceeds these limits, the used information must somehow be condensed [Miller, 1956; DeGroot, 1978; Simon and Chase, 1973]. This mechanism of condensing information was termed *chunking* and the condensed information referred to as *chunk*. A classic example is remembering a telephone number consisting of several digits, e.g six. Remembering each digit means that six entities are required to be 'loaded' into working memory. If the same number is remembered by two groups of three digits each, only two entities are 'occupied' *and* more information is available. In addition, since the learned information, i.e. the chunks, are available for a long time (up to years) and robust against interference the storage of this information was suggested to take place in long term memory (LTM), e.g. [Charness, 1976; Simon and Chase, 1973; Ericsson and Kintsch, 1995; Gobet and Clarkson, 2004]. Furthermore, due to the short reaction times of expert performance, an efficient retrieval mechanism was assumed, and since experts scan presented stimuli in a distinct way it was proposed that they learn to recognize the most informative patterns in the presented sensory information and that such patterns serve as pointers to the chunks in LTM [DeGroot, 1949, 1978; Charness et al., 2001; Curby and Gauthier, 2009; Fiser and Aslin, 2005], similar to a stimulus-response type reaction. Finally there are experimental results suggesting that the organization of information units is hierarchical which also contributes to the faster reaction times [Freyhof et al., 1992].
In summary, a chunk is considered to be:

after long terms of deliberate practice [Ericsson and Lehmann, 1996]. For example, a period of 10 years was suggested by Siman and Chase 1973 to acquire the performance of a top-level chess player.

[4]A limitation of seven +/- two items was proposed by Miller 1956, but later corrected to three or four [Cowan, 2001] items.

- a long-term memory information
- grouped in a meaningful way as a single, ”perceptual unit”
- retrievable in “one act of recognition”
- pointed to by learned perceptual patterns
- presumably hierarchically organized

Thus, expert performance is considered not to be caused by a “higher general processing ability” [Gobet and Clarkson, 2004], but rather explainable as a memory effect, where experts have acquired a large database of chunks and learned to identify informative patterns in the sensory input which trigger particular chunks.

Apart from the chunking theory other alternative explanations were offered in the behavioral domain, however less well defined.

A comparison and discussion of different theories can be found in Gobet and Chassy [2009].

The above information was related to intellectual skills. Concerning motor skills, the term “chunk” is also often used in a more general way in the literature, referring to a sub-element of a longer motor sequence, ordered in a hierarchical fashion. Sometimes this is also referred to as motor primitive. However, here, we use the word chunk in the sense of the chunking theory as just described. The formation of chunks for motor control has been proposed by e.g. Rosenbaum et al. [1983]; Freyhof et al. [1992]; Koch and Hoffmann [2000]; Sakai et al. [2003]; Hoffmann [2007]. It was shown in Sakai et al. [2003] that same action sequences were decomposed into different chunks by different people and that the formation of chunks lead to a more efficient processing of a given visuomotor sequence. It was also shown that chunk formation can take place without conscious awareness of the subject, i.e. implicit learning can precede explicit knowledge, [Nissen and Bullemer, 1987; Koch and Hoffmann, 2000].

Neurobiological processes related to chunking have been suggested to take place in the basal ganglia [Graybiel, 1998]. The involvement of the dominant parietal area, basal ganglia, and presupplementary motor area in chunking were argued for by Sakai et al. [2003], and Sakai et al. [2004] suggested that the formation of chunks take place in the cerebellum.

Concerning computational modeling, chunking was formalized in the computer program EPAM, Elementary Perceiver and Memorizer Mode [Feigenbaum and Simon, 1962, 1984; Richman et al., 1995]. It was also used in the cognitive architecture ACT-R (Adaptive Control of Thought-Rational) where it was first applied to intellectual and later to perceptual motor skill learning [Anderson et al., 2004].

In summary, expert behavior which is obtained after long deliberate practice, i.e. related to the late phase of skill learning, is characterized by very fast and good performance of the subject and can be explained by the acquisition of a database containing compiled sequences of actions indexable by informative patterns of perceptual input. Note, that this does not explain which rules chunk formation underlies. Do chunks correspond to structured and meaningful entities, e.g. an entire, or a particular part of a grasping motion, or are chunks arbitrary parts of longer sequences formed by mere statistical frequency. Also the question of generalization remains unanswered, by which is meant that chunks should not only be usable for the exact same perceptual cues but also for similar ones. One option is that generalized versions of chunks are stored. Another possibility is to store exact sequences. This would be relatively memory intensive and require a generalization at some other point in time, e.g. at retrieval time.

3.2.3 Forward Models, Chunking, and Deliberative Behavior

We have seen from the above that internal models and chunking can both lead to action plans. In the following we want to compare these plans also with respect to deliberative behavior.

The coupling of inverse and forward models can be used for internal simulations which generates a sequence of state-action pairs. For convenience we denote such a sequence as “planC”. A chunk triggered by some specific sensory input pattern is considered to be a sequence of actions. Thus, an obvious difference between planCs and chunks is that the former contain more information in form of the states which are expected to be encountered. Since planCs are iteratively generated based on predicted states they are more flexible in comparison to chunks which are fixed sequences. Which seems to be a disadvantage concerning chunks actually has benefits. Since chunks are said to be evoked by a stimulus-response mechanism, capacities which are used for the iterative generations of planCs are not necessary and can be used for other purposes. Also, since chunks require no internal reasoning entire action sequences are available almost immediately. In fact, one characteristic of expert behavior which is explained by chunking is increased speed of performance.

Another obvious difference between planCs and chunks is that the former require knowledge of a desired next, or final state. The sequence is then generated to achieve this goal, i.e. deliberative behavior is enabled. Chunks cannot provide this. However, chunks are, according to the reviewed information, only compiled after a long time of deliberate practice. Therefore, it is likely that created chunks stem from such action sequences that successfully lead to a desired goal-state. Hence, one can assume that a goal is implicitly given when a particular chunk is recalled.

In summary, planCs and chunks both contain action sequences. PlanCs are more flexible but expensive to obtain, since generated iteratively. They require an explicit goal and thus enable deliberate behavior. Chunks can be considered to be inflexible and fast to obtain. They are reactive but it can be assumed that a goal (intention) is implicitly given.

3.3 Application to Driving

After having reviewed theories about human skill acquisition now the question is tackled which of the hypothesized mechanisms can be applied to the task of learning anticipatory driving from a human teacher and how to realize it.

As stated in the task specification in section 1.6 the human teacher is expected to not be a novice driver but experienced and thus beyond the early stage of skill learning. It therefore seems reasonable to equip the system with the necessary means to realize the subsequent late phase, i.e. to **apply chunking to the given task**. As reviewed above, chunking denotes a process during which chunks, which can be compiled action sequences, are stored in long term memory indexable by perceptual cues (compare section 3.2.2). In other words actions are mapped to a cue-vector, interpretable as a state-vector, i.e. the set of variables that describe the current state of a system, see section 3.2.1. With this interpretation the process is equivalent to policy learning in reinforcement learning [Sutton and Barto, 1998] which is also a well-known method for imitation learning [Dautenhahn and Nehaniv, 2002; Argall et al., 2009]. However, instead of storing single-step actions, here sequences, i.e. chunks are used.

To realize chunking a mechanism for generalization is necessary, as pointed out in section 3.2.2. The system should be able to answer to unknown situations, but which are similar to what the system has experienced before, with adequate control output. Thus, to realize chunking/policy learning, the following steps are required:

- Identification and extraction of those cues in the sensory input that guide the task. This is also called feature selection and is related to the problem of state definition as explained in section 1.1.
- Creation of a database to hold sense-act pairs, i.e. cue-vectors and action sequences, which constitutes the memory of the system.
- Definition of a suitable chunk retrieval mechanism and means for prediction and generalization.

Further details concerning the realization of the given task are given in the following.

3.3.1 Feature Selection

Concerning the first item, identification of task-relevant sensory cues, two strategies were compared in section 2.3. One is to compute those parameters from the sensory input that are required for used white-box control laws, e.g. absolute measures of distances, lengths and curvatures. In this case the cues are overt but expensive to obtain in terms of the introduced dependence of given a priori knowledge and the necessity to remove the perspective effect. Another option was to use a black-box approach to directly map sensory input to actions, which leaves the cues disguised but is cheap in the sense that no further given knowledge and also no image processing is needed. Here, a compromise is made between the use of given knowledge and a black-box approach. Because it is obvious that the trajectory of the lane is determining the actions, the system is equipped with a functionality for lane extraction. A condensed version of the extracted lane is then mapped to actions. However, the computations of absolute measures, removal of the perspective effect and/or 3D reconstructions are explicitly avoided and all perceptual information used is purely in 2D image coordinates. This constitutes the postulated economical factor stated in the hypothesis in this work (section 1.5), that the system is supposed to work with as little given knowledge as possible.

3.3.2 Generalization

The third item from the enumeration above indicated the need for prediction and generalization mechanisms. This is motivated as follows: One goal of this work is to enable the system to predict sequences of actions, i.e. action plans adequate for a currently perceived situation based on observations of human driving. Since these observations, the training data, are limited, i.e. not containing every possible cue combination, the system (and also humans) must have means for generalizing from these examples to new situations. As pointed out to in section 3.2.2 this process is not clearly specified in the skill-learning research.

In the machine learning domain several strategies for supervised learning are known that, according to Aha [1997] can be divided into two classes, *eager*- and *lazy-learning* methods which are applicable to the problem of generalization from given example data. A second possibility is to combine a given plan with a closed-loop controller which adapts it to the momentary state on-the-fly if necessary. This also constitutes a form of generalization. Both options are explained in the following.

Eager- and Lazy-Learning

Eager- and lazy-learning methods are applicable to problems in which the goal is to predict some output $f(x)$ from presented input data x, based on given example pairs $(x_i, f(x_i))$, the training data, with $1 \leq i \leq l$), where l is the number of samples. This is also the problem in the task given here.

Eager-learning methods compute $\hat{f}$, a global estimation of the true function f, from the given examples during training, after which the training data is not needed anymore. The training- or learning phase can be time consuming, however making predictions can be fast, since it only requires the input to be processed by $\hat{f}$. Examples are global function approximators like certain kinds of artificial neural networks, or decision trees.

Lazy-learning methods (also called instance-based or local-learning methods) in contrast keep the training data and based on it compute a local approximation of $f(x)$ at retrieval time. Thus, learning is fast, since it consists only of storing the training data, but it can be memory intensive and possibly slow at retrieval time. Examples are locally weighted regression techniques, or radial basis function networks (compare appendix B.2). Lazy learning algorithms do not suffer from data interferences, as is the case with for example certain types of artificial neural networks (compare appendix B.1). When training the latter it must be ensured that the given training data is well distributed and that no information is overrepresented. In this case the performance of the trained network would be distorted in favor of the overrepresented examples. In addition lazy learning techniques often allow more precise approximations of $f(x)$ due to their local nature.

A more thorough comparison of eager- and lazy learning can be found in Mitchell [1997, chapter 8], or Aha [1997] and Bottou and Vapnik [1992].

Since lazy learning involves the storage of experienced information similar to what was suggested in the reviewed information on skill learning which indicated that chunks are stored in long term memory, a lazy learning approach will be used in this work.

Reactive and Planned Control

Another way to generalize from given examples is to add a closed-loop reactive controller which alters a planned trajectory on-the-fly. This has been used for path planning in the Stanley example in section 2.2.2, and is also biologically plausible, as discussed in Daw et al. [2005]; Glover [2004]. Furthermore, such a two-fold mechanism was suggested to be involved in human steer control in car driving in Donges [1978].

Hence, in addition to lazy learning for generalization we will introduce a reactive controller which can alter a given trajectory online if fine-tuning is required.

3.3.3 Summary

In summary, the chunking mechanism, proposed to play a role in the late phase of human skill learning can be related to policy learning. It will be applied to the given task of learning to drive and the required visual cues are partly preselected by explicitly detecting visible lane markers in the image frames, however avoiding the construction of a metric map. To realize chunking and the necessary generalization a lazy-learning approach will be chosen and a reactive controller for online adaptation. The following chapters deal with the realization of the chunking process first for the indoor scenario (4.2, 4.3), and then for the real-car application in 6, beginning with a description of the (indoor) lane detection mechanism in 4.2.

Chapter 4

Indoor Driving and Chunking

"The more you learn, the more places you'll go."
Dr. Seuss

In this chapter chunking is realized and applied to the problem of learning driving from a human supervisor in the indoor scenario, described in section 4.1. According to the previous section 3.3 this requires the realization of three mechanisms of which one is the identification and extraction of task-relevant sensory cues (item 1). These can then be used for the creation of a database, serving as index to retrievable action chunks (item 2), requiring means for generalization (item 3). Due to the complexity of item 1, it is treated in its own section (4.2) which is highly self-contained. Its output is then used in the main section, 4.3 (published in [Markelic et al., 2008]), which tackles items 2 and 3. It describes the application of chunking to the given task, and evaluates and discusses the approach. The result is a system which learns the driving task with the ability to produce action plans without the need to construct a metric environmental map or sophisticated predefined model-knowledge. As such it constitutes a novel and alternative form to previous autonomous driving control with the advantage of being simple and cheap which makes it very attractive for industrial purposes, showing the potential of using skill-learning as paragon.

4.1 Experimental Indoor Setup and Notations

The experiments in the indoor environment were carried out on a four-wheeled robot (a modified VolksBot [Volksbot, 2000]) shown in Fig. 4.1(a) of size 50×60 cm with two motors for differential drive, i.e. one motor for driving the wheels on each side. It was equipped with a firewire black and white camera, "Guppy" from Allied Vision Technology (in fact the robot was equipped with two cameras, however, only one was used for the purpose of this work). The used varifocal lens was from the company Compaq with possible focal lengths between 1.8-3.6mm. The laboratory setup simulated a street environment, shown as a sketch in Fig. 4.1(d), where the driver could control the robot from a special station, see Fig. 4.1(b). From there he or she could see ”through the robot’s eyes” by means of a TV on which the robot-camera output was displayed, a typical view is shown in Fig. 4.1(e). Furthermore, the driver could control the robot’s heading direction and velocity by means of a game steering wheel and pedal set. The communication between human control output (velocity and directional commands) and robot sensory input (camera frames) was realized with a peer-to-peer architecture between a laptop placed on the robot and a desktop computer. The laptop, connected to the vehicle’s motors and camera, sent the latest image frame to the TV, waited for a subsequent control input (generated by the human) from the desktop computer, which was connected to the steering wheel and pedal set, and passed it to the vehicle-motors, compare Fig. 4.1(c).

We refer to such a perception-action cycle as *timestep*, in which we measure time t. Thus, the notation “time (timesteps)” means that time is given in discrete sense-act cycles. Note, that the duration of each such timestep is variable with 30ms $\leq t \leq$ 150ms, where the average duration is 83ms.

For the control input we use the following notation: st denotes a single steering- , and vel a velocity command. Both signals take numerical values with $st \in [-128, 128]$ related to the steering angle, where -128 maps to 180° left and 128 to 180° right steering, and $vel \in [-512, 512]$ is related to the voltage sent to each motor, where -512 maps to driving backwards at full speed (2.2m/s with fully loaded batteries), 0 maps to zero velocity, and 512 to going forward at full speed. Thus, the notation $steer \in [-128, 128]$ or $speed \in [-512, 512]$ indicates that control is given in these numerical values, instead degrees or meters per second.

To simulate night-driving conditions, the robot was additionally equipped with three infrared beamers that were placed below the camera as shown in Fig. 4.2(a). Furthermore, the camera lens was replaced by one without infrared light cut filter. A typical indoor “night-driving” view is shown in Fig. 4.2(b).

4.2 Lane Detection

To accomplish a task in the world any intelligent agent must have the ability to extract the task-relevant information from its sensory input. In the case of vision-based lane-following this means the agent must be able to detect the lane, or some indication of it, in the incoming image frames (compare also section 3.3.1). Here, we introduce a feature-based lane marker detection method, tailored to meet the requirements of the indoor-robot scenario. In contrast to many other reported algorithms for “real-world” street detection, it does not consider occlusions or interruptions of the lane, but instead focuses on the detection of arbitrarily shaped high curvature curves while requiring little a priori knowledge. Kalman filter based tracking is applied to reduce the rate of false-positives.

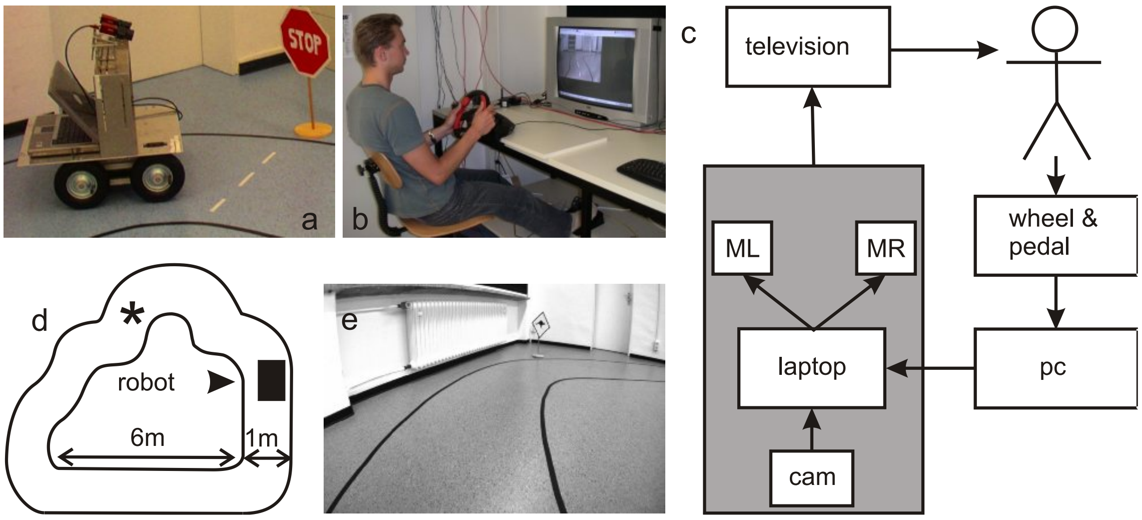


Figure 4.1: a) A car-like robot. b) Control station. c) Sense-act cycle in the experimental setup: cam denotes camera, and ML and MR left and right motor, the shaded area indicates the robot. d) Sketch of the track used for training, the "*" marks a difficult turn. e) Typical scene recorded by the robot.

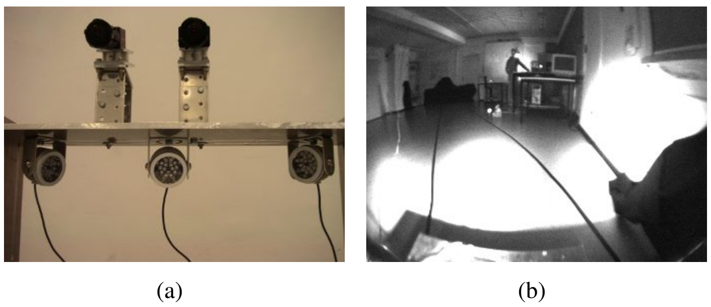


Figure 4.2: (a) Three IR beamers mounted on the robot. (b) A typical indoor "night-driving" view.

The method works in image coordinates and does not reconstruct a predefined street model. It works with a single, uncalibrated, monochrome camera and is shown to be fast, with an average processing rate of 15 Hz, and reliable, with a detection rate of 100% and a false positives rate of 0.44%. Although it has certain limitations concerning outdoor use, due to its reliability and speed, we will use this method in a slightly modified version also for the real car, see section 6.4.5.

4.2.1 Introduction

Lane detection is a prerequisite for autonomously driving systems and also advanced driver assistant systems (ADAS). Since cameras are cheap, passive distal sensors, vision-based lane detection has been extensively studied and various algorithms on this topic were reported.

The requirements for lane detection algorithms are: speed of processing (at least at a rate of 15 Hz), reliability (a low rate of false-positives) and robustness (against changing illuminations, reflections, and occlusions). For the indoor-scenario of this work however, robustness was less important than a) the ability to detect high curvature curves (since the laboratory street course contained very challenging parts as can be seen from Fig. 4.1(d) and 4.3(a), b) to require little a priori knowledge, and c) to make only few system requirements (according to the hypothesis of this work).

To cope with the typical challenges in "real"-street detection, such as changing illuminations, occlusions, versatile types of roads etc., usually model-based algorithms are applied. For instance, [Dickmanns, 2007; Goldbeck and Huertgen, 1999] used clothoid curves, [Wang et al., 2003; Kim, 2006; Wang et al., 1998; Aly, 2008] used second and third order splines, [Taylor et al., 1996; Liu et al., 2010] made straight line assumptions, and [Southall and Taylor, 2001; McCall and Trivedi, 2004] used a third order polynomial as street model. The constraints imposed by the models increase the robustness of object detection, given that the object, in this case the street, appears as given by the model. If it differs from the expectation however, it might not be detected at all. Since the goal here was to detect lanes of *arbitrary* shapes a model-based approach was undesired. In addition, a street model constitutes predefined knowledge which we also tried to avoid here.

By contrast, feature-based object detection, a bottom-up approach, constructs objects directly based on low-level image features without given model knowledge. Its advantage is the potential to detect arbitrary, also beforehand unknown shaped lanes, but at the same time, due to the few constraints it is prone to detect false-positives, or to merge parts of false-positives into a lane description rendering it useless. Another direct consequence of using only few constraints is that such algorithms are known to be sensitive to occlusions. Examples for feature-based "real"-lane detection are [Bertozzi and Broggi, 1996; Pomerleau, 1995; Broggi and Berte, 1995; Apostoloff and Zelinsky, 2003]. However, all these approaches require the removal of the perspective effect, which necessitates specific a priori knowledge violating the demands here.

The algorithm developed for this work[1] acts basically as a line tracer. As such it can extract each lane marker independently and is feature-based. Furthermore, it makes only three assumptions: (i) during the initialization of the process the sought lane marker is within a known interval at the image bottom, (ii) the marker is a smooth curve of a minimal length, (iii) there is a strong intensity contrast between lane marker and ground. The method comprises three consecutive steps, edge detection, curve construction, and lane construction. Its output is an ordered vector of pixels describing a curve in the image that can contain small gaps and which can easily be changed into a different representation, e.g. it can be approximated by a spline, a polyline, i.e. straight line segments, or else. To filter out false positives a Kalman filter is used which tracks the starting position of the detected marker adding considerable robustness.

In the following Method section 4.2.2, first an overview of the algorithm is given, followed by a detailed description of the individual processing steps. Its performance is presented in the Results section 4.2.3, and finally its advantages and disadvantages are discussed in the Conclusion 4.2.4 of this section.

4.2.2 Methods

4.2.2.1 Overview of the Lane Marker Detection Algorithm

The overall algorithmic flow of the lane marker detection method is depicted in Fig. 4.3, exemplary showing the detection of the right marker. The incoming image 4.3(a) is processed by a conventional edge-detector (labeled “edge detection” in the figure) which returns information about which pixels form edges (the edge image is shown in 4.3(b), white pixels denote edges) and what their orientations are, (the discretized orientation image is shown in Fig. 4.3(c), where same gray values denote same pixel orientations). This processing step is explained section 4.2.2.2 of this chapter.

Edge and orientation information serve as input to the next routine, the “curve segment construction” explained in section 4.2.2.3. It is applied twice, once to trace curve segments running from right to left in the image, see Fig. 4.3(d) and a second time to trace segments from left to right, shown in Fig. 4.3(e).

The “lane construction”, see section 4.2.2.4, merges appropriate curve segments to construct longer curves, the candidate lanes, from which the final detected marker is chosen, as shown in 4.3(f). If no candidate curve is available, the algorithm returns a notification that no lane marker was found.

After an initial successful detection a Kalman filter is used to track the lane’s starting position which allows excluding other potential lanes and thus effectively reduces the number of false positives as will be explained in 4.2.2.5. As further post-processing a detected lane marker can optionally be smoothed, compare section 4.2.2.6.

[1]An implementation of it is published under the GNU General Public License under https://sourceforge.net/projects/linefinder.

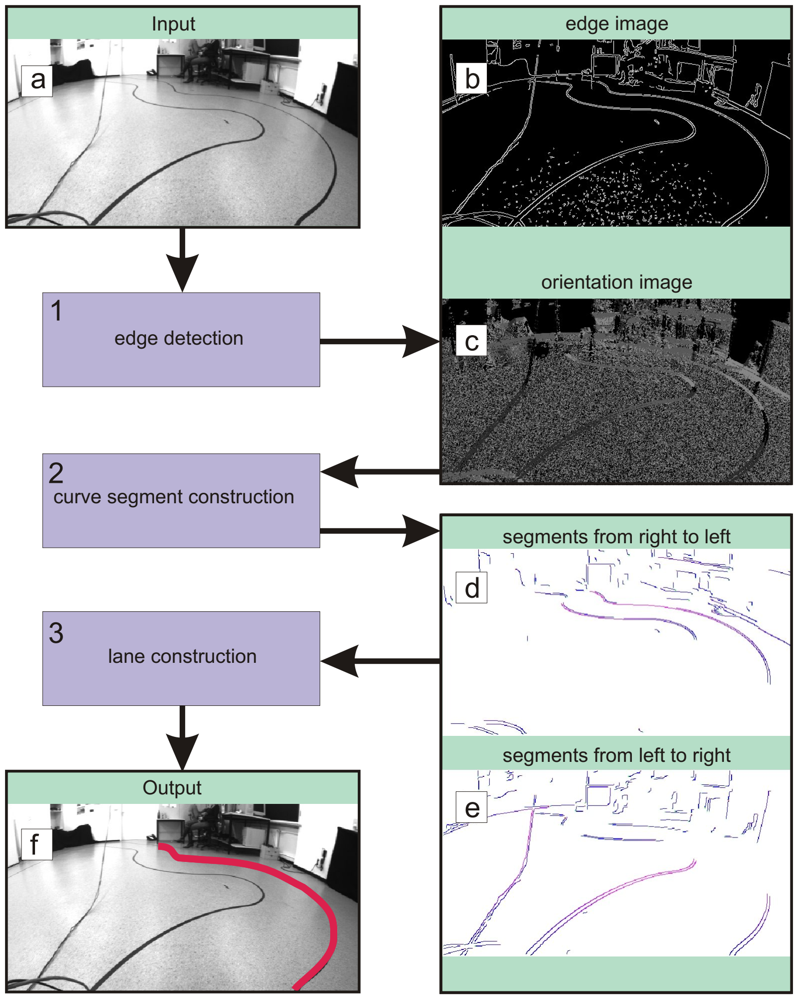

Input
a
edge image
b
1
edge detection
orientation image
c
2
curve segment construction
segments from right to left
d
3
lane construction
segments from left to right
e
Output
f

Figure 4.3: Flow of the lane marker detection algorithm, here exemplary for the right lane boundary. See text for a detailed process description.

4.2.2.2 Edge Detection

The line tracing as realized in this work requires edge- and orientation information of pixels. Edges are image areas with a high intensity contrast and several standard techniques are available for their detection. Here, the well-known Canny operator [Canny, 1986], or B.6 for a short description) is used with a mask size of 9×9, and dynamic thresholds, where the higher threshold is set to 1.3 times the average image intensity, and lower threshold to 0.66 times the average image intensity (compare table C.1 in the appendix C.1). A typical edge image is shown in Fig. 4.3(b).

Pixel orientations are the orientation of the image gradient at a given pixel (see B.4 and B.5). The latter, i.e. the image gradient, is already determined during the processing of the Canny operator for which it employs another filter, in this case the Sobel operator, [Sobel and Feldman, 1973] (see B.3). This gradient can thus be used to calculate the required pixel-orientations which are here discretized, such that each pixel can have an orientation of either 0°, 45°, 90° or 135° which reduces the complexity of the subsequent processing step. An example of computed discrete pixel orientations is shown in Fig. 4.3(c).

4.2.2.3 Curve Segment Construction

Based on the edge and orientation information of each pixel, obtained from the previous step, edge-pixels are linked to form higher-level entities, i.e. curve segments. This is achieved by linking edge pixels with similar orientations together (tracing). To prevent tracing from getting stuck in image parts with complex edge configurations it was found useful to apply the procedure twice, in opposing directions. In one sweep the image is scanned for lines that run from left to right and in a second run from right to left. The tracing procedure, described in the following, results in line segments with a unique starting and end point, and each pixel in-between having exactly one preceding and one succeeding neighbor. This facilitates the fitting of functions to the final curve which describes the lane marker.

The procedure is realized by a master function that iterates over all pixels starting from the lower left of the edge image to the upper right and which evokes a slave function with each detected edge pixel. The slave function does the actual tracing by subsequently testing each neighbor of the edge pixel, from which it is evoked, for being an edge pixel too, while having an appropriate orientation. If so, this neighbor is linked to the initial edge pixel and the slave function is recursively evoked, now by the added neighbor. Depending on the direction that is scanned for, every pixel is defined to have three ordered neigbors, e.g. for scanning from left to right, the first neighbor of a given pixel p is the pixel above p, the second is the pixel to the upper right, and the third the pixel to the right, as shown in Fig. 4.4(a). The neighbors for the scan from right to left are defined analogously and given in Fig. 4.4(b). The slave function checks each neighbor in the given order and labels those that get linked to a curve segment as "read", such that same pixels cannot belong to more than one curve segment. As said, the slave function checks each neighbor (that belongs to

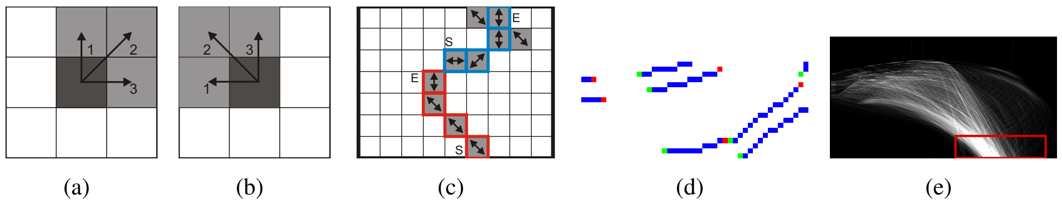


Figure 4.4: (a): First, second and third neigbor of a pixel (dark gray square) for the scan from left to right, and (b) for the scan from right to left. (c) Gray squares denote edge pixels and arrows their orientations. The curve segment construction algorithm leads to the depicted result: The curve segment, (red), going from right to left, is the result from running the algorithm with the ordered neighbors shown in (a). The blue segment used the ordered neighbors in (b). Start- and end-pixels are denoted by 'S' and 'E'. (d) Magnified curve segments from a real image with start-pixels in green and end-pixels in red. During the lane construction routine such segments are linked together. (e) 2000 right lane markers additively plotted. The brighter a region the more markers were found at that spot.The red rectangle denotes the expected lane marker starting interval for the initialization of the process.

an edge) for its orientation. The orientation is initially the orientation of the first edge pixel with which the function is evoked. However, if no neighbor is found fulfilling this criterion the condition is relaxed by also testing for orientations equal to the original one ±45°. The first pixel from which the slave function is evoked is labeled "start-pixel" and the last "end-pixel". After the slave function returns, the master function proceeds to the next edge pixel and the procedure begins anew, until the upper left of the edge image is reached. Pseudo-code for these two functions is given in the appendix C.22 (master function) and C.23 (slave function). An example visualizing the procedure is shown in Fig. 4.4(c), and an example with real data in Fig. 4.4(d).

Note, a feature of the routine is that it cannot "return", i.e. come back to lower parts of the image. This is also depicted in Fig. 4.4(c), where the most right pixel is ignored by the routine. This way it is avoided that the complete boundary of a lane marker is returned, which starts at the image bottom runs around the lane marker and comes back to the image bottom.

4.2.2.4 Lane Construction

This routine takes the ordered curve segments from the previous processing step as argument and links those segments together which are likely to be part of the sought lane marker producing long candidate curves. The procedure includes the interpolation of small gaps between segments which are typically a few pixels wide. If curve segments are linked depends on the size of the gap between them and the difference of their orientations as described in detail below.

The routine exploits that start- and end-pixel of each curve segment are known. It first checks each segment if it starts within an expected region. For each such found "seed" segment it then tests if there is a close-by segment with a similar orientation. A close-by segment is one whose start-pixel is within the specified gap width (40 pixels), measured from the end-pixel of the segment that is currently being processed. The orientations compared are those of the start- and end-pixel of the two segments. If their difference is smaller than a specifiable threshold (we used $\pi/2$, compare table C.1 in the appendix C.1) the segments are linked together and the procedure starts anew, testing segments close to the new end-pixel. If several segments are detected fulfilling the condition of vicinity, the most similar one, concerning the orientation, is chosen and linked. Some typical segments (the orientation is not depicted) are shown in Fig. 4.4(d).

This way only candidate curves are produced that start within an expected region and all others are neglected, which reduces processing time. The expected region for the initialization of the process was chosen to be the interval in which most lane markers fell determined by additively plotting markers from many (2000) images, as shown exemplary for the right marker in 4.4(e). This fixed interval is only used for initializing the routine and is afterwards tracked dynamically.

The output of this lane construction algorithm is a) either exactly one detected line, which is returned as lane marker if longer than a minimal given length, or b) several candidate lanes, of which the longest (during initialization) and later, the one closest to the previously detected lanes) is returned as lane marker, or c) nothing, in which case a warning is issued.

4.2.2.5 Lane Tracking

The constraint for the seed curve to start within a certain interval is important for reducing processing time but most of all for disambiguating seed lane-edges from other edges. As said, this interval (see Fig. 4.4(e)) is only used in the initialization phase of the algorithm. After a lane marker has been detected for the first time, its starting position is dynamically tracked using a standard Kalman filter (KF), [Kalman, 1960].

The KF is a recursive state estimator for linear, possibly stochastic, dynamical systems (parametrized by time, i.e. given in state-space representation), that have access to measurements related to the system's state. The system's dynamics model is given in Eq. 4.1 and the measurement process is modeled by Eq. 4.2.

$$\mathbf{x}_t = \mathbf{A}_t\mathbf{x}_{t-1} + \mathbf{B}_t\mathbf{u}_t + \boldsymbol{\epsilon}_t, \qquad \text{state evolution} \tag{4.1}$$

$$\mathbf{z}_t = \mathbf{H}_t\mathbf{x}_t + \boldsymbol{\nu}_t \qquad \text{measurement model,} \tag{4.2}$$

The state vector at time t is given by $\mathbf{x}_t \in \mathbb{R}^n$, $\mathbf{A}_t$ is an $n \times n$ matrix describing the change of the system over time, $\mathbf{u}_t \in \mathbb{R}^m$ is called the input vector and contains control input given to the system, for example the voltage applied to a motor, and the $n \times m$ matrix $\mathbf{B}_t$ models how the control input affects the system. Finally, $\boldsymbol{\epsilon}_t$ is a Gaussian random vector of length n, describing the process noise, i.e. those properties that cannot, or only with difficulties, be incorporated into the model. It has zero mean and its covariance is denoted by $\mathbf{Q}$, see Eq. 4.3. The measurement $\mathbf{z}_t \in \mathbb{R}^l$ as stated in Eq. 4.2 is related to the true state by the $l \times n$ matrix $\mathbf{H}$ and the multivariate normal random vector

$\boldsymbol{\nu}_t$ of length l, which models the measurement noise with zero mean and covariance $\mathbf{R}_t$, see Eq. 4.4.

$$\boldsymbol{\epsilon}_t \sim N(0, \mathbf{Q}_t) \tag{4.3}$$

$$\boldsymbol{\nu}_t \sim N(0, \mathbf{R}_t) \tag{4.4}$$

To predict an adequate search interval for the starting position of a lane marker, the horizontal image coordinate of the pixel (its x-coordinate) at which the detected lane starts at the bottom of the image is taken as the state to be estimated. The KF will represent this state (just a scalar) as a normal distribution parametrized by its mean and variance and if the variance is prevented from getting too small it can directly be used as search interval. Here, we enforced a minimal variance of 700 pixels (i.e. a standard deviation of 26.5 pixels) which we found to lead to good results.

Since the observed lane and in particular its starting position at the image bottom change very little from frame to frame, $\mathbf{A}_t$ is set to 1, and $\mathbf{B}_t$ to 0. In other words, the very simplified assumption is made that $\mathbf{x}_t$ equals $\mathbf{x}_{t-1}$ plus some noise. Also, since the measurement is already the quantity that is estimated, H_t is set to 1.

To make the filter conservative against believing new measurements that differ much from the expected values the process variance is set to $Q = 0.01$, and the measurement variance to a very large value $R = 4000$. It is not assumed that ϵ nor ν change over time. Finally the filter requires a guess about the initial system state, i.e. an initial mean and variance, which are set such that they correspond to the rectangle width, shown in Fig. 4.4(e).

4.2.2.6 Smoothing

Since no global constraints were used during the lane marker detection, other than the marker's starting position and a minimal length, it can occur that either the lane marker construction routine links unrelated segments together, or that edges are detected which do not belong to the same objects. An example is shown in Fig. 4.5(a), where the detected lane boundary erroneously ends in the skirting board, as can better be seen in the magnification in Fig. 4.5(b). This occurred due to an intensity contrast caused by the shadow of the table, see Fig. 4.5(c), which was detected as edge by the edge finding routine. To improve such cases the Gestalt principle of "good continuation" [Koffka, 1999] is used which tells that continuous lines are more likely to belong to the same object than discontinuous ones. Hence, a filter is applied that checks if a detected lane marker fulfills this criterion, i.e. that the lane is smooth everywhere, and if not simply cuts it at a spotted discontinuity. The latter is characterized by a sudden change in the curve's curvature. Since the curvature of a digital line cannot be computed as for a continuous one, of which a mathematical function description is available, it needs to be approximated with a different procedure. Here, we compute the so-called *K-cosine curvature* [Rosenfeld and Johnston, 1973] for each pixel of the detected marker, which is the cosine of the angle between two vectors. For a digital line these two vectors are determined by the current pixel, for which the curvature is to be calculated and the

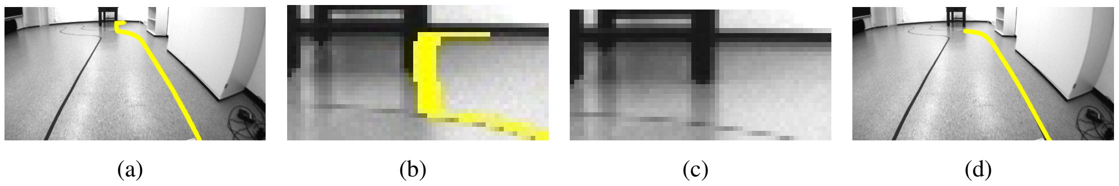


Figure 4.5: (a) An example of a partly erroneous lane marker detection. (b) A magnification of the erroneous part. (c) The magnification shows shadows which caused the error. (d) The result of the smoothing filter. The part after a spotted discontinuity was cut off.

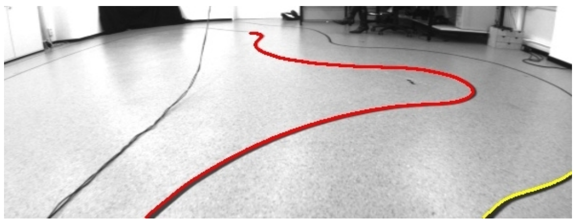


Figure 4.6: The right lane marker is only recognized up to the part where it leaves the field of view of the camera. The remaining marker is not detected.

pixels K (we used K=6) positions before, and after it. Thus,

$$c(l(p)) = cos(\theta) = \frac{\vec{a}\vec{b}}{|\vec{a}||\vec{b}|}, \tag{4.5}$$

with c being the *K-cosine curvature*, $l(p)$ denoting the pixel of the digital line at position p, and $\vec{a}$ and $\vec{b}$ being vectors such that $\vec{a} = (l(p-K), l(p))$ and $\vec{b} = (l(p), l(p+K))$. A discontinuity is detected if a pixel's K-cosine curvature exceeds a certain threshold (we set this value to $\pi/2$, a list of all parameters used in this chapter is given in the appendix C.1). The result of this routine for the given example is shown in Fig. 4.5(d).

4.2.3 Results

The presented method was implemented in C/C++ on a standard laptop with a 2.6 GHz mobile processor and works at 14 Hz on average for finding left and right marker on image patches of size 600×280 pixels, and at 15 Hz for finding only one marker. As desired, high curvature curves with arbitrary shapes can successfully be detected.

Due to the choice of a feature-based object detection approach, the known disadvantages for such procedures apply here too, which means that the algorithm cannot handle large occlusions. In this case it will only detect the part of the lane before the occlusion but not the rest. This can can be seen in Fig. 4.6 where only the beginning of the right lane marker is detected up to where it exceeds the camera's field of view, which is the same effect as that of an occlusion, and the remaining part is not detected.

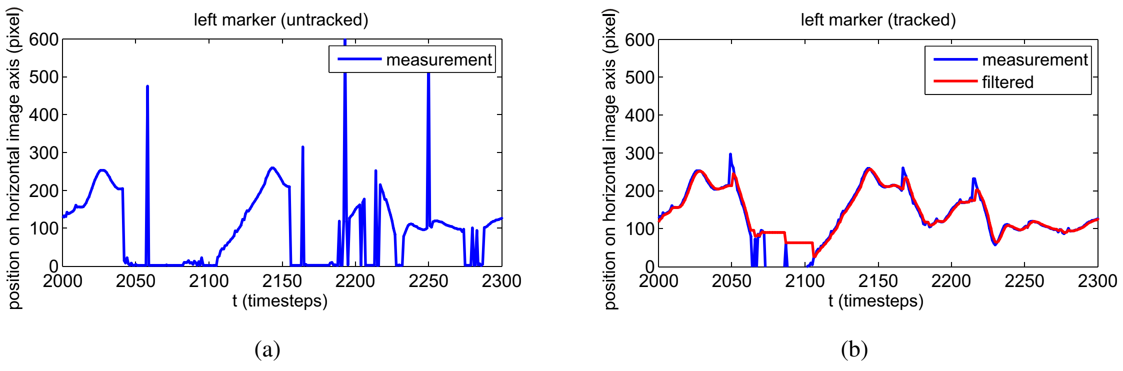


Figure 4.7: The detected left lane marker's first pixel coordinate on the horizontal image axis plotted over 300 images without tracking (a) and with tracking, (b). The original measurement is shown in blue and the red plot is the filtered signal.

To evaluate the performance of the presented method it was applied to find left and right lane markers in 5997 gray-value images of the said size, corresponding to roughly 12 rounds on the depicted track in Fig. 4.1(d). From these images 4905 contained a visible left marker which was never mis-detected, and 5733 frames which contained a visible right marker which was also never mis-detected. Furthermore, in the 1092 images which did not contain a left marker a false one was detected in 48 cases and in the 264 frames that did not contain a right marker a false right marker was found in 5 cases. Thus, the detection rate of the presented algorithm for the used test-set is 100% and the rate of false positives is 0.44%. The adaptive search interval realized by the Kalman filter considerably contributes to these achieved rates, in comparison, without it the rate of false positives is 26.01%. This can be seen from Figs. 4.7(a) and (b) which both show the value of the horizontal image axis coordinate of the starting position of the detected lane marker over 300 frames. The left plot is very irregular reflecting the high amount of false positives. The right plot shows the smooth change in the starting position of the left lane marker. In addition, the filtered signal is plotted in red which even improves the few mis-detections. Thus, tracking the markers adds considerable robustness.

4.2.4 Conclusion

The presented algorithm for detecting arbitrarily shaped curves in an indoor scenario is simple, fast, and robust. It has only few requirements concerning the hardware, using a single gray-value camera. It requires little pre-coded knowledge, using only three assumptions to specify lane markers, and in particular it does not require the removal of the perspective effect and thus the camera to be calibrated and the images to be undistorted! Finally, it successfully detects arbitrarily shaped, high curvature lane markers, and hence it meets all requirements stated in 4.2.1. In addition, since it does not rely on color information it can be used in the night-driving scenario, too, requiring merely some minor additional image preprocessing (smoothing, and adjustment of the Canny filter parameters).

Due to the choice of a feature-based object detection approach, the algorithm cannot handle large occlusions. This can be improved by extending the tracker to not only consider the starting position of the marker but to consider the entire curve instead. This topic will be tackled in chapter 5.

In the next section the extracted sensory information will be used for the realization of the proposed chunking mechanism.

4.3 Chunking

This section realizes chunking and applies it to the stated task (learning proactive driving from a teacher). It takes the extracted task-relevant information from the previous section as input and uses it to tackle the remaining issues (item 2 and 3) from the list given in section 3.3. This is a) providing means for acquiring a database for storing chunks (compiled sequences of actions) indexable by informative patterns of perceptual input (item 2), and b) a mechanism for generalization (item 3).

These mechanisms are realized by means of a database, and by two lazy-learning methods and the combination of a planner and a reactive controller for generalization (compare section 3.3.2). All components are applied in conjunction with the database to the given task and compared to identify and evaluate the best combination, which is then also applied to a night-driving scenario.

The resulting system successfully accomplishes the given task while requiring little predefined knowledge.

4.3.1 Overview

In the following we give an overview on the individual mechanisms and their interplay. Each individual functionality is then described in its own section.

The ability to store and recall observed perception-action pairs is realized by means of a database to which we refer as perception-action repository, PAR, to emphasize its role for linking perception to action, (see section 4.3.2). Fig. 4.8 shows the used system architecture that includes the PAR. It is a loop in which the human driver and the system perceive sensory stimuli from the world and transform them into percepts, labeled ’human sense-act’ and ’system sense’ in the Fig., where the latter involves the described lane marker detection routine, indicated by the label ’sense lane’. The human transforms his or her percepts into driving actions (’act’) which can be observed by the system. It then assigns its own perceived sensory cues to these observed driving actions and stores the resulting sense-act pairs in the PAR. This process is called ’training’, shown by the blue box named ’system training mode’ and is described in section 4.3.2.2. In turn, the system can retrieve appropriate human actions by querying the PAR with its percepts, explained in section

4.3.2.3. These actions can be used for example for autonomous driving, as indicated by the green box named ’system autonomous mode’, where the retrieval step is indicated by the arrows ’query’ and ’retrieve’. However, since the obtained actions are snapshots from previous driving they must be adapted to the currently perceived situation. Thus, the problem of generalization must be tackled, denoted ’prediction of action sequences’, see section 4.3.2.4. Thus, the output of the system in autonomous mode under optimal conditions are sequences of driving actions, i.e.
action plans, a detailed overview on all possible outputs is given in section 4.3.2.4.5. Finally, either the system’s or the human’s actions are transferred to the effectors of the car, closing the perception-action loop.

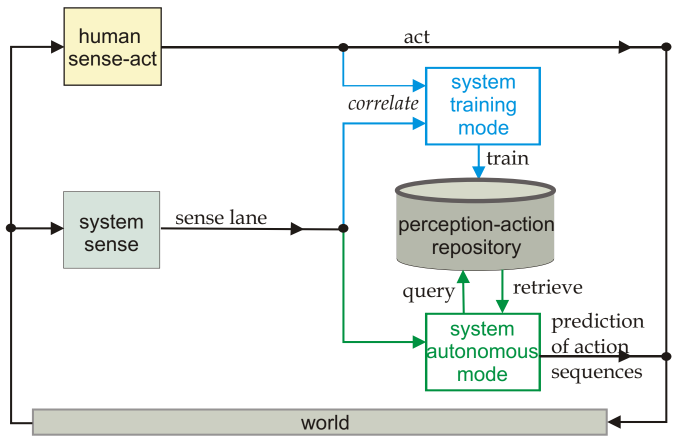


Figure 4.8: System architecture for PAR use.

To realize the prediction, i.e. the generalization (compare also section 3.3.2, we suggest two lazy-learning mechanisms AVG and DIFF (see sections 4.3.2.4.1, and 4.3.2.4.2) which return the desired actions plans. Therefore, the PAR in combination with such a prediction, or generalization method resp., is also referred to as “Planner”. In addition to the generalization based on lazy learning, we use a second mechanism for fine-tuning lateral control which is a reaction-like mechanism, and thus referred to as reactive controller, RC, see section 4.3.2.4.3.

The methods are evaluated open- and closed-loop and it is found that the PAR in combination with AVG is most advantageous in terms of performance and simplicity. Thus, these two, PAR and AVG, are combined with the RC, tested for the resulting predictive capacity, and applied to the night-driving scenario, (see section 4.3.3.6). Although the generalization ability of the presented chunking approach is found to be limited, the overall performance is good as shown in the Results section 4.3.3. Finally, the advantages and disadvantages of this method are discussed in the Conclusion of this Chapter, 4.3.4.

4.3.2 Methods: The Perception-Action Repository

4.3.2.1 Sense-Act Pairs

The sense-act pairs stored in the PAR are designed to reflect the assumption that during chunking while driving humans associate particular street trajectories with subsequent stereotypic driving behaviors. Thus, the sense-part, i.e. the cue- or state-vector, $\mathbf{s}$, is chosen to contain an indication of the observable street ahead, here a condensed description of the extracted right lane marker, $\mathbf{v}_{\text{right}}$, from a current image frame, I_t. Where, t denotes current discrete time measured in sense-act cycles, compare section 4.1. The right lane marker was chosen because it constitutes an indication of the street ahead and because it is visible in the majority of the recorded frames which is often not the case for the left marker. It is obtained by the lane detection method described in section 4.2 which returns it in form of a list of pixels.

The retrieval step requires the comparison of state-vectors, or rather the comparison of the individual entries of the state-vectors with each other. The extracted right lane markers, 2D curves, are expensive to compare pixel-wise and also to store, since they are data intensive. Consequently a condensed curve description is desired. A spline approximation is convenient for a compact curve representation, however it is not suitable for curve comparison (unless special precautions are taken, see section 5.2.2), since the control points representing the approximated curve can be very different for two curves that appear similar to humans, which is shown in Fig. 5.1(a) and (b), (compare also [Blake and Isard, 2000] p. 59 about the problem of curve comparison). Thus, comparing the control points of two splines might be misleading concerning the similarity of the curves described by the splines. Here, we approximate the curve with a polyline, i.e. a set of straight lines, see Fig.

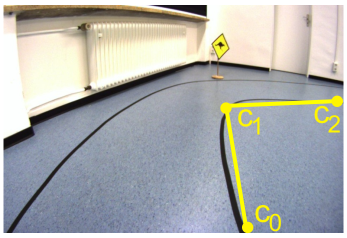


Figure 4.9: The right lane marker described by a polyline with supporting points c_0, c_1, and c_2.

4.9, obtained with the Douglas-Peucker method, [Douglas and Peucker, 1973]. Thus, the curve is represented by the supporting points of the polyline, denoted $c_0, c_1, \ldots,$ (compare Fig. 4.9) as given in Eq. 4.6, where l denotes the length of $\mathbf{v}_{\text{right}}$, i.e. the number of supporting points. Although this is crude and curve information like precise curvature is lost, the supporting points for similar curves are similar too, and thus enable an efficient curve comparison.

$$\mathbf{v}_{\text{right}} = [c_0, c_1, \ldots, c_l] \qquad \text{description of right lane marker.} \tag{4.6}$$

In addition to the visual cues we augmented the system's state-vector to consider the vehicle's past to some extent, assuming that the vehicle's state is not completely independent of that. (E.g. the situation of being exposed to a straight street might be a different one if the vehicle just left a curve, compared to the situation when the vehicle did not leave a curve but was already on a straight stretch of road.) For that we increase the state-vector to also hold a short sequence (of length $m = 20$) of steering actions, $\mathbf{st}_{\text{past}}$, conducted by the human, *prior* to observing I_t (the image frame). Thus, the state-vector is as given in Eq. 4.8. To each such state-vector, the sense-part, we assign an act-part, the chunks which are sequences of future human driving actions, one for steering and one for speed, executed by the human driver *after* observing I_t. We refer to them as $\mathbf{st}_{\text{fut}}$ and $\mathbf{sp}_{\text{fut}}$ for steering and speed. Their lengths are supposed to resemble the number of actions necessary to cover the part of the street observable in I_t. Since we do not know exactly to how many actions this corresponds (for that it would either be necessary to calculate absolute curve lengths, or to have the system recognize when reaching a position that was previously at the limit of its field of view), we use the experimentally determined value based on the approximate length of the curve:

$$n = \lfloor \frac{1}{8} \sum_{i=1}^{l} |c_{i-1} - c_i| \rfloor. \tag{4.7}$$

Thus, a PAR entry , $\mathbf{e}$, – the final sense-act pair – is as given in Eq. 4.9.

$$\begin{aligned} \mathbf{s} &= \{\mathbf{v}_{\text{right}}, \mathbf{st}_{\text{past}}\}, && \text{state-vector} && (4.8)\\ \mathbf{e} &= \{\mathbf{s}, \mathbf{st}_{\text{fut}}, \mathbf{sp}_{\text{fut}}\} && \text{PAR entry = sense-act pair} && (4.9) \end{aligned}$$

The action sequences $\mathbf{st}_{\text{past/fut}}$ and $\mathbf{sp}_{\text{fut}}$ are as given below in Eqs. 4.10, 4.11 4.12,

$$\begin{aligned} \mathbf{st}_{\text{past}} &= [st_{t-1}, st_{t-2}, \ldots, st_{t-m}], && (4.10)\\ \mathbf{st}_{\text{fut}} &= [st_t, st_{t+1}, \ldots, st_{t+n}], && (4.11)\\ \mathbf{sp}_{\text{fut}} &= [sp_t, sp_{t+1}, \ldots, sp_{t+n}], && (4.12) \end{aligned}$$

with *st* and *sp* denoting single steering and speed signal values (actions).

4.3.2.2 Training

During the training phase a new sense-act pair, $\mathbf{e}$ (compare Eq. 4.9), is added to the PAR if it represents new knowledge, i.e. if no entries are already available containing similar information. Precisely an entry is added if: a) the curve description $\mathbf{v}_{\text{right}}$ of the state-vector of the probed sense-act pair has a different length than those curve descriptions of the state-vectors of the sense-act pairs that are in the PAR. In other words if a completely new street trajectory is observed. Or b) if there are such entries, but none of them has a state-vector *sufficiently similar* to the one probed for adding, which is explained in the following: We define similarity between two state-vectors $\mathbf{s}$ and $\mathbf{s}*$ as the individual differences between their components, i.e. the difference between $\mathbf{v}_{\text{right}}$ and $\mathbf{v}^*{}_{\text{right}}$, which we denote as ϵ_v, and the difference between $\mathbf{st}_{\text{past}}$, and $\mathbf{st}^*_{\text{past}}$, denoted ϵ_st. Thus, the similarity between two state-vectors is given by the vector $\boldsymbol{\epsilon}$ in Eq. 4.13.

$$\boldsymbol{\epsilon} = [\epsilon\_\text{v}, \epsilon\_\text{st}] \qquad \text{difference between state-vectors.} \tag{4.13}$$

The difference, ϵ_v is the root of the summed squared differences between the single entries of $\mathbf{v}_{\text{right}}$ and $\mathbf{v}^*{}_{\text{right}}$ as given in Eq. 4.14. In addition the term includes a weight vector $\boldsymbol{\omega}$ which punishes differences between two lane markers close to the image bottom more than differences appearing closer to the horizon, i.e. $\omega[i] \geq \omega[i+1]$ (we used 20, 10, 5, 5 for the first four $\boldsymbol{\omega}$ entries and 1 for all remaining ones).

$$\epsilon\_\text{v} = \sqrt{\sum_{i=0}^{l} \omega[i](\mathbf{v}_{\text{right}}[i] - \mathbf{v}^*_{\text{right}}[i])^2}, \qquad \text{difference between perceptual cues,} \tag{4.14}$$

$$\text{with } |\mathbf{v}| = |\mathbf{v}^*| = l. \tag{4.15}$$

Note, that the difference ϵ_v is only defined for curve descriptions of same lengths. Analogously defined is the difference, between two action sequences, in this case $\mathbf{st}_{\text{past}}$, and $\mathbf{st}^*_{\text{past}}$, thus ϵ_st is as given in Eq. 4.16.

$$\epsilon\_\text{st} = \sqrt{\sum_{i=0}^{m} (\mathbf{st}_{\text{past}}[i] - \mathbf{st}^*_{\text{past}}[i])^2}, \qquad \text{difference between action chunks.} \tag{4.16}$$

Finally, *sufficiently similar* means that both differences are below pre-specified thresholds ($thresh\_\text{v}$ denotes the threshold for the perceptual cues, i.e. the curve description, and $thresh\_\text{st}_{\text{past}}$ the threshold for the steering sequence), i.e. a new sense-act pair is added to the PAR if Eqs. 4.17 and 4.18 are fulfilled:

$$\epsilon\_\text{v} \leq thresh\_\text{v}, \tag{4.17}$$

$$\epsilon\_\text{st} \leq thresh\_\text{st}. \tag{4.18}$$

We set $thresh\_\text{v}$ to 10 and $thresh\_\text{st}$ to 20. If a sufficiently similar entry is already available, the currently tested entry is not discarded, but its attached action sequences are merged (averaged) with those of the most similar entry in the PAR. This is done, since the goal is to capture the human driving behavior and by averaging actions for frequently encountered situations the resulting (synthesized) driving behavior is smoothed and less effected by human 'outlier driving'.

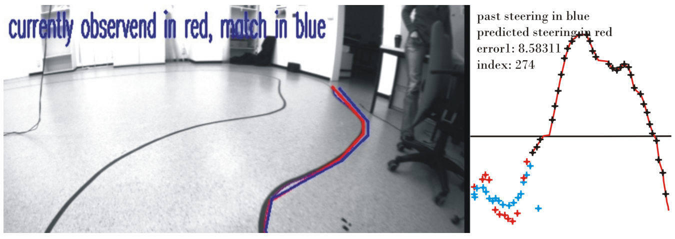
The PAR is complete if a predefined number of entries is reached, or the two cases in which entries are added (not merged) to the PAR as just described do not occur within a user specifiable time. Note, training can be done online while the robot is tele-operated or offline using recorded data.

4.3.2.3 Retrieval

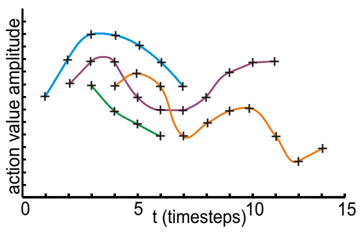
To retrieve information from the PAR it can be queried with a current state-vector which is compared to the state-vectors of all PAR entries which have a curve description of the same length as the curve description in the query-state-vector. (Because similarity between curve descriptions was defined only for descriptions of same lengths, see Eq. 4.14.) The action sequences attached to the most similar PAR entry are returned. The most similar PAR-entry, which we also call the best match, is the entry with the lowest overall differences, i.e. for which the computed differences ϵ_v, and $\epsilon\_\mathrm{st}_{\mathrm{past}}$ are smaller than those computed for all other entries. If several PAR entries lead to the same minimal values for ϵ_v, and $\epsilon\_\mathrm{st}_{\mathrm{past}}$ the first of these entries is selected as best match. Thus, the state-vector comparison for the retrieval is similar to a pattern matching process.

The return parameters of a query are either: 1a) the error vector $\boldsymbol{\epsilon}$ containing the differences ϵ_v, and $\epsilon\_\mathrm{st}_{\mathrm{past}}$ between the best match and the query, and 1b) the action sequences $\mathbf{st}_{\mathrm{fut}}$ and $\mathbf{sp}_{\mathrm{fut}}$ assigned to the best match, or: 2) an indication that no match could be retrieved. The latter occurs either when there was no entry that the query could be compared to, or the best found match was unacceptably bad, i.e. the assigned differences exceeded predefined thresholds, where $threshR\_\mathrm{v}$ denotes the threshold for the perceptual cues (we used 200), and $threshR\_\mathrm{st}_{\mathrm{past}}$ the threshold for the past steering sequence (we used 100).

The retrieval process is visualized in Fig. 4.10(a).


currently observend in red, match in blue
past steering in blue
predicted steering in red
error1: 8.58311
index: 274

(a)



(b)

Figure 4.10: (a)A screenshot of the PAR in retrieval mode. Left: The detected polygonized right lane marker is shown in red and the lane marker of the best found match in blue. Right: The solid red line shows the retrieved steering chunk where black crosses denote discrete actions. The vertical axis indicates the magnitude of the values and the horizontal axis which is placed at 0 at the vertical axis denotes time. . Positive steering values correspond to right steering and vice versa. Thus, the obtained chunk predicts a right turn. The dotted blue line denotes past steering $\mathbf{st}_{\mathrm{past}}$ of the current state-vector and the dotted line in red $\mathbf{st}_{\mathrm{past}}$ of the retrieved best match. (b) Shows the overlay of chunks (each colored line denotes a chunk, and dots indicate discrete actions) over time if the PAR is queried every timestep (here only shown for the first 4 timesteps).

Note, the error vector $\boldsymbol{\epsilon}$ indicates how well a queried state is represented in the PAR, i.e. how familiar an encountered situation is to the system. If frequently chunks are returned that have large assigned errors the resulting control might not be well suited for the currently driven track. Thus, the returned errors inform about possibly dangerous situations.

4.3.2.4 Generalization and Prediction

In autonomous mode the system queries the PAR each timestep with the current state-vector and receives the corresponding action chunks as visualized in Fig. 4.10(b), (unless no entry in the PAR was accepted as a good enough match in which case no chunk is returned). The retrieved chunks are adequate for the system's current state to a varying degree, depending on how well the current state-vector is reflected in the PAR. The prediction process describes how the obtained action chunks are used to obtain action plans suitable for the current situation, i.e. it represents a way to generalize from known to unknown situations. In the following, two mechanisms are introduced. One is based on a difference equation and thus referred to as DIFF and the other is based on simple averaging and thus called AVG.

Furthermore, steering control is considered to be a two-level process [Donges, 1978] using short-term and look-ahead information, where we use the word "short-term" to denote relevant visual information that is temporally and spatially close to the vehicle and "look-ahead" to denote visual information that is relevant in the future, i.e. further away from the vehicle. We therefore introduce a simple reaction-like mechanism, RC, which is combined with the predicted steering sequence from the Planner, i.e. the PAR in combination with AVG or DIFF. This allows us to modify an obtained steering plan on-the-fly to fine-tune it to the current situation. This can be necessary if the encountered situation is not well represented in the PAR. In other words it presents an additional mechanism for generalization (compare also section 3.3.2).

4.3.2.4.1 DIFF The DIFF method considers the appropriateness of retrieved action chunks for a current situation. As said, this is indicated by the assigned error vector $\boldsymbol{\epsilon}$ (compare paragraph 4.3.2.3). Accordingly, two chunks can be compared by means of their error vectors, and a chunk with error vector $\boldsymbol{\epsilon}$ is better suited than another chunk with error vector $\boldsymbol{\epsilon}^*$ if $\boldsymbol{\epsilon} \leq \boldsymbol{\epsilon}^*$, that is each component is of $\boldsymbol{\epsilon}$ is smaller or equal than the according component in $\boldsymbol{\epsilon}^*$.

DIFF works off the actions of a single chunk until a better one is retrieved and creates smooth transitions between the two. This is depicted in Fig. 4.11(a). To acknowledge that a good chunk obtained a few timesteps before becomes less and less reliable over time, we increase its error vector entries at each timestep, i.e. we devalue a current chunk over time. The action signal computation with DIFF is achieved by predicting the next action signal a_{t+1} based on the last one plus an increment, see Eq. 4.19. Where an action signal represents a current steering or a speed command, i.e. either st_t or sp_t. The increment is chosen such that the resulting signal value is between the last one and the one(s) given by the current chunk, which are denoted $\tilde{a}_t$ and exemplary shown in Fig.

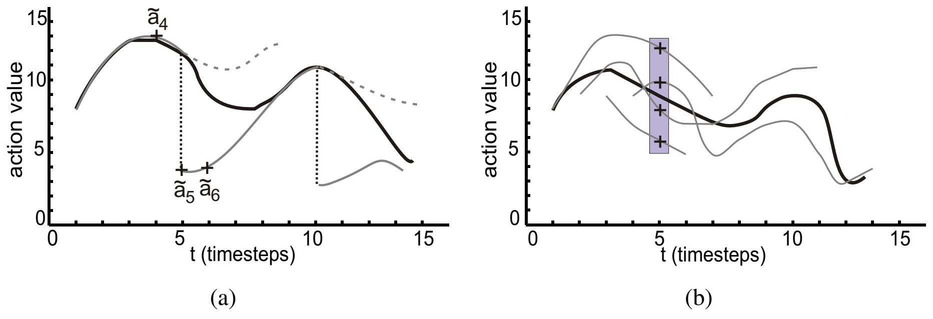


Figure 4.11: (a) and (b) visualize schematically the effect of DIFF (a) and AVG (b). Gray lines denote retrieved chunks and black crosses indicate discrete action values. (a) DIFF works off a single retrieved chunk until a better one is found, indicated by the vertical lines. The actions on a current chunk are denoted $\tilde{a}_t$, as exemplary shown. As depicted, there might be no smooth transition between consecutive chunks and the purpose of the proposed difference equation is to smoothly join such pieces together. The resulting signal of DIFF is drawn in black. (b) AVG averages over the action values of the last k retrieved chunks (gray lines) belonging to the same timestep, which is indicated as gray rectangle. Results on real data for both methods are shown in Fig. 4.14.

4.11(a). The increment is the discounted sum of n (specifiable by the user) differences between the last action value and the corresponding value of $\tilde{a}_{t+i}$, where i is an index from 0 to $n-1$, as given in Eq. 4.20.

$$a_{t+1} = a_t + \Delta_{a_t} \tag{4.19}$$

$$\Delta_{a_t} = \sum_{i=0}^{n-1} \alpha_i \frac{\tilde{a}_{t+i\tau} - a_t}{(1 + \frac{a_t}{a_{max}})G}, \tag{4.20}$$

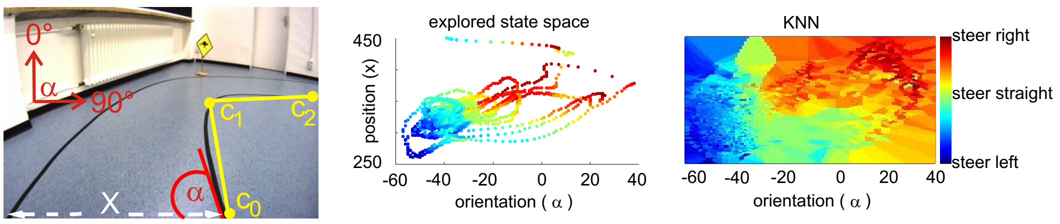
where $\alpha_i = e^{-(i^2/\sigma^2)}$ is a decay term which devalues the influence of current chunk entries the further they reach in the future and σ is a specifiable constant. Optionally the variable τ can be set as a constant determining the sampling frequency on the current chunk. It influences how fast to move from one chunk to another. If a low value is chosen, the resulting control sequence lingers longer in the vicinity of a discarded chunk before reaching the values of the new one and vice versa. The denominator serves to decelerate the growth of the computed action values if they are already close to known limits learned from training data. It can be tuned by the constant G. We used the following parameter configuration: $\sigma = 4$, $G = 10$; $n = 10$; $\tau = 1$.

4.3.2.4.2 AVG The second method, AVG, in contrast to DIFF, keeps the latest k retrieved chunks for steering and acceleration in memory, and simply averages over action values belonging to the same time step. Assuming that these values are contained in a buffer $\mathbf{g}_{\text{buf}}$, indicated by the gray box in Fig. 4.11(b), a single predicted action is computed as given in Eq. 4.21.

$$a_t = \frac{1}{|\mathbf{g}_{\mathrm{buf}}|} \sum_{i=0}^{|\mathbf{g}_{\mathrm{buf}}|-1} \mathbf{g}_{\mathrm{buf}}[i]. \tag{4.21}$$

Thus, every action value in the resulting prediction is a linear interpolation between the examples learned before. This is similar to the k-nearest neighbor algorithm which uses k closest training examples to compute a value for the variable of interest [Hart, 1967].

4.3.2.4.3 The Reactive Controller The purpose of the RC is to correct predicted steering plans on-the-fly if necessary. Similar to the PAR, RC maps visual cues obtained from the extracted right lane marker to human actions. However, only cues are used that describe the immediate future of the system concerning steering, i.e. short-term information. The RC maps only one control action to the cues instead of sequences (chunks) and is therefore not able to return action sequences. Precisely it is designed as follows: We define the short-term information of the robot-car by the tangent constructed from the beginning of the extracted street boundary and describe it by the angle α between tangent and horizontal border of the image and its starting position x on the horizontal axis at the bottom line of the image, see Fig. 4.12(a). To acquire the supervisor's policy we assign human actions (from the training set) to the state space spanned by the two parameters, see 4.12(b) . To generalize to unknown situations, i.e. to fill the empty spaces, we use the k-nearest neighbor algorithm (other interpolation methods can be used, too), the result is shown in 4.12(c). Note that this simple approach results in a very fast controller (requiring only the time necessary for looking up a value in a matrix) that can follow arbitrary curves. In fact, the RC can be seen as a reduced PAR, that only considers the beginning of a perceived lane marker, i.e. its position and direction and the immediate necessary steering action, similar to donkey and carrot on a stick navigation. It is not applicable to the generation of speed commands, since speed generation is much dependent on distal visual information, which can easily be shown, see C.3.



(a) (b) (c)

Figure 4.12: (a) X and α denote the short-term information used for the RC, where α is measured in degrees as indicated. (b) The acquired steering policy from the supervisor. (c) The interpolated policy using nearest neighbor.

4.3.2.4.4 Combination of Planner and Reactive Controller Correction of the predicted steering plan can be necessary if an encountered situation, defined by the state-vector, is not well represented within the PAR, as indicated by the error vector $\boldsymbol{\epsilon}$ returned from the PAR retrieval, compare section 4.3.2.3. Thus, combining RC- and Planner-output should depend on this measure, such that in case of an unfamiliar situation, i.e. high entries in the error vector, more emphasis is put on the RC output and vice versa. This is formalized in Eq. 4.22, where ω_ϵ indicates a weight derived from $\boldsymbol{\epsilon}$, such that high error vector entries result in a high value for ω_ϵ and the other way round as given in Eq. 4.23.

$$st_{t+1} = \omega_\epsilon RC + (1 - \omega_c) planner, \tag{4.22}$$

$$\omega_\epsilon = 0.5(\epsilon\_\mathrm{v}/threshR\_\mathrm{v}) + (\epsilon\_\mathrm{st}/threshR\_\mathrm{st}_{\mathrm{past}}), \tag{4.23}$$

where $\epsilon\_\mathrm{st} \leq threshR\_\mathrm{st}_{\mathrm{past}}$ and $\epsilon\_\mathrm{v} \leq threshR\_\mathrm{v}$ as explained in section 4.3.2.3

4.3.2.4.5 Algorithmic Flow The algorithmic flow resulting from the combination of the described modules is summarized in Fig. 4.13. The detected lane marker is fed into the Planner (as part of the explained state-vector) and the RC. The latter generates only a single steering action and the Planner, given that the PAR contained an adequate match,returns an action plan for steer and speed, as indicated by the arrow labeled 'match'. The steering signals are then combined with the output from the RC. The system fails if the lane marker is (repeatedly) mis-detected, see the arrow labeled 'wrong detection'. If no lane marker could be provided, or nothing could be returned from the Planner, the system falls back on the latest action plans generated earlier, see arrows labeled 'no detection' and 'no match'. If the plans are worked off the system cannot produce any actions, resulting in a 'failure'. Otherwise the remaining plans are returned, indicated by the box 'return remaining plans steer/speed'. A special case occurs if a lane could be detected but no match could be retrieved from the PAR. Then the algorithmic flow follows the green arrows in the figure. Because then the RC can be used to generate a single steering action and if a previous plan is available the remaining steering plan and the RC action can be combined as before, shown by the dotted green line, and the remaining speed plan is simply returned. If no plan is available, no speed control can be generated, however single actions generated by the RC can be used for steer control.

Thus, the generated plans and the RC add robustness to the system by enabling it to generate actions, although no sensory input could be provided (the plans), or if sensory input is provided but unfamiliar to the system (RC).

4.3.3 Results

To test the performance of the PAR and the algorithms DIFF, AVG and RC data is produced from eight laps of human driving on the track in the lab, see Fig. 4.1(d) (always in the same direction). The PAR is trained with the data from five laps and the remaining data is used for testing.

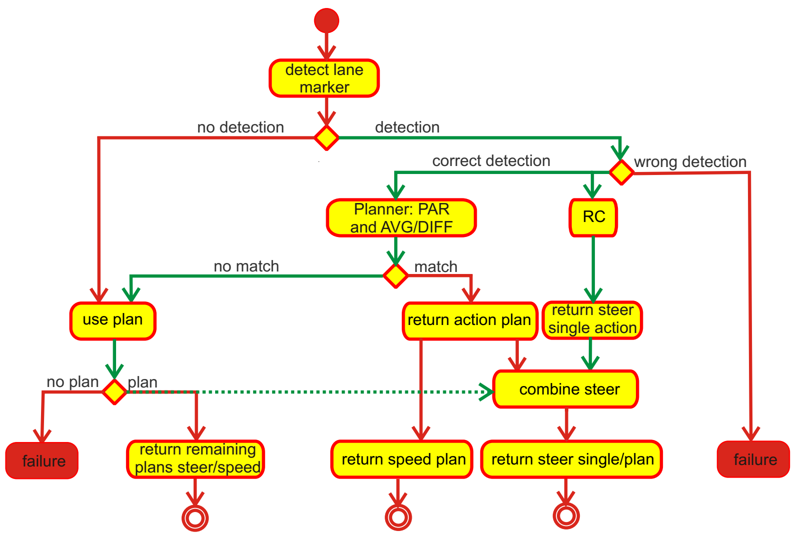


Figure 4.13: Algorithmic flow of the Robot Setup, akin to an UML (Unified Modeling Language) activity diagram.

4.3.3.1 Open-Loop Performance on Known Track

The methods (PAR with AVG, PAR with DIFF, and RC) are applied to the recorded training data and compared to the human signal. The results are plotted in Fig. 4.14(a-e) (to keep the plot clear only one lap of driving is shown) where it can be seen that all three methods capture the human behavior, where AVG and DIFF give smooth output and RC is jerky. Concerning AVG, best results were obtained for speed prediction when averaging over the last $k = 20$ buffer entries and for steering the last $k = 10$. Concerning DIFF, best results were obtained with the following set of parameters: $\tau = 1$, $\sigma = 4$, $G = 10$.

Additionally, we plot the root of the RSS (residual sum of squares) between algorithmic output and human signal. These errors and confidence intervals (95%) are plotted in Fig. 4.16(a) for steer and 4.16(b) for speed. It can be seen that there is little difference between AVG and DIFF. The larger error for RC compared to AVG and DIFF can be explained by its jerkiness. It is also observable that the error for speed prediction on average is higher than for steer which is understandable because there is more variance in the human speed data than in the steering data. This can be seen for example in the speed plot in Fig. 4.14(b) or (c) between timesteps 300 and 500 on the abscissa. The depicted speed signal in this interval can be considered to be constant, however, the small deviations between human and synthesized signal accrue to a relatively large error.

4.3.3.2 Closed-Loop Performance on Known Track

As the previous test was a quantitative comparison, it is necessarily offline, and does not inform if the system behavior would also be acceptable if the controllers were used in a closed-loop setup,

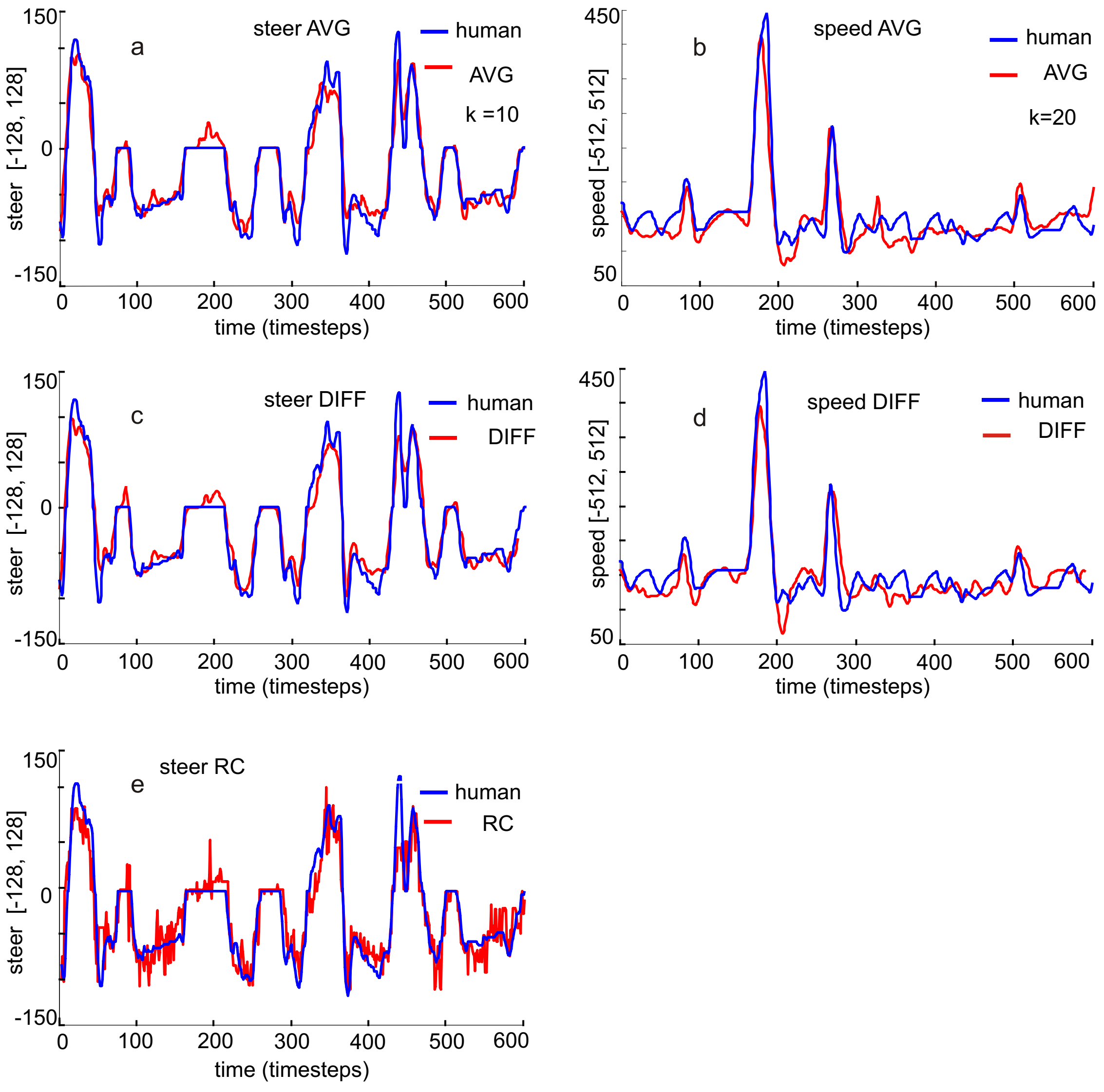


Figure 4.14: Performance of methods on predicting st_{t+1} and sp_{t+1} (not RC). Shown is one lap of driving. (a) and (b): result of AVG, ”k” is the amount of entries in the buffer over which was averaged. (c) and (d): result of DIFF. (e): result of RC for steer control.

i.e. when the generated action of the controller indeed affects its future sensory input. Therefore, we let the robot run on the track in autonomous mode.

We find that with all three controllers it can follow the road well. With AVG and DIFF it follows the track smoothly, and since there is no significant difference between the performance of both methods but DIFF is more complicated to implement and with more parameters to tune we do not consider it in the following anymore but focus on AVG.

In contrast to the smooth behavior of the two Planners, the jerkier signal of the RC also leads to a jerkier lateral behavior. However, due to the inertia of the robot it is less strongly visible than what could be expected from the plotted signal. (A video of the RC performance is on the enclosed cd, "RobotLearnedDriving.avi" or available online under www.markelic.de/-professional/Documents/)

4.3.3.3 Open-Loop Performance of Combination of Planner and RC on Unknown Track

The RC should correct steering output from the Planner if the Planner informs that it is unfamiliar with the encountered situation. Thus, in case of an unfamiliar street environment that is not represented in the PAR, the robot should still be able to issue appropriate steering signals, albeit, without the ability to plan ahead.

We trained the robot in one direction, and since the set up track is circular the robot was almost exclusively exposed to turns in the same direction, in this case to the right, thus, when turned around it is facing turns to the left, which are (almost) not represented in its PAR.

For an open-loop evaluation we let the human drive this unknown track and at the same time record the suggested steering actions of RC, Planner, and the combined signal. One would expect the latter to capture the human signal better than RC or Planner output alone. We show an excerpt of the steering signal of this drive in Fig. 4.15, where at around timestep 100 on the abscissa this behavior can be well observed. The negative human steering value indicates a steep (left) curve, which is not well known by the robot. The amplitude of the signal is important as it describes how much is turned. Over- or understeering without correction leads the robot off the track. It can be seen that the suggested signal from the Planner indicates less left steering, since it does not know what to do in this situation. The RC signal captures the amplitude of the human steering signal better. In this unfamiliar situation the combined output is more determined by the RC signal, therefore it also captures the human behavior better - however, it is also jerkier. In less critical situations the combined signal is smooth, since it is more determined by the Planner. Thus the combination of the RC and Planner has the desired effect.

4.3.3.4 Closed-Loop Performance of Combination of Planner and RC on Unknown Track

To test the closed-loop performance we let the robot drive on the unknown track using a) only the Planner b) only RC, and c) only the combined signals from Planner and RC for steering. (Speed is produced by the Planner, and when this fails a default speed is used.) With the Planner the robot

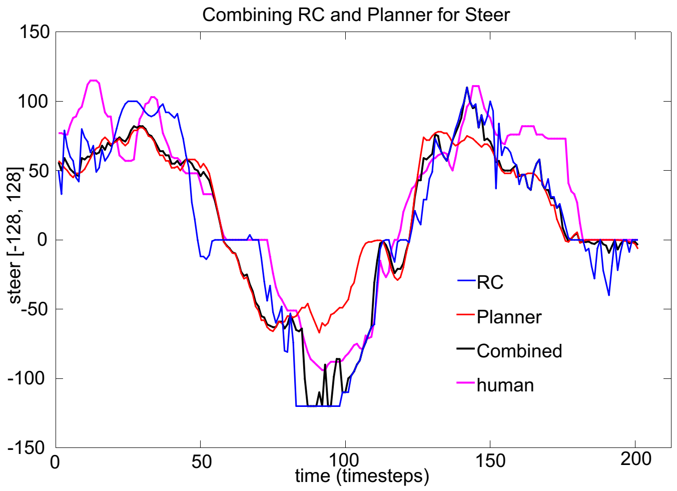


Figure 4.15: Comparison of the combined signal to RC, Planner, and human output, where the robot was driven by the human. At around $t = 100$ it can be seen how the combined signal is better than the Planner output by being drawn closer to the RC signal.

drives smoothly but looses the track in one high curvature turn (compare Fig. 4.1 (d)) due to the explained reason that this situation is not well represented in the PAR. Using the RC it is able to stay on track as expected, however, the behavior is not as smooth. Finally, when using the combined signal it drives smoothly on the known parts, which constitute the majority of the encountered situations, and in addition it manages to stay on the track even during the described difficult turn. This confirms the results from the open-loop test.

4.3.3.5 Predictive Performance

To evaluate the performance of the system concerning sequence prediction, we again test quantitatively and qualitatively.

For the quantitative evaluation we apply the Planner to the training data to generate an action a few timesteps ($t = 0, 10, 20, 30$) ahead, which we then compare to the signal elicited by the human at that timestep. We sum the difference over the entire training data and plot it in Fig. 4.16(d). We also included RC[2] in this plot, mainly for comparison. This result is shown in Fig. 4.16(c). It can be seen that RC's predictive capacity is very poor - as expected, and that AVG's predictive capacity is high in comparison, but the error increases with the number of timesteps to be predicted ahead. This indicates that the actions in the sequence generated by AVG are more precise in the beginning and less reliable with longer predictions, just as expected.

For the qualitative test, we construct an experiment in which we let the robot be controlled first by several humans with differing robot-driving expertise and then by the Planner and RC. Shortly

[2]Since the RC cannot predict sequences we had to "trick" here. To predict the action for $t = 10$, we constructed the RC by mapping $(\alpha_t, x_t) \mapsto st_{t+10}$. We proceeded analogously for $t = 20$ and $t = 30$.

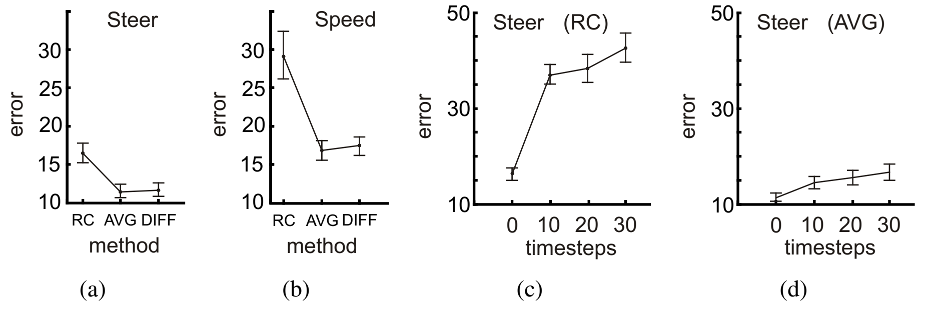
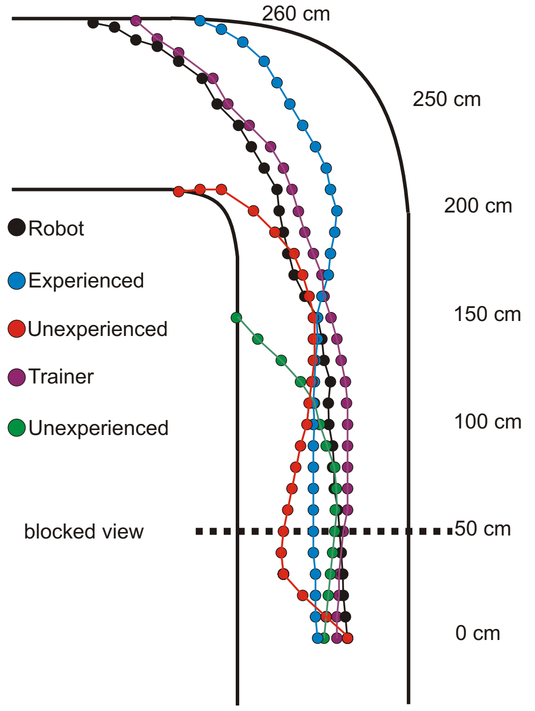


Figure 4.16: (a) Comparing the performance of AVG, DIFF and RC for open-loop steering generation. The plotted error is the root of the summed squared difference between the human action signal and the signal generated by each method. (b) The same for speed generation. (c) The quality of steer predictions of RC for t, t+10..t+30 timesteps ahead. (d) The same for PAR with AVG. As expected, the error for AVG is much lower than for RC indicating the capacity of AVG for action prediction.

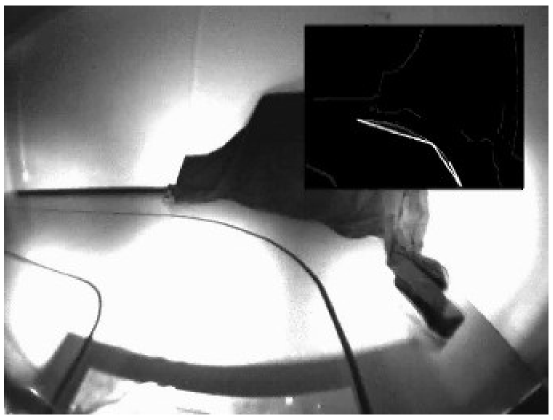
before a turn we abruptly block the visual input and the drivers are supposed to follow the hidden track as long as possible. The experiment ends, when the robot leaves the track. Since the view is blocked before a turn, a real change in steering is necessary, requiring the driver to to plan his/or her actions. By measuring and comparing the produced driving trajectories of this “blind drive”. it is possible to see how humans behave and the artificial control in comparison. We only measure the steering signal, since we need to set speed to a constant value. This is necessary because the human drivers tend to stop the robot when the visual feedback suddenly vanishes. We conduct this experiment with four drivers: two are not trained in driving the robot, one intermediate driver, and the expert, who also generated the training data for the robot. The result is shown in Fig. 4.17(a). It can be seen that the robot does perform the turn, which means that it successfully uses its generated plan and executes it. Furthermore, it can be seen that this trajectory is similarly to the one generated by the trainer, who also is able to perform the turn. It also shows that the less well trained humans lose the track quicker than the robot and the more experienced drivers.

4.3.3.6 Night Driving

The Planner in combination with RC were applied to the night driving scenario, described in section 4.1. Since the described lane detection method is based only on edge information, not color or else, it can almost directly be used, requiring only minor modifications regarding the image preprocessing, i.e. some additional smoothing and adjustment of the Canny parameters (we used 100 and 350). Still, the night situation is more challenging due to strong shadows, high illumination intensity contrasts, and also reflections. Therefore, with the proposed method, lanes cannot be detected with the same look-ahead and reliability as during day driving. As a consequence, the system sometimes faces many unfamiliar situations in a row, such that a predicted action plan

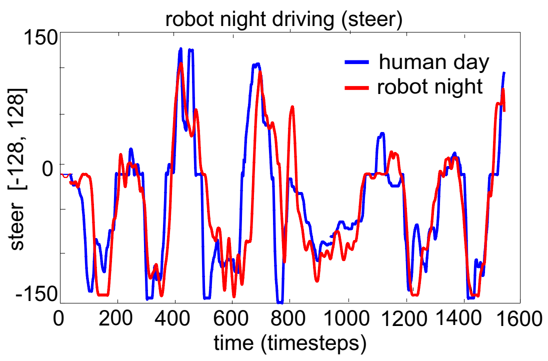


(a)

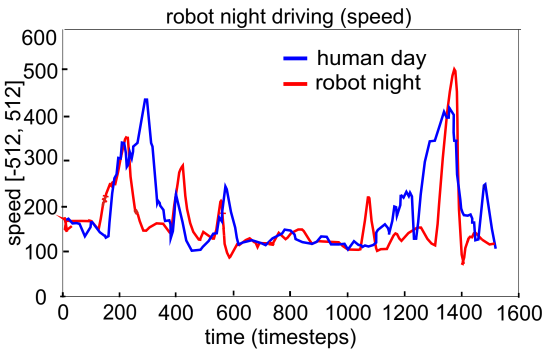


(b)

Figure 4.17: (a) Top view on part of the track. Shown is the driven trajectory of the drivers and the robot, sampled every 10 centimeters. The horizontal line denotes where the view was blocked. (b) The robot's view during night driving.



(a)



(b)

Figure 4.18: Robot performance during night-driving compared to human day-driving. (a) Steer, (b) Speed.

is completely worked off and no action for speed can be generated anymore. (This situation corresponds to the green arrow labeled “no match” in Fig. 4.13 and the subsequent path labeled “no plan”.) Still, the system is able to follow the track in these cases due to the RC and a used default speed.

To compare human and system output the human *day* driving signals are plotted together with the system *night* signals, as shown in Fig. 4.18(a) and (b), excluding the special cases described above. 4.18. It can be seen that the signals are highly similar, for steer more than for speed. It must be noted that the shutter time of the cameras during night driving is shorter since the aperture is almost fully open and as a consequence more images are recorded during the night-scenario then for the same time in the day-scenario. To compare the two corresponding signals (steer-day and steer-night, and speed-day and speed-night) the day signals plotted are interpolated and sampled such that the amount of day driving actions corresponds to the amount of night driving actions.

4.3.4 Conclusion

The goal of this chapter was to a) computationally realize the proposed chunking mechanism with respect to the task of learning proactive driving from a human supervisor, in the indoor-scenario, and b) to evaluate its performance according to the given task.

Chunking was realized by a database-like structure, the PAR, which was embedded in an architecture working at $\geq$ 15 Hz. The problem of generalization was tackled with a lazy learning approach placed on top of the PAR. In addition, for steering control a reaction-like mechanism (RC) was used to correct steering output on-the-fly. It was shown that the system is able to capture the human driving behavior well concerning steer and speed control, thus it successfully accomplished the learning task. Furthermore, the system was shown to have a predictive capacity by successfully producing action plans adequate for future driving. These can be used to cope with short lacks of missing sensory input which might occur due to technical problems. Thus, the ability to cope with such outages makes a system more robust. In addition, if not dependent on permanent sensor processing, capacities are freed which can be used for other tasks.

The experienced main disadvantage is that the method as implemented does not generalize very well, i.e. only situations can be handled that are known to the system to some degree, at least concerning speed.

The main advantage of this method, which sets it apart from many traditional approaches to vehicle control, is that it can generate action *plans* with very little a priori knowledge, in particular no physical models are necessary to obtain action sequence predictions. It is also able to generate speed signals without the necessity to undo the perspective effect. Concerning the biological aspect, this shows that fast action plan production for movement generation during driving does not necessarily require a detailed explicit 3D world model. Concerning practical applications it makes this approach easy to implement and to use. Since this method makes so little requirements it is very general, not limited to driving, but possibly applicable to other skill-learning tasks too.

Chapter 5

From Action to Perception

”We must perceive in order to move, but we must also move in order to perceive“
James Gibson

5.1 Introduction

A common approach in robotics, and also pursued so far in this work, is to treat perception and action as two independent processes[1]. The goal of perception is considered to extract information from the sensory input and to pass this information to the action generating process. Thus, perception is regarded to be *passive* and to *precede* the generation of actions. However, alternative views emphasize the interdependence and cyclic structure of both processes. Perception guides action but in turn action influences and improves perception, hence the term *perception-action cycle*, [Greenwald, 1970; Gibson, 1979; Prinz, 1997; O’Regan and Noë, 2001; Hommel et al., 2001; Jeannerod, 2006; Barsalou, 2009]. According to this view perception is an *active* process comprising *anticipation* and *expectation* [Aloimonos et al., 1987; Bajcsy, 1988; Ballard, 1991]. Anticipation is considered to be the result of having previously learned the perceptual consequence of one’s own action, which corresponds to learning a forward model as described in section 3.2.1. As discussed there, forward models are considered to be the basis for various mechanisms fundamental for intelligent behavior. For example, they are believed to serve as internal representations which again are needed for the ability to reason and plan. They are considered to play a key-role in imitation learning and intention understanding [Meltzoff, 1988; Demiris and Meltzoff, 2008]. If forward models can be learned from experience, as suggested, and indeed lead to all the said benefits, they

[1]This reflects a trend in behavioral sciences which for a long time interpreted perception and action from an information theoretic stance as rather independent and primarily top-down processes [Posner, 1978; Marr, 1982; Massaro, 1990].

are very interesting for commercial use, too. A system that can improve its own perception and thus increase its reliability, that can learn to plan by itself without having to be preprogrammed requires less design effort and therefore less development costs.

Learning forward models has been investigated in the robotics domain, e.g. in Hoffmann [2007]; Hoffmann and Möller [2004]; Koch and Hoffmann [2000]; Demiris and Meltzoff [2008] however, these studies focused on the emergence of cognitive processes.

Here, we investigate, if and how forward models can improve the system's performance for the stated task. Learning a forward model in our case means to enable the system to predict the appearance of the detected lane marker after the execution of a certain motor command. First, we consider the RC (see section 4.3.2.4.3), which is the simpler case, hence it operates on a straight line description of the lane marker in section 5.2.1, and after that we extend this idea to the more complicated case of the planner which considers the marker's complete shape in section 5.2.2. The work is concluded by discussing the experiences gained with this approach in section 5.4.1.

5.2 Methods

5.2.1 A Forward Model for the Reactive Controller

Like all reactive systems the Reactive Controller used here, works without internal representation and depends on external stimuli. This strength (fast, and frequent simple moment-to-moment control) is simultaneously a drawback. A reactive system is not robust to missing or erroneous sensory information and it is not able to plan ahead. According to the above, these issues should improve when employing a forward model. Here, we describe the learning of such a model for the RC and how we test it with respect to the two issues a) handling of missing sensory input and b) planning capacity.

The RC works on a linear approximation of the observed lane marker which is described by an angle and a position, α and x, as shown in Fig. 4.12. Furthermore, since the RC predicts steering only, in this section we work on data with constant speed. Consequently, the goal is to identify a system which realizes the mapping given in Eq. 5.1:

$$(\alpha_t, x_t, st_t) \mapsto (\dot{\alpha}_t, \dot{x}_t), \tag{5.1}$$

based on given observations in the form of Eq. 5.2,

$$(\alpha_t, x_t, st_t, \alpha_{t+1}, x_{t+1}), \tag{5.2}$$

which are available from the supervised driving. To gain better control, we split Eq. 5.1 into two simpler ones given in Eqs. 5.35.4

$$(x_t, \alpha_t, st_t) \mapsto (\dot{x}_t). \tag{5.3}$$

$$(x_t, \alpha_t, st_t) \mapsto (\dot{\alpha}_t) \tag{5.4}$$

We use data from supervised driving with fixed speed, from which 3500 entries serve as training data and 800 for testing. We test four different system identification approaches using partly the C++ library "fann" [Nissen, 2005] and partly Matlab:

- A multilayer perceptron (MLP).
- A radial basis function network (RBF).
- A probabilistic RBF.
- A simple interpolation using nearest neighbor realized with a 3D look-up-table (LUT) analogous to the training of the RC itself as described in section 4.3.2.4.3.

A short overview of these methods is given in the next three paragraphs 5.2.1.1, 5.2.1.2, and 5.2.1.3.

We test the learned models offline regarding their predictive capacities and show the results in section 5.3.1. Furthermore, we implement the model based on the MLP on the robot and test it online regarding the issue of missing sensory input.

5.2.1.1 Multilayer Perceptron

A multilayer perceptron, MLP, is an artificial neural feed-forward network with information flow from the input- to the output layer, compare also appendix B.1. Here, we realize it with the fann library. The net for predicting $\dot{\alpha}$ has two hidden layers with 200 and 80 neurons, the net for predicting $\dot{x}$ has three hidden layers with 100 neurons each. Both nets are trained with the resilient-propagation algorithm (RPROP, [Riedmiller and Braun, 1993]) and the transfer function of the input and hidden layers is the tangens hyperbolicus, thus we map all input data to the interval $[-1, 1]$. The output activation function is linear and unbounded.

5.2.1.2 Radial Basis Function Network

Radial basis function networks, RBFs are introduced in the appendix B.2. We here use the Matlab implementation (newrb) which requires the specification of the variance of the kernel function K, compare Eq. B.4, called the spread, a maximal tolerable network error (see Eq. B.1) and a maximal number of neurons. For the network predicting $\dot{\alpha}$ we set the spread to 5, the maximal tolerable network error to 1, and the maximal number of neurons to 3500, i.e. the number of training samples. This setting results in a net with 331 neurons. For predicting $\dot{x}$ we use the same settings, but a spread of 6, and obtain a net with 470 neurons.

5.2.1.3 Probabilistic Radial Basis Function Network

Probabilistic RBFs, are like RBFs but with an additional network layer which makes them useful for classification tasks (see appendix B.1). An input vector is compared to all RBF neurons and each such neuron is associated with a particular class. The closer the input vector is to a particular neuron the closer to one is the neuron's output. The additional layer selects the class associated with the neuron which outputs the highest value. We use the Matlab function newpnn which implements the method reported in Wasserman [1993][p. 35-55] to obtain a probabilistic RBF.

For predicting $\dot{\alpha}$ we use a spread of 6 and for predicting $\dot{x}$ a spread of 4. The number of neurons in both nets equals the number of training samples, i.e. 3500.

5.2.2 A Forward model for Curve Prediction

In this section we extend the forward model from the reduced straight line description to the entire curve. This requires a suitable curve parametrization for which the forward model can be learned.

5.2.2.1 Curve Representation and Approximation

To describe the detected lane marker, a planar curve, previously a polyline was used (compare Fig. 4.9). This was shown to work well within the described setup, however it induces information loss since it is not possible to reconstruct the initial curve from it. Here, we aim at improving this shortcoming, while fulfilling the same requirements as before, that is the curve description should:

(a) be concise, to ensure efficient storage

(b) allow efficient curve comparison

(c) be able to describe arbitrary curve shapes

(d) be able to describe deformable shapes

(e) be fast to achieve, i.e. the approximation procedure should be fast

Spline curves (e.g. [DeBoor, 1978], compare also B.8) meet most of these criteria[2] and several high-quality and free programming libraries are available to accelerate spline implementation and use, e.g. [Eberly, 2010; Lavoie, 2002]. However, since available implementations for spline approximations were found to not suffice our demands (e.g. requiring the number of control points for the approximation to be predefined, which in our case cannot be provided since depending on the detected lane marker) we designed our own approximation method tailored to better suit our needs, see algorithm 1.

Another issue is curve comparison. Finding out how much two curves or shapes resemble each other involves finding a metric which can be computed fast and which is robust, i.e. which

[2]Given they have a higher degree than 1, which would just be another polyline.

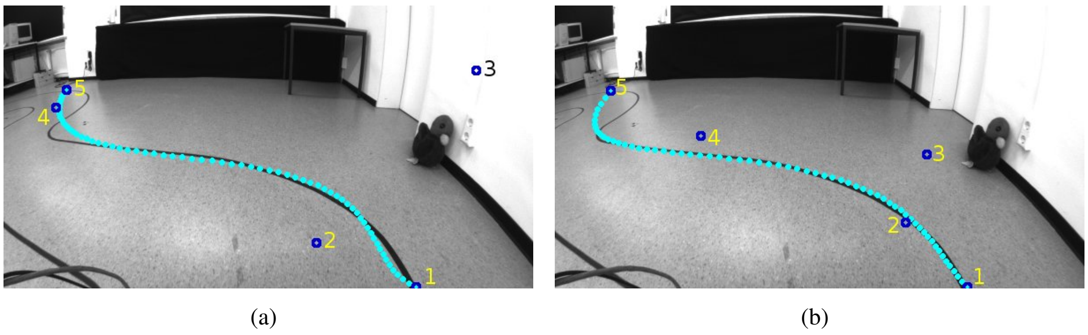


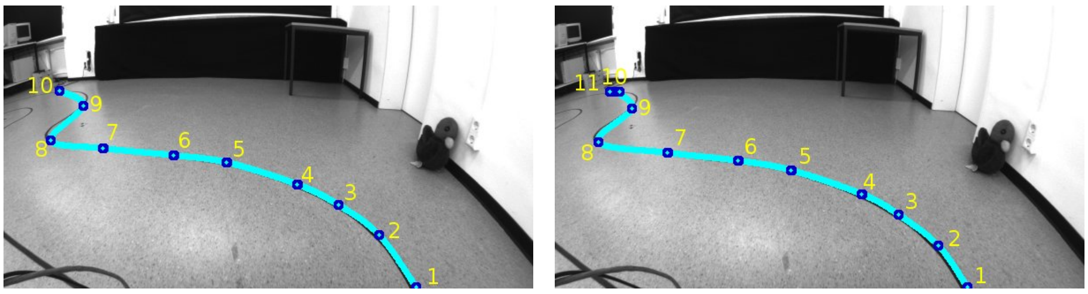
Figure 5.1: (a) Least squares approximation of the lane marker with a cubic B-Spline. Dark blue circles denote control points, numbered to emphasize corresponding points, turquoise points are points on the spline. (b) The same for the subsequent frame. It can be seen that although the two curves (in turquoise) are relatively similar the positions of their corresponding control points are not.

assigns small distances to similar curves (that appear similar to a human) and larger distances to differently looking curves. It is usually not feasible to use the curve representation directly to determine the similarity between two curves, for example by computing the summed distances between corresponding control points of two curves. This can be seen from Fig 5.1, where similar curves are represented by a configuration of very different control points. Therefore, it is common to evaluate the curves first, i.e. to compute points on the curve, and then compare the evaluations. In addition to computing the cross-correlation other techniques to compare parametric curves are the so-called *Hausdorff* distance [Huttenlocher et al., 1993], or the *Fréchet* distance [Alt and Godau, 1995], where the former is based on point sets and the latter takes the order of points on a curve into account. These methods are not suitable here, since detected lane markers sometimes differ almost exclusively in their lengths which would lead to great distance measures, although the curves actually appear similar for the length of the shorter one.

The approach taken in this work is to approximate the detected lanes in such a way that we can define a simple metric on the curve *representation*. For that we use an interpolating spline, i.e. one whose control points are *on* the curve. Here, we use the Catmull-Rom- also known as Overhauser-spline, abbreviated as CR-spline, see [Catmull and Rom, 1974] and appendix B.8.1. We use a piecewise approximation method which ensures that the number of control points is determined automatically. Furthermore, it ensures that control points for similar curves are at similar positions. As consequence we can now use the curve representation, i.e. the corresponding control points to compare two curves. The metric we use is the $l2$-norm, as given in Eq. 5.5.

$$d(c_1, c_2) := \|c_1 - c_2\| \qquad \text{difference between two curves, } c_1 \text{ and } c_2 \text{ where} \tag{5.5}$$

$$\|x\| := \sqrt{< x, x >}, \tag{5.6}$$



(a) (b)

Figure 5.2: Result of approximation with Catmull-Rom spline for two consecutively detected curves. (Same as in Fig. 5.1.)

with $< x, x >$ denoting the scalar or inner product, and c_1 and c_2 the spline curve descriptions, i.e. vectors of 2D points, i.e. the control points of the CR-spline.

The used approximation routine is given in pseudo-code below in Algorithm 1.

Algorithm 1: Approximate list of pixels with CR-Spline.

```
Input: curvePixels[] //(detected lane marker as) list of pixels
Output: curveSpline[] //(detected lane marker) approximation as CR-Spline
1 set start and end point of curveSpline to first point of curvePixels;
2 while curveSpline[endpoint] != curvePixels[endpoint] do
3     while residual between curveSpline and curvePixels< $\epsilon$ do
4         move curveSpline[endpoint] towards curvePixels[endpoint];
5     if residual between curveSpline and curvePixels $\geq \epsilon$ then
6         insert new curveSpline control-point;
```

The variable ϵ in algorithm 1 indicates a specifiable threshold (we used the value 100). A resulting approximation is shown in Fig. 5.2(a) and (b).

In summary, the used approximation procedure and spline representation fulfill all of the given criteria. The representation is considerably smaller than the list of pixels of the original curve, thus it is a concise description. Similar curves have similar representations and thus comparisons can efficiently be done based on representations instead of evaluated curves. Arbitrary shapes can be approximated with CR-Splines and they can describe deformable shapes, too by moving the control points. In addition the procedure is fast, on 1000 lab images the approximation takes 2.81 ms on average per image. Thus, the task to find a suitable curve representation is solved.

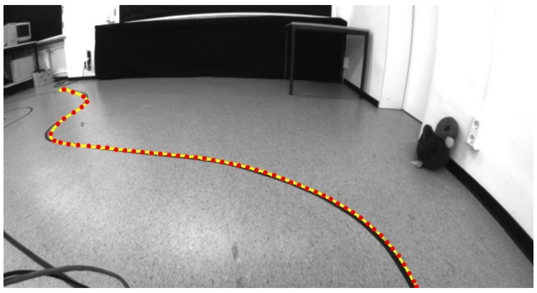


Figure 5.3: Producing training data. The yellow line is the detected lane marker and the red points are used as training samples.

5.2.2.2 Curve Prediction

To predict the expected curve shape after an action was executed we predict the shift of each control point of a spline separately. We work on the same data as before, i.e. the velocity is constant and we consider only steer in this test. The precise mapping we realize is given in Eq. 5.7.

$$(x_i, y_i, \theta_i, steer) \mapsto (\dot{x}_i, \dot{y}_i), \tag{5.7}$$

here x_i and y_i are the horizontal and vertical coordinates of the i'th control point of a spline, θ_i is the direction of the point's gradient, and steer the executed action command. The output is the expected shift of the control point. For the realization of this mapping we again use a MLP which we realize with the fann library. We train it with RPROP with a learning rate of 0.7 and use two hidden layers with 20 and 30 neurons. The transfer function of the input and hidden layers is the tangens hyperbolicus, thus we scale all inputs between $[-1, 1]$. The output activation function is linear and unbounded. Again we use the available 3500 image-action pairs for training. However, since the curve control points lie *on* the spline any curve point can be used as training sample. Therefore, we sample the spline curve every 10 pixels as shown in Fig. 5.3, and this way achieve in total 125,283 training samples.

5.3 Results

5.3.1 Forward Model for the Reactive Controller

We first test the issue of missing sensory input. In the described setup the robot, when relying exclusively on the RC, is not able to issue adequate control commands when the detected lane is outside its field of view. In this case the default behavior is to simply repeat the last action. This is not sufficient to keep the robot on track in case of high curvature curves, as can be seen in video "Robot_Without_Forward-_Model.avi" (on enclosed cd, or online under www.markelic.de/professional/Documents/), summarized with snapshots in Fig. 5.4. (Of course, for solving this particular problem the other lane marker can be used in addition, however, we are interested in the principle problem that a reactive system is limited because of its dependency on reliable continuous input.)

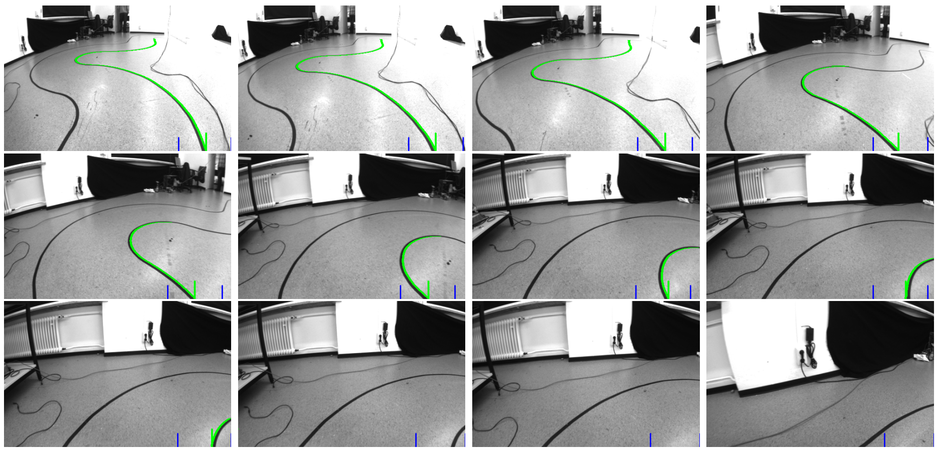


Figure 5.4: In sharp turns the necessary sensory input is not in the robot's field of view. The detected lane marker is green, the blue bars indicate the tracked lane marker's starting interval where the green bar is the estimated lane marker's starting position.

To test the effect of using a learned forward model, we let the robot drive on the same difficult part of the track as shown above (this part is marked with a "*" in Fig. 4.1(d). Now, in case of missing sensory information the robot issues motor commands based on its belief about the marker's position which is iteratively updated by the forward model according to the issued action. The result is shown in video "Robot_With_Forward_Model.avi" (on enclosed cd or online under "www.markelic.de/professional/Documents/") and summarized with snapshots in Fig. 5.5. It can be seen that the robot is now able to stay on track. In addition it easily re-catches the marker when it reappears in the field of view.

In addition to this observable behavior, we plot the state variables predicted by the forward model in comparison to the run without the model as well as the resulting steering signal in Fig. 5.6. It can be seen that although the observable behavior is much better than without forward model, the variables do not seem to be very well estimated. Of course there is no true data that the estimation can be compared to however, the fluctuations, especially in Fig. 5.6(b) are high, indicating a high variance and thus a possibly unreliable signal.

This can be understood when looking at the results shown in Fig. 5.7, where the performance of each learned forward model for each state variable on test data is shown together with a plot of the residual. The residuals of the MLP are the highest which is better seen in Fig. 5.8.

The look-up-table performs best concerning the orientation and the RBF performs best concerning the second state variable, i.e. the position, closely followed by the LUT. In fact, the residuals for all models are relatively high, indicating a mediocre correlation between the used state variables

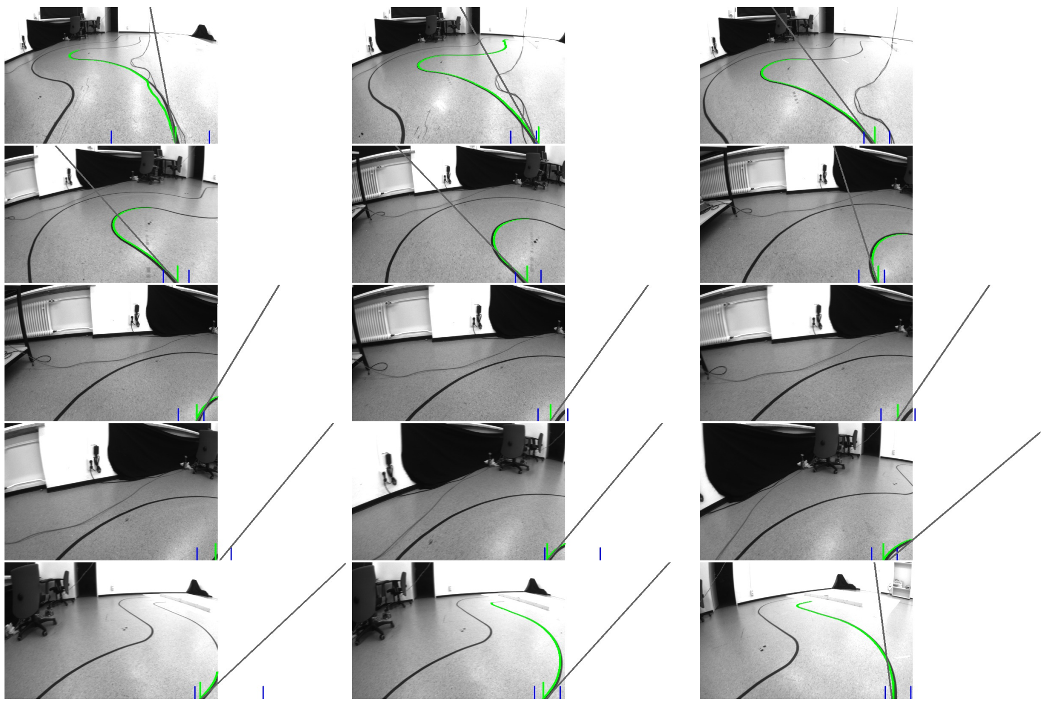


Figure 5.5: The gray line approximates the close-by street trajectory and is used by the RC to issue steering commands. If the marker slips out of the robot's field of view its position is updated by the forward model reflecting the robot's belief about it.

and the issued actions. This can also be inferred by noise, for example slip of the robot.

It was said that forward models are considered to be a basis for the ability of planning, where plans arise by coupling a forward model with an inverse model which allows the alternating prediction of a next state, an action based on this predicted state, again a next state and so on. To test this type of planning capacity we here apply each learned forward model to each entry of the test set to predict the next 99 states (the number was arbitrarily chosen and corresponds to approx. 8 seconds of driving) by coupling it with the RC (as substitution for an inverse model). We then compute the residual between these 800 sequences of 99 predicted states and the real data, and finally compute the mean over these 800 residual-sequences. We plot the mean and the confidence interval (95%) in Figs. 5.9(a) and (b). Thus, Fig. 5.9(a) shows how fast the discrepancy between the predicted and the real steering signal increase over a plan of a length of 99 timesteps on average. To better understand this result we also plot the mean residual for simply taking the latest steering command (labeled 'same value') which we used as default behavior for the robot in case of missing sensory input. It can be seen that with all forward models the average residual is lower for approximately the first 80 timesteps than when using 'same value'. It can also be seen that between timestep 5 and 40 the error increases rapidly, which means that plans get worse the further they reach into the future. In section 4.3.2 we introduced the Planner which generates plans as fixed action sequences. These are different from the plans generated by the coupling of internal models

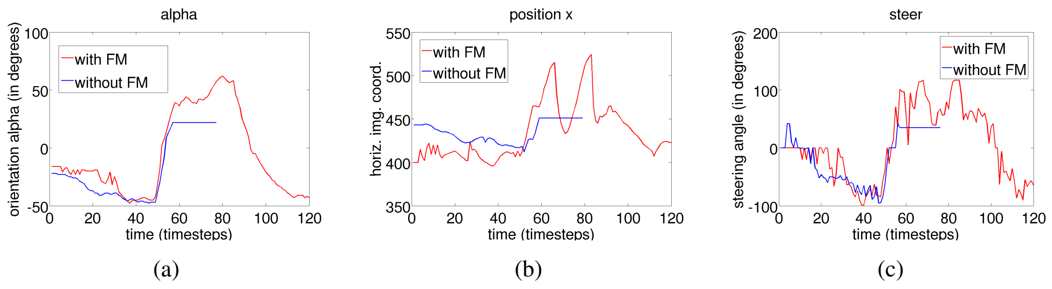


Figure 5.6: The predicted state variables α (a) and x (b) and the steering signal (c) from the run with forward model compared to the run without.

in that they are less flexible and do not consider any feedback information. A plan from the Planner is a sequence of actions $(a_t; a_{t+1}; a_{t+3}; \cdots)$, where a plan from the coupling is a more complex sequence of alternating state-action pairs $(s_t, a_t; s_{t+1}, a_{t+1}; s_{t+1}, a_{t+3}; \cdots)$. For convenience we will refer to a plan from the Planner as “planP” and from the coupling of internal models as “planC”. Since both contain a sequence of actions we are interested in a comparison of the performance of these two types of plans, thus we plot the performance of the planner (in this case we trained it on the same data as the forward models) on the test data, compare Fig. 5.9. We see that its error and variance are considerably lower until timestep 80 than for all planCs. The steep increase of the error of the Planner after timestep 80 is caused by the way planPs shorter than 99 timesteps were handled by the comparing routine. Missing values were put to a fixed arbitrary number. Thus, this high error (and also its variance) is an artifact and can be neglected. It does indicate however that retrieved planPs cover on average 80 timesteps.

5.3.2 Results for the Forward Model for Curve Prediction

To test the capacity of the forward model for the entire curve we proceed as follows: In each image of the test data of the recorded drive we detect the right lane marker and approximate it as CR-spline. Starting from this initial detection we let the network predict the next curve based on the conducted action. The output is the basis for the next prediction and we repeat the procedure. To visually inspect the performance of the forward model we plot the predicted curve on top of the true curve and show some typical results in the image sequences in Figs. 5.10, 5.11, 5.12, 5.13. From these sequences it is observable that the predictions coincide well with the true curves for a variable number of timesteps, e.g. 19 in Fig. 5.10 or 10 in Fig. 5.12, and then start to diverge. It is also visible that single control points that are predicted incorrectly lead to unusual lane shapes, e.g. at the end of the sequence in Fig. 5.13.

To evaluate the performance of the forward model more globally we measure the difference between the predicted and the actual curve, using the metric described in section 5.2.2.1. In addition, we normalize the calculated difference by dividing by the number of control points. If the amount

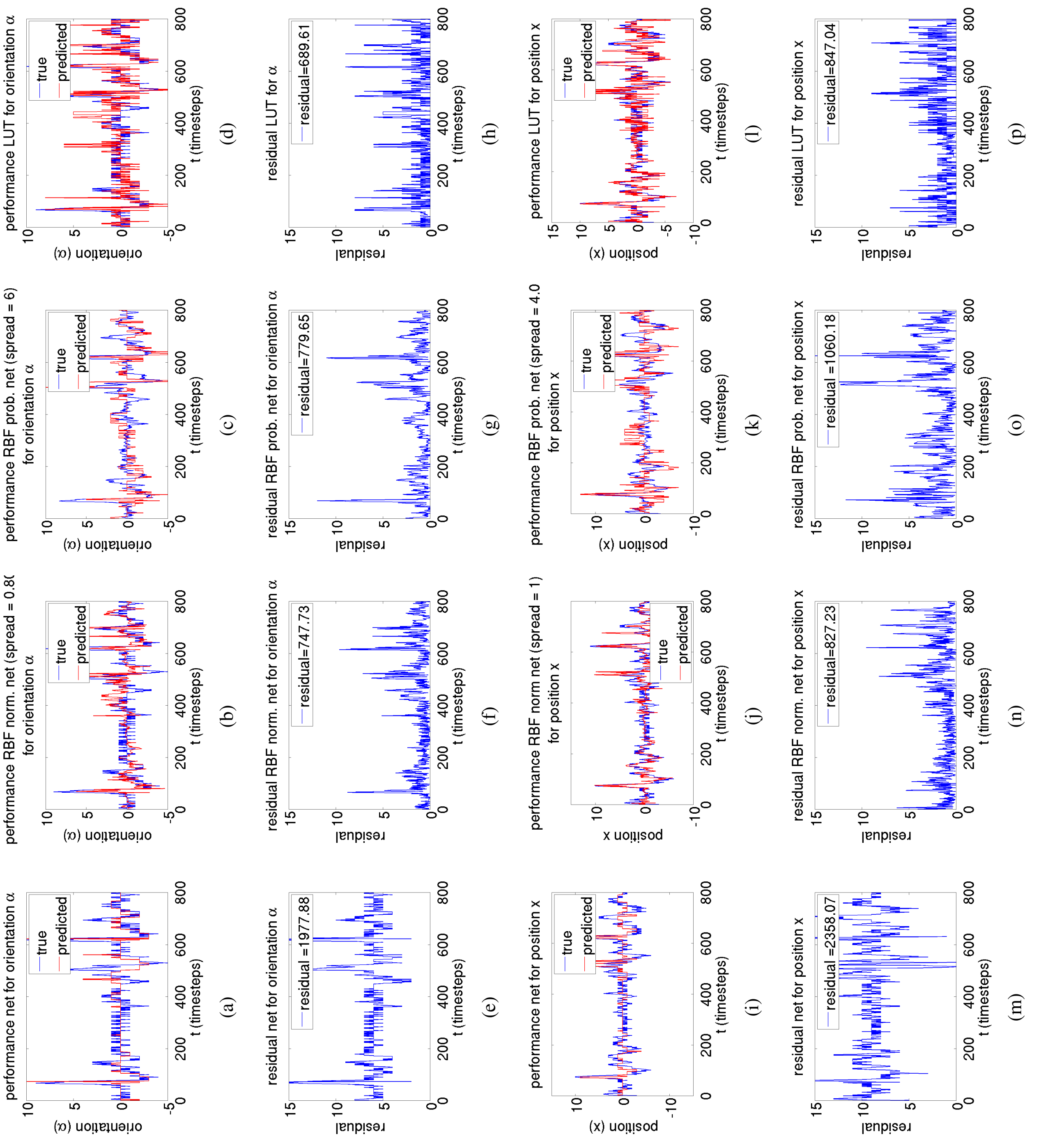


Figure 5.7: Performance of all trained forward models on test data for the two state variable α and x.

of control points of both curves differ we consider the minimal set of corresponding control points. Thus, for each of the 800 test data entries we obtain a sequence of 99 subsequent prediction errors. Again, we compute the mean difference and the confidence interval over these 800 sequences which is shown as plot 'fm' in Fig. 5.14. For comparison we repeat the same procedure but instead of using the predicted curve we use the first detected curve in such a prediction sequence, i.e. we compute the error arising if no prediction is possible and the system must rely on the last known measurement. We denote this with 'sv', indicating 'same value', analogously to what we did for evaluating the forward model for the RC. The result is also plotted in Fig. 5.14. It can be seen that the performance of the forward model is significantly better than that of 'sv' for the first approx. 45 timesteps but from then on 'sv' has a lower prediction error. This is explainable since the latter describes a fixed shape that is compared to a detected lane marker, whereas the curve described by the forward model can take any arbitrary shape. Over time the predicted curve will look more and more different from a lane marker, as already indicated at the end of the sequence in Fig. 5.13 and this explains the steadily increasing error. In addition, the forward model cannot predict parts of the street that were not visible in the frame from which the prediction series started. From the shown image sequences it becomes clear that only the first few predictions (≤ 20, i.e. approx. two seconds) are close enough to the real street to be of use.

5.4 Conclusion

The goal was to test if employing a learned forward model to the given task can lead to some of the benefits ascribed to forward models.

5.4.1 Conclusion Concerning the Forward Model for the Reactive Controller

We tested the RC in conjunction with a learned forward model with respect to the two issues a) handling of missing sensory input and b) planning capacity. It was shown, that a learned forward model could indeed serve as an internal representation for the RC and for a particular situation

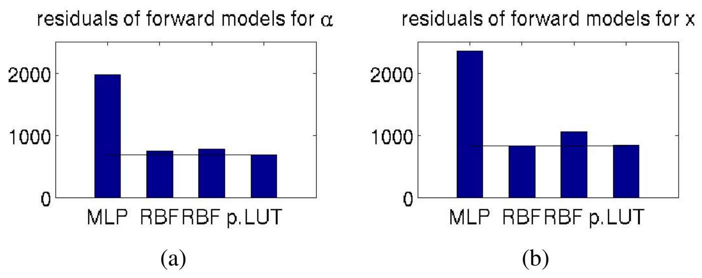


Figure 5.8: Comparing the performance of the different forward models on the test data for $\dot{\alpha}$ (a) and $\dot{x}$ (b). RBF p. denotes the probabilistic RBF.

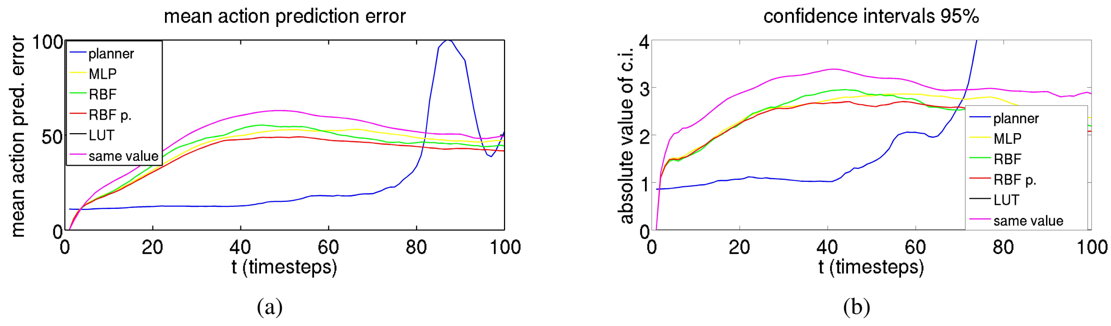


Figure 5.9: Comparing planning abilities of the coupling of each learned forward model (MLP, RBF, probabilistic RBF, LUT) and RC to taking the same last action command (same value) and the planner. (a) shows the mean and (b) the confidence interval of the expected value.

substantially improved the robot behavior, i.e. prevented it from loosing track. A possible conclusion that might be drawn from this is that learning forward models presents a way to augment simple reactive systems with internal representations and thus improve their shortcomings while preserving their advantages. Although this appears reasonable and also very appealing we encountered several difficulties with this. We did show that all learned forward models had relatively high residuals, thus, the acquisition of forward models is not a simple task. It requires highly correlated features and "good", i.e. (relatively) noise-free data. Furthermore, it can require very large amounts of data, depending on the complexity of the model to be learned and the size of the state space spanned by the state variables. A further very critical issue is that it is not clear how extrapolation should be handled. Unknown situations which are not within the convex hull of encountered situations cannot be guaranteed to be handled appropriately by the model.

Concerning the proposed planning capacity. We showed that the performance of such plans obtained by the suggested coupling of inverse (here RC was used) and forward model was slightly better than no planning at all, i.e. when simply repeating the same last value. However, the error in the prediction increased much faster compared to the planP (generated by the proposed Planner). Insufficiencies of the models and noise in the data quickly accrue and at some point the computed planC might diverge. Since the Planner returns a fixed sequence of actions that does not consider any feedback information it is less flexible than planC but at the same time much more robust since it is not affected by increasing errors and uncertainties. This shows an advantage of feed-forward over feed-back control. In summary, we believe that two conclusions are possible: 1) If biological systems indeed rely on the suggested coupling for planning, regardless the quality of their acquired models, errors accrue here too and therefore they must have a way to handle the increasing uncertainty. Which leads to the question how this is achieved? 2) The other possible conclusion (which in fact does not exclude the one above) is that this error is not handled at all and that biological systems do simply not use the suggested coupling for generating plans covering more than very short periods. It seems plausible that instead the same principle is used with data

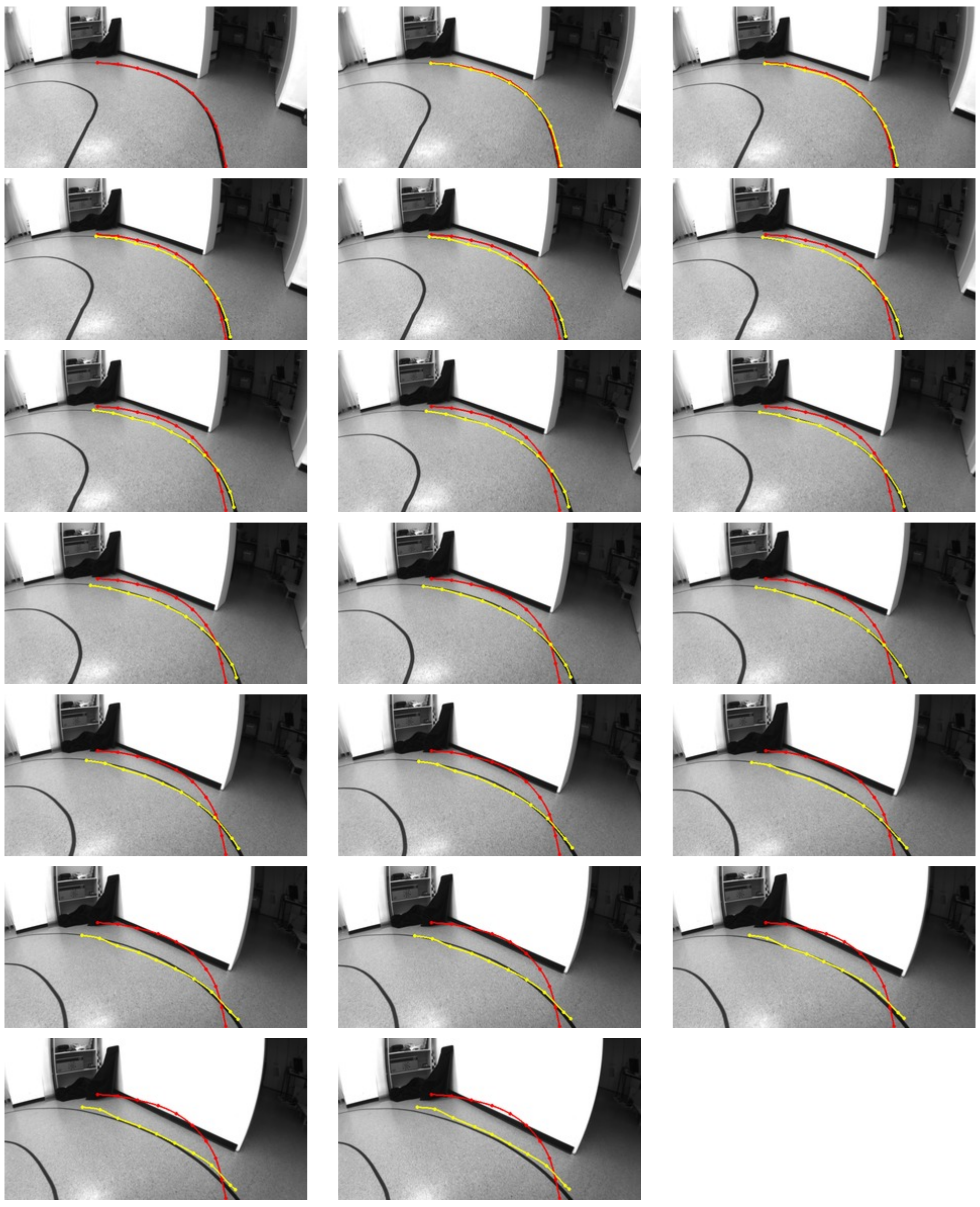


Figure 5.10: Showing the performance of the trained forward model. The red curve is the initial detection and the yellow curve the prediction based on the detection and the conducted action.

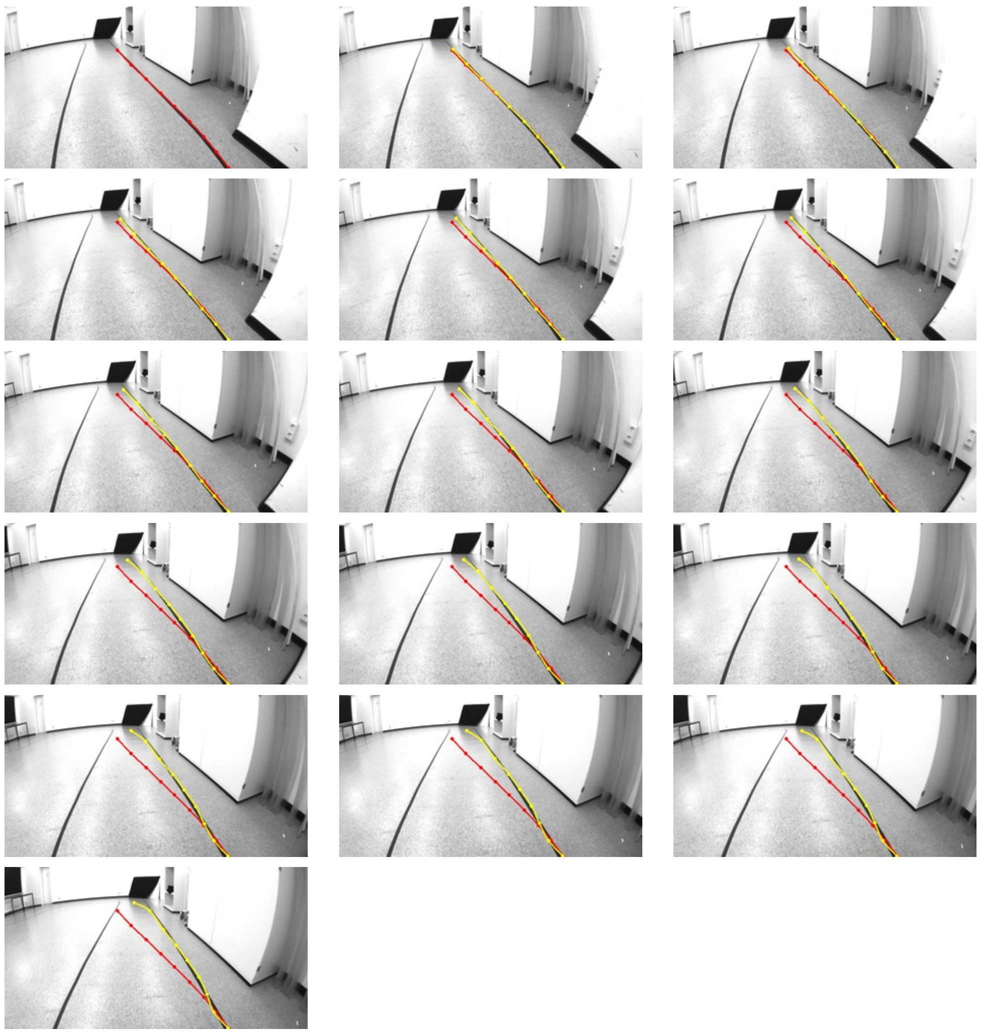


Figure 5.11: Another example of the performance of the trained forward model. (The red curve is the initial detection and the yellow curve the prediction.)

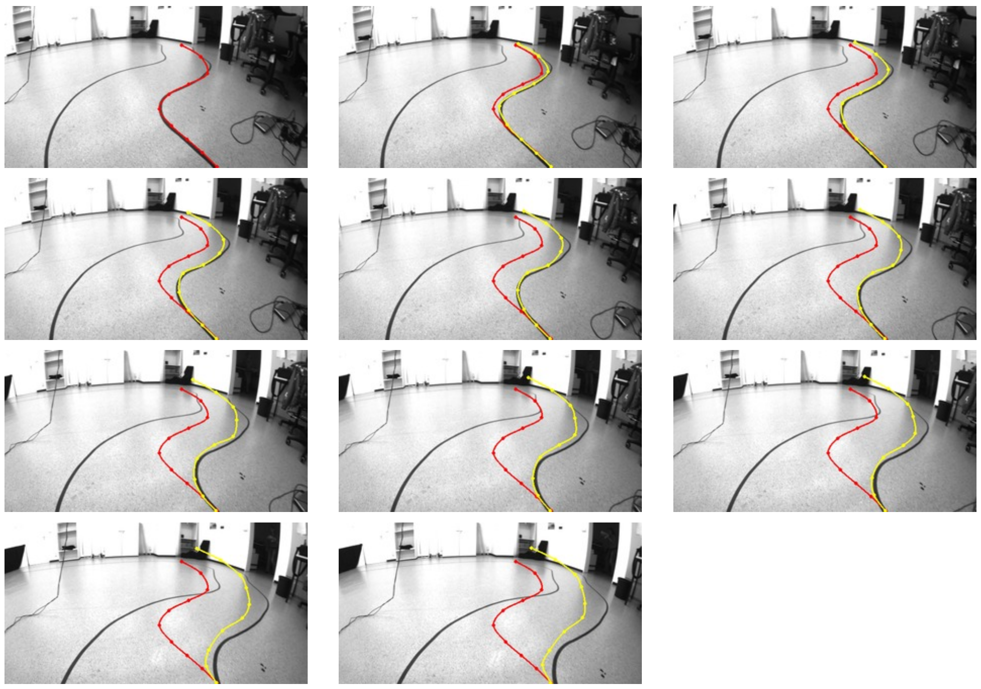


Figure 5.12: The prediction diverges after 8 timesteps. (The red curve is the initial detection and the yellow curve the prediction.)

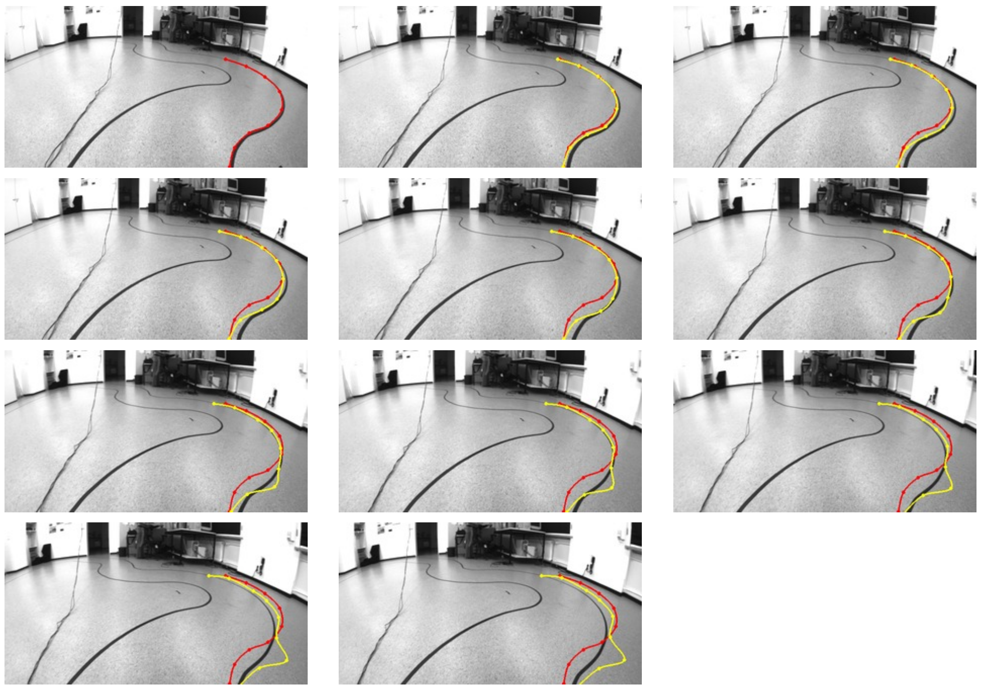


Figure 5.13: Showing how a single mal-predicted control point leads to an unrealistic lane shape. (The red curve is the initial detection and the yellow curve the prediction.)

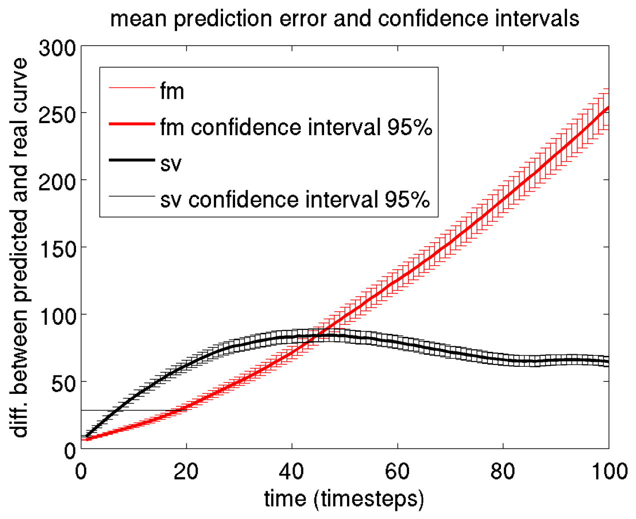


Figure 5.14: Mean difference and confidence interval between the predicted and the true curve, denoted 'fm'. The prediction length was 99 timesteps. In comparison the difference between the initially detected curve and the true curve is shown, denoted as 'sv'. It can be seen that the curve predicted by the forward model is closer to the true curve for approx. 40 timesteps, from then on it diverges.

compiled into a richer representation to obtain “higher-level” plans, which are flexible *and* robust against disturbances. E.g. when complete chunks are worked on instead of atomic actions. In this case one should investigate the formation, chaining and modification of chunks. In fact, this conclusion fits very well to the idea that chunks are organized hierarchically, compare section 3.2.2.

In summary, concerning the practical use of forward models in our described setup we did encounter some promising benefits, e.g. the performance of the system even with only a mediocre forward model was better than without one. However, they are expensive to achieve and methods for automating their acquisition and use are required.

5.4.2 Conclusion Concerning the Forward Model for Curve Prediction

For predicting the entire curve we used an interpolating spline approximation and a neural network which we trained to predict the shift of curve points according to conducted actions. By using this for the spline control points we could predict the entire expected curve shape.

We evaluated the predictive capacity of this forward model by plotting predicted lane markers on top of the current image and by plotting the average prediction error over the test data for 99 timesteps. We also showed the error that resulted when no prediction was possible but the last detection was used instead. It could be seen that the forward model performed considerably better on average for the approximately first 40 timesteps before it diverged and that an error below 30 is acceptable which is only the case for the first few (*leq*20) timesteps.

It could also be seen that single mal-predicted control points lead to very unusual curve shapes.

By unusual we mean unusual for lane markers. This is a known feature of active contours (see appendix B.7) which our approach is related to. A possible solution may be the introduction of a so-called shape-space [Blake and Isard, 2000, compare chapter 4] which restricts the possible control point configurations to “good” shapes which in our case would correspond to possible lane markers.

It is important to note that this forward model differs from the one presented for the RC in that it cannot be used for predicting the curve if it is not in the camera’s field of view. This restriction results from the fact that we always start with a visible curve.

In summary, the presented forward model can predict the evolution of the visible curve (i.e. in the camera’s field of view) very well, but only for a very limited number of timesteps. This cannot be used for maintaining a long term belief about the lane marker however, the very attractive feature of our model is its potential use for tracking. So far we track only the lane’s starting position. By using our forward model in combination with a nonlinear state estimator we can possibly overcome the main shortcoming of our lane detection method, which is its incapacity of handling larger occlusions. This should be achievable by tracking individual spline control points, which if the lane being tracked is occluded can rely on the prediction of the forward model. This is very promising and pursued in follow-up work.

Chapter 6

Outdoor Driving - The DRIVSCO System

"In the end we retain from our studies only that which we practically apply."
Johann Wolfgang von Goethe

As pointed out before, this work aimed at a working real world application. This was realized in form of the "DRIVSCO System", a learning system for basic human driving implemented in the course of the European DRIVSCO project.

This chapter describes this work (also submitted for publication in [Markelić et al., 2010]) which is the result of a joint effort by the international group members. The output of this thesis considerably contributed by providing the lane extractor and the learning part which was trained with provided real driving data. Contributions of the project partners are indicated as such.

Since this application is based to great extent on the methods and ideas developed earlier, parts of this chapter are analogous to previous sections, especially section 4.3.

6.1 Introduction

Advanced driver assistance systems (ADAS) that adapt to the individual driver have a high potential for the car industry since they can reduce the risk of accidents while providing a high degree of comfort. Conventional systems are based on a general moment to moment assessment of road and driving parameters. To arrive at a judgment of the current situation they use control laws from which they derive output signals to aid the driver [Mammar et al., 2006; McCall and Trivedi, 2006; Safelane, 2008; Li et al., 2006]. However, inter-individual differences in driving can be large, e.g., it is known that different people pursue different driving styles, [Ulleberg and Rundmo, 2003], and driving strategies [Kandil et al., 2009]. It is difficult for conventional control systems to accommodate these differences which leads to suboptimal driver support. However, this can have negative effects on the driver's behavior [Brookhuis et al., 2001; Lindgren and Chen, 2007], i.e. safety is decreased instead of increased. Observations like these were the motivation for us to investigate and build a system that drives in the same way as its user(s). These reasons also lead to the situation that current R&D efforts of the car industry focus on systems which take the driver and its behavior explicitly into account [Coughlin et al., 2009; Hoch et al., 2007]. In addition to the safety aspect, such systems will also be accepted more easily by the user because they will provide more comfort; an important sales argument.

In the current study we will describe a system based on imitation learning, hence, a system that learns to interpret basic aspects of the road (lanes) in the same way as its driver, reproducing the driver's actions. In addition we demonstrate its use as a basic driver assistance system by issuing warning signals if the driver deviates from his/her predicted default behavior. The so-called DRIVSCO[1] system is realized by a multi-threaded, parallel CPU/GPU architecture. It is vision based, operates in real-time on real roads and also includes a novel form of data driven detection of independently moving objects (IMOs). The system has been designed for the use on motorways and country roads.

Before describing the details of our system and comparing it to the literature (see State of the Art 6.3), we will shortly explain its structure as a guideline for the reader (Fig. 6.1). Thus, this chapter is organized as follows: in section 6.2 the overall structure of the system is presented. It is compared to the state of the art in section 6.3, its realization explained in section 6.4 and results presented in section 6.5. In section 6.6 we conclude and discuss the presented work.

6.2 System Overview

The overall structure of the DRIVSCO system is shown in Fig. 6.1. It is in great parts analogous to the structure of the indoor-system, shown in Fig. 4.8.

Again, the yellow box, "human sense-act", symbolizes the human driver who senses the environment, denoted by the "world" box, and responds to it with adequate driving actions, "act". At the same time the system senses the environment, "system sense", via recorded image frames. The sensing process consists of detecting left and right street lanes, "sense lane", and independently moving objects, IMOs, "senseIMO". The latter refer to objects that move with respect to the static environment, where detection is not restricted to predefined objects such as other cars or motorcycles. If an IMO is detected a warning is triggered, "warning IMO", if it is considered to affect the driver (see section 6.4.4). Once the lane detection is initialized it is compared to a prediction about the expected lane positions, "generate lane expectation", which is obtained by using previously captured human action data and the previously detected lane, indicated by the notation $t-1$, to predict the current lane structure. This way incorrectly detected lanes can be filtered out (see section 6.4.5). The learning system is realized by the perception-action repository,

[1]short for Driving School.

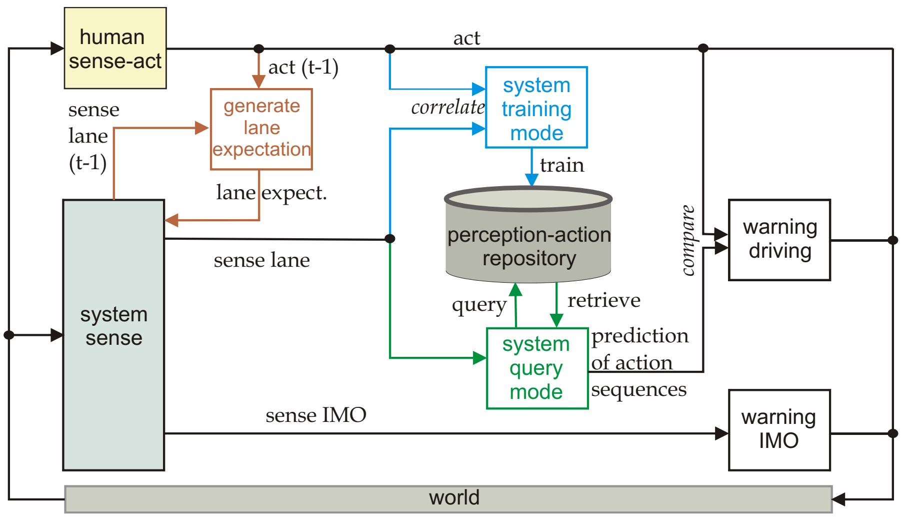


Figure 6.1: Block diagram of the DRIVSCO overall system structure. The notation $t - 1$ indicates an earlier time frame.

PAR, again depicted as the gray container in the figure, which serves as the system's memory (see section 6.4.6). As already explained in section 4.3, the experience stored in the PAR is a state vector containing a concise lane description extracted from the current image frame and sequences of preceding and succeeding action data. and there are two modi: 1) *training*, during which the system gathers experience to fill the PAR, denoted in blue "system training mode" (see section 6.4.6.3), and 2) *retrieval* during which the system queries the PAR to retrieve stored knowledge denoted in green "system query mode" (see also section 6.4.6.3). Based on such accumulated PAR returns, action plans for expected human steering and acceleration behavior for lane following are predicted, "prediction of action sequences" (see section 6.4.6.4). To demonstrate the system's use as an assistance system the predicted action sequence is compared to the driver actions and a warning is issued if they are differing too much, "warning driving" (see section 6.4.6.5). In other words, the driver is informed when deviating from its learned default driving behavior. The warning signals are displayed on a screen close to the driver, shown in Fig. 6.3c and 6.4.

6.3 State of the Art

Since this work aims at generating sequences of future driving actions based on visual sensory input it is closely related to vision based autonomous driving. As discussed in section 2.2, approaches in this field are often based on control theoretic white-box methodologies which heavily rely on predefined knowledge imposing certain restrictions on such methods. In addition these systems do not, or only in a limited way, provide the means for individualization. By contrast, imitation learning [Argall et al., 2009] aims at extracting a policy for a given task by using examples provided by a teacher. This reduces the amount of required a priori knowledge and the need of explicit programming and thus facilitates human computer interaction. A famous example of imitation learning for driving is the ALVINN system, see section 2.2.3. Further imitation learning work by Pasquier describes the learning of several driving skills with a fuzzy neural network [Pasquier and Oentaryo, 2007]. Such motor skills are difficult to formalize and the fuzzy net represents a way of capturing semantic rules from continuous control input. It was aimed at emulating the role of the cerebellum during cognitive skill learning, and the algorithms for lane following were tested in simulation. Similar work, but dealing with helicopter flying was reported in [Sammut et al., 1992]. A novel form of inverse reinforcement learning, [Ng and Russel, 2000], was introduced in [Abbeel and Ng, 2004] and applied to learning particular driving styles from example data obtained from simulation.

The EU funded project DIPLECS [Diplecs, 2010] reports similar goals to those pursued here, however, DIPLECS does not aim at building a complete system. Research conducted at Motorola [Hwang et al., 2003] aims at building an adaptive driver support system using machine learning tools. Its architecture was partially implemented in a prototype system built upon a simulator.

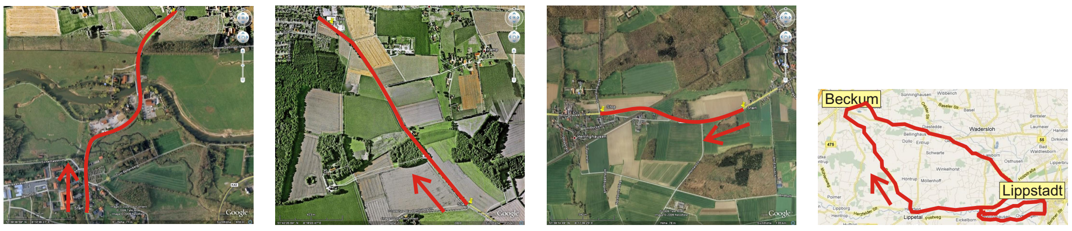
In addition to lateral control, new generations of driver assistance systems will contain support for longitudinal velocity control during curve negotiation, e.g. [Freymann, 2008]. Current, (non-imitation based) methods are usually not taking the street trajectory into account but simpler aspects like an obstacle in front, e.g. Adaptive Cruise Control systems using radar or laser for obstacle detection, known speed limits, e.g. Intelligent Speed Adapters and Limiters [Brookhuis and de Waard, 1999], or leading vehicles, e.g. [Tahirovic et al., 2005]. Focusing on curve negotiation and based on imitation learning are [Pasquier and Oentaryo, 2007; Partouche et al., 2007] and [Kwasnicka and Dudala, 2002], which all employ fuzzy neural networks trained on human control data.

This work, unlike similar approaches, describes the realization of a complete system implemented on a real car that learns the prediction of action plans, i.e. sequences of steering and acceleration actions, together with a novel form of independently moving object detection. One big difference to others is that here data is obtained from real car driving and not from a simulator.

6.4 Submodules and Methods

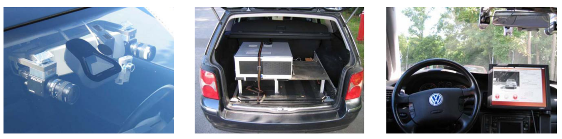
6.4.1 Hardware

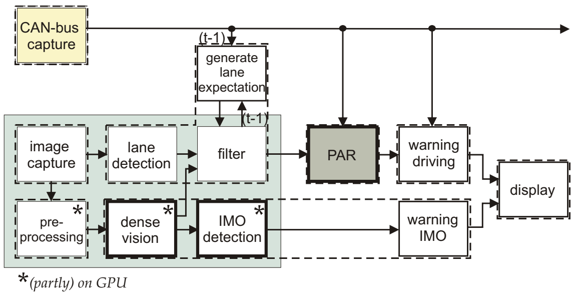
The used car is a Volkswagen Passat provided by the DRIVSCO partner Hella KGaA Hueck & Co (Lippstadt, Germany). The following sensory data is accessed via the CAN-bus: steering angle, indicating the steering wheel's position and taking values between $-360°$ (left steering) and $360°$ (right steering), velocity in km/h, longitudinal acceleration taking values between $-10\text{m}/s^2$ and $10\text{m}/s^2$, and curve radius measured by a gyroscope and taking values between $-15,000\text{m}$ and $15,000\text{m}$. Furthermore a camera stereo rig is mounted behind the windscreen as shown in Fig.



(a) (b) (c) (d)

Figure 6.2: The tracks on which training data was recorded, (a) s03 (1km), (b) s05 (2km), (c) s06 (1km), (d) long tour between Lippstadt and Beckum (69km).





(a) (b) (c) (d)

Figure 6.3: (a) Mounting for the stereo camera system. (b) Fixation of the computer in the trunk. (c) Installation of the monitor in the car. (d) The realization of the DRIVSCO system. Boxes denote processing stages and the dashed frames indicate individual threads. Arrows denote the information flow, with the exception that the display stage connects to *all* stages. The ”PAR“, ”dense vision“ and ”IMO detection“ are key parts of the system. The notation $t-1$ indicates an earlier time frame.

6.3(a). We use two color Pulnix TM-1402CL cameras which deliver 1280×1024 raw Bayer pattern images at a frequency of 20 Hz. Furthermore a Global Positioning System (GPS) receiver is added to allow the visualization of driven routes, for which we used Google Maps, compare Fig. 6.2. All computations are carried out on a PC with an Intel Core i7 - 975, 3.33 GHz quad–core processor with simultaneous multi-threading (SMT) enabled and 12 GB RAM. The used graphics cards are an NVIDIA GTX295 for computation and a smaller one for displaying purposes. The PC is kept in the car’s trunk as shown in Fig. 6.3(b) and the output of the system is shown on a screen next to the steering wheel as shown in Fig. 6.3(c).

6.4.2 System Architecture

The DRIVSCO system is realized as a multi-threaded, pipelined real-time system where shared memory is used for interprocess communication. Due to the complexity of the pipeline structure and the involvement of multiple CPU-cores and multiple GPUs in its computation, we have developed our own modular architecture. This work was provided by project partners. By simplifying the coordinated use of the heterogeneous computational resources in modern computers, this architecture allows for independent development of the individual processing stages. To the knowledge of the authors of [Markelić et al., 2010] no such CPU/GPU pipeline framework combining task- and data-parallelism hitherto existed, allowing a stage to share data with multiple other stages — e.g. the output of the preprocessing stage is used by both ”lane detection” and ”dense vision”, as seen from Fig. 6.3(d). The system structure from Fig. 6.1 is realized via the architecture shown in Fig. 6.3(d). Each information processing entity is referred to as a stage and runs in a thread as indicated. All stages are connected through a pipeline where communication is realized by shared memory buffers to which processes can write to and read from. The processing time of each individual stage is below 50ms and the entire system works at the camera rate of 20 Hz. The “preprocessing”, “dense vision” and “IMO detection” (compare Fig. 6.3(d)) stages involve massive amounts of computations but achieve a frequency of 20 Hz through the use of two GPUs while all other processes run on the CPU. The “CAN-bus capture” stage is triggered every 50ms and makes the relevant car CAN-bus data available to other processes. During the “image capture” stage raw camera data is received and white level calculations are performed to control the camera’s shutter time. The images are then undistorted, rectified and downsampled to 640×512 pixels during the “preprocessing”. The boxes “lane detection”, “dense vision” and “IMO detection” are explained in section 6.4.5, 6.4.3, and 6.4.4. In addition a special display unit is integrated which connects to all buffers in the pipeline allowing the user to view the output of *any* stage, including the generated warning signals by means of a graphical user interface (GUI). A screenshot of this GUI is given in Fig. 6.4. It shows a current image with detected lanes on the left, and a history and prediction of steering angles of two seconds on the right. The prediction is plotted in blue and the actual human steering in red. In addition the computed egomotion from the IMO detection stage is displayed in green. The gray dashed boundaries around the prediction indicate predefined threshold values used to issue a warning if exceeded by the actual driving. In this case the ”unexpected driving” button below the displayed lane detection image is flashed. The system can also conveniently be used in an off-line mode to replay recorded scenes which is important for error analysis. Note, this display is designed for R&D purposes and should be simplified for a real driver assistance system.

6.4.3 Dense Vision

The work described in this section is contributed by project partners. The visual cues used for the IMO detection are computed during the “dense vision” stage, see Fig. 6.3(d). Dense vision algorithms process the visual signal in its entirety, as opposed to sparse vision algorithms that only operate on interest points such as edges or corners. The cues are dense disparity (the horizontal difference in image location for corresponding pixels in the rectified left and right image) and optical flow (the 2D image motion of each pixel). The algorithms used rely on the phase of quadrature pair Gabor filter responses [Daugman, 1985] (Fig. 6.5B), extracted at multiple orientations and

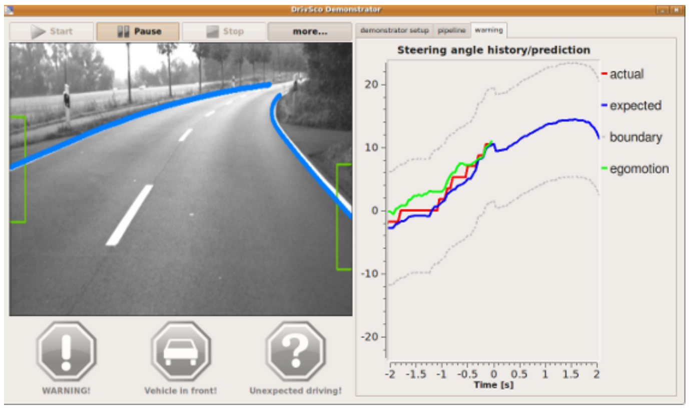


Figure 6.4: One tab of the system GUI, showing the detected lanes in the current image on the left and a history of prediction and steering angles of two seconds on the right along with the computed egomotion. Thresholds are shown in gray.

scales, to establish correspondences. The GPU implementation [Pauwels and Van Hulle, 2008] of a phase-based algorithm [Gautama and Van Hulle, 2002] is used to compute optical flow (Fig. 6.5C). This algorithm integrates the temporal phase gradient across orientation, and gradually refines its estimates by traversing a Gabor pyramid from coarser to finer levels. The optical flow is computed for the left video stream only. The dense disparity algorithm (Fig. 6.5D) is very similar to the optical flow algorithm but operates on phase differences between the left and right image as opposed to temporal phase gradients. These aspects are described in more detail in [Sabatini et al., 2007].

6.4.4 IMO Detection and Warning

Recent (off-line) approaches for the detection of IMOs have achieved very good results using model-based techniques [Leibe et al., 2008, 2007]. One limitation of such approaches is the difficulty to detect IMOs early on. Distant objects will occupy only small patches in the image that are difficult to match to the model. Another limitation is that moving objects, for which no model is provided, cannot be detected and are simply ignored by the system. In this work we rely on a model-free detection mechanism that is more general in the sense that it will respond not only to cars, but to any sufficiently large (11×11 pixels) moving object. The IMO detection component combines dense vision cues (optical flow and stereo, see section 6.4.3) in real-time to compute egomotion and independent motion (the parts of the image that move with respect to the static environment) to detect an IMO in front.

The whole process is complex and cannot be described here in detail, thus the reader is referred to [Pauwels et al., 2010] for further information. Here, we only give la short overview summarized in Fig. 6.5. A nonlinear instantaneous-time model [Zhang and Tomasi, 2002] is used to extract egomotion from the optical flow. To obtain optimal estimates, an iterative minimization procedure

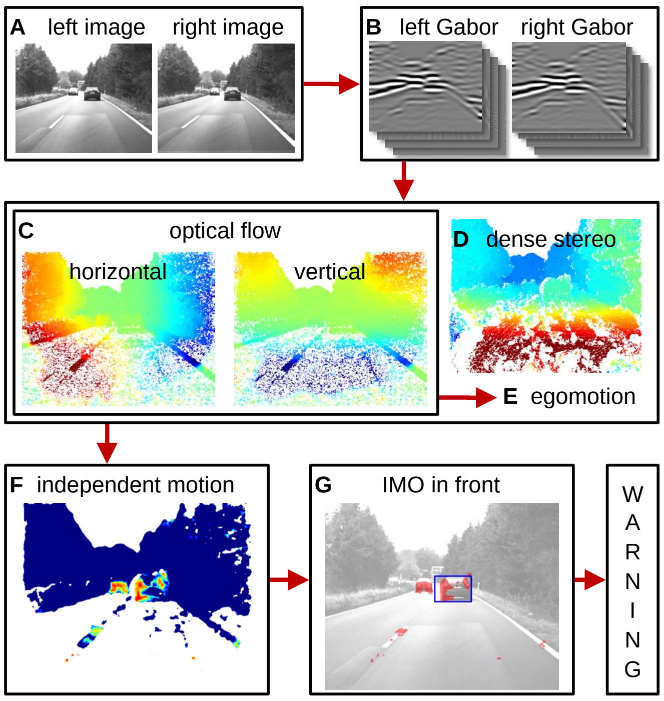


Figure 6.5: Real-time IMO detection. Multi-orientation, multiscale Gabor filter responses (**B**) are extracted from the stereo image pair (**A**) and used to compute dense optical flow (**C**) and stereo disparity (**D**). The horizontal and vertical optical flow is color-coded from -15 (dark red) to +15 pixels (dark blue) and the stereo disparity from -50 (dark red) to +20 (dark blue) pixels. Combined with egomotion (**E**, extracted from the optical flow, not shown) these cues allowe the extraction of independent motion (**F**, likelihood increases from blue to red). This independent motion signal was gathered in a fixed region of the image (**G**) and when it exceeds a threshold, a warning is issued.

is used that relies on M-estimation [Mosteller and Tukey, 1977] for outlier compensation. A total of 32 different initializations are explored to deal with local minima. The data-intensive parts of the algorithm run entirely on the GPU. Independent motion is detected by evaluating the depth/flow constraint [Thompson and Pong, 1990] at each pixel. Deviations from this constraint point to inconsistencies between the optical flow, disparity and egomotion and result from noise or independent motion. The deviations are assigned to independent motion if they comply with a 3D translation model in a local region surrounding the pixel (Fig. 6.5F). To detect a moving vehicle in front, the (pixelwise) independent motion signal is accumulated inside a fixed region in the image (blue rectangle in Fig. 6.5G). A warning is issued when more than 30% of the pixels within this box are considered independently moving.

6.4.5 Lane Detection and Filtering

The lane detection we use for the DRIVSCO system is the same described in section 4.2. Since it has certain shortcomings concerning outdoor use, because it does not provide means for handling occlusions, we implemented another method published for outdoor lane detection, based on a detection called CHEVP and a subsequent B-Snake based lane tracking, published in [Wang et al., 2003]. We tested this on recorded car data and found that although these methods were claimed to have several advantages, e.g. to work with versatile types of streets, even unmarked ones, able to handle occlusions and different illuminations, they were not usable for the desired setup. It was simply too slow, sometimes taking up to several seconds to process a single image. Also it did not work robustly enough in high curvature turns, a video ("streetDetectionSplines.avi") is contained on the enclosed cd, or available online under www.markelic.de/professional/Documents/. From this aspect, our indoor lane detection outperformed this method, however with the constraints that lane markers must be clearly visible in the images, and occlusions are neglected.

To correct the lane detection against false positives, instead of using the smoothing routine reported in section 4.2.2.6, an additional filter is applied (provided by the project partners) which uses 3D information to verify if the detected lane is in the ground plane. This is achieved by using the computed dense stereo map (see section 6.4.3) which attaches 3D information to each point of the extracted lane in form of disparity values. There is a linear relationship between disparity values and horizontal image coordinates and the filter checks whether the extracted lane fulfills this criterion. Since the disparity contains noise, RANSAC [Fischler and Bolles, 1981] was used to fit a line to the disparity data of the extracted lane. If an estimate with slope in a tolerable range can be fitted, it is believed that the lane is in the ground plane, otherwise it is rejected.

As indicated by the entry "generate lane expectation" in Fig. 6.3(d) a final feedback mechanism aids the stability of the lane detection by generating lane expectations based on the human behavior. Note, not a learned forward model is used as presented in chapter 5 but an analytical one. The velocity and steering angles of the car are used to derive its Rigid Body Motion (RBM), which is then used to predict the position of a detected lane one frame ahead.

The expected lane is then compared to the detected one in the following frame, and if both differ too much (a threshold is used) the detection is rejected.

6.4.6 The DRIVSCO Perception-Action Repository

The PAR used for the DRIVSCO system is based on the same mechanisms as described in section 4.3.2. The information stored in the PAR include both lane markers (left and right), and instead of speed acceleration (obtained from the car CAN-bus, see section 6.4.1) is predicted. This is because the speed data is found to often contain a linear trend and in general a high variance. Acceleration is thus better predictable. For generalization, we use the AVG method as explained in 4.3.2.4, and we do not use a reactive component for fine-tuning in case of the real car. The biggest difference is that the DRIVSCO PAR is trained on real driving data. The rest of this section was added for completeness and to document the details concerning the DRIVSCO learning procedure.(A video showing the original current input image on the left and the retrieved match for a real drive on the right is on the enclosed cd 'analysisDB.avi' or online under http://www.markelic.de/professional/publications.html following the link 'driving with a database'.)

6.4.6.1 Data, Preprocessing, and Default Driving

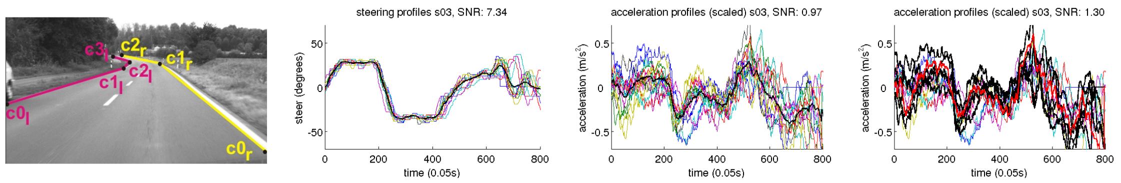
The driving data stemmed from a single driver and was provided by the project partner Hella KGaA Hueck & Co[2]. It consists of three repeatedly driven tours which are referred to as s03, s05 and s06, compare Fig. 6.2(a)-(c), and a track denoted “long tour”, see Fig. 6.2(d), which was driven twice.

The goal is to learn the default human behavior concerning steering and acceleration for lane following, i.e. in context free situations. By this, driving without additional traffic and without special situations, like intersections, etc. is meant. This requires filtering the original driving data, which was not necessary for the previously presented robot setup.

This can be achieved by using the IMO detector to identify and exclude situations with IMOs in front as well as by using inconsistencies in the lane detection to identify and exclude situations like intersections. However, since the IMO detector was not available in due time for this work the filtering was done manually. Hence, PAR entries for lane following are based on context-free driving examples.

In addition to the acquisition of context-free situations we also remove action sequences that are obvious outliers as described in the following: In Fig. 6.6(b) and (c) fifteen steering and acceleration signals from the same track and driver are shown along with their means plotted in black. We observe a much higher variance in the acceleration data than for steering which is quantified by the mean signal to noise ratio, SNR (we compute $E[\frac{|\mu|}{\sigma}]$, with μ being the mean and σ the standard deviation), where a high SNR indicates good and a low one bad predictability of a signal. For the

[2]Parts of this set are available at the web-page [Eisat, 2010].



(a) (b) (c) (d)

Figure 6.6: (a) The extracted street boundaries described by polylines and the corner points. The vectors of the corner points $(c_{0_{\mathrm{l/r}}}, c_{1_{l/r}} \ldots)$ for left and right lane constitute the visual state description. (b) Steering and (c) acceleration signals from fifteen runs from the same track and driver. The mean is plotted in black. "SNR" refers to signal to noise ratio. (d) The sequence closest to the mean is shown in red. Black signals are considered to be sufficiently similar.

shown data we obtain an SNR value for steering of 7.43 and for acceleration 0.62. By removing outliers (which are detected by an automatic procedure during which the individual signals are compared to the mean) the latter can be increased to 1.3. In Fig. 6.6(d) black signals are those acceleration signals found to be sufficiently similar and others (differently colored plots) are the ones that were sorted out. The sequence closest to the mean is plotted in red. This data is what we use for training the PAR.

Driving data assigned to similar situations is averaged during training. Similarity is defined by the resemblance between left and right street lane of the observed image and one already contained in the PAR as well as a short sequence of previous steering data. A formal definition is given in section 4.3.2.2. Hence, for tracks that are being driven multiple times the system defines the default human driving behavior over the mean of the action sequences obtained from driving on the same track as exemplary shown by the black plot in Fig. 6.6(b). Note, however, learning starts with the first entry in the PAR and every additional information just improves the performance of the perception-action mapping. Single-example performance is reliable for steering, but remains restricted for acceleration control, see section 6.6. Note, as curve shapes are fairly similar, after some learning the system is able to generalize into unseen curves. This will be shown later (see Fig. 6.9).

Since the signals predicted by the PAR should correspond to the default behavior of the human, i.e. the mean action-signal, we evaluate the performance of the PAR by comparing its output against the human action sequence closest to the mean signal, i.e. the red plot in Fig. 6.6(d), (which we do not include in the training data.)

6.4.6.2 Sense-Act Pairs

The state vector, $\mathbf{s}$, for the DRIVSCO PAR is

$$\mathbf{s} = \{\mathbf{v}_{\text{left}}, \mathbf{v}_{\text{right}}, \mathbf{st}_{\text{past}}\}, \qquad \text{state vector.} \tag{6.1}$$

with:

$$\mathbf{v}_{\text{left}} = [c_{0_l}, c_{1_l}, \ldots, c_{l_l}], \qquad \text{left lane,} \tag{6.2}$$

$$\mathbf{v}_{\text{right}} = [c_{0_r}, c_{1_r}, \ldots, c_{l_r}], \qquad \text{right lane,} \tag{6.3}$$

$$\mathbf{st}_{\text{past}} = [st_{t-1}, st_{t-2}, \ldots, st_{t-m}], \qquad \text{preceeded steering.} \tag{6.4}$$

Here, l_l and l_r denote the lengths of $\mathbf{v}_{\text{left}}$ and $\mathbf{v}_{\text{right}}$. Their entries $c_{0_{l/r}}$, $c_{1_{l/r}}$, ... are as indicated in Fig. 6.6(d). Again, $\mathbf{st}_{\text{past}}$ is a sequence of discrete steering actions that preceded the image frame from which the lanes are extracted, and its length m is set to 50 (corresponding to 2.5s of driving), and t denotes time. st refer to discrete steering actions. Thus, a PAR entry, $\mathbf{e}$, for the DRIVSCO system is as given in Eq. 6.5.

$$\mathbf{e} = \{\mathbf{s}, \mathbf{st}_{\text{fut}}, \mathbf{a}_{\text{fut}}\} \qquad \text{PAR entry} \tag{6.5}$$

The action sequences $\mathbf{st}_{\text{fut}}$ and $\mathbf{a}_{\text{fut}}$ are:

$$\mathbf{st}_{\text{fut}} = [st_t, st_{t+1}, \ldots, st_{t+n}], \qquad \text{subsequent steering,} \tag{6.6}$$

$$\mathbf{a}_{\text{fut}} = [a_t, a_{t+1}, \ldots, a_{t+n}], \qquad \text{subsequent acceleration,} \tag{6.7}$$

with st and a denoting single steering and acceleration signal values (actions)[3]. The lengths of these sequences are again supposed to resemble the number of actions necessary to cover the part of the street observable in the observed image. Since it is not known exactly to how many actions this corresponded, a fixed value, $n = 100$, is used which is 5 seconds of driving corresponding to 97m at a speed of 70km/h which is considered reasonable for country road driving.

6.4.6.3 Training and Retrieval

The training, i.e. the filling of the PAR with sense-act pairs is analogous to the training described in section 4.3.2.2, only that the left lane description is considered, too. The same holds for the retrieval step which is explained in section 4.3.2.3. Note, training is done off-line (hence, not during driving). This would be desired also in any commercial system as the training procedure was demanding (due to the removal of context-dependent scenes) and should thus take place, when the car is not operating.

6.4.6.4 Prediction of Action Sequences

The prediction of action sequences, the final output of the PAR, is achieved by using the AVG method, introduced in section 4.3.2.4. For a final smoothing a moving average filter (window-size=10) is applied to the resulting signal.

[3]The discretization is according to the camera frame rate of 20 Hz.

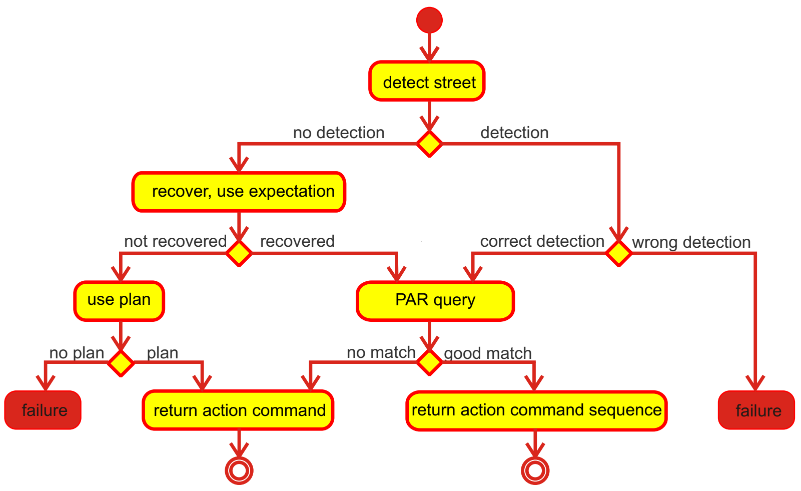


Figure 6.7: Algorithmic flow of the driving system (akin a UML activity diagram).

6.4.6.5 Warning

We implemented a simple warning mechanism in the system to demonstrate its use for driver assistance. A warning is generated if the actual human steering deviates too much from the expected driving given by the computed prediction and specified by a threshold determined empirically based on the off-line analysis of the test-drive sequences. Fig. 6.4 exemplary shows the actual driving, the prediction and the thresholds.

6.4.7 Algorithmic Flow

The algorithmic flow of the system concerning action prediction is summarized in Fig. 6.7.

No output can be generated if lanes can repeatedly not be detected, or if they were repeatedly mis-detected, i.e. other items erroneously identified as lane. Some critical cases can be compensated by other mechanisms, e.g. the generated lane expectation can be used instead of an undetected lane. If this prediction is considered unreliable the previous action plan could be used for as long as it contains entries. In this case the control is open loop and only single action commands can be issued. In the optimal case the system returns a sequence of action commands.

6.5 Results

The performance of the IMO detection is reported in detail in [Pauwels et al., 2010], thus here it is focused on the learning system.

The developed lane detection algorithm was evaluated on each tour of the available data set which comprised a variety of country roads with both clear and damaged lane markers (a total of 11.400 frames with humanly detectable lane markers present in 11.055 frames). In 98.9% of the

11.055 frames a valid lane description was extracted. In 5.6% of these cases only the left and in 8.5% only the right marker was detected. Both markers were found in 84.8% of these cases.

To evaluate how well the action predictions match the human default behavior we use the provided data set.

After filtering it, as explained in section 6.4.6.1, approximately 80 min. of training data were obtained, resulting in a PAR with 90,470 entries, adequate for evaluating the system performance.

First, we test the performance on a known track, i.e. one that the system had seen before. For that we train the PAR with all runs but the one closest to the mean which we consider to resemble the human default behavior as explained in section 6.4.6.1 and which we use for testing. The smoothed steering and acceleration signals from the algorithm are plotted against the signal generated by the human for s03, s05, and s06 in Fig. 6.8.

As an additional measure of similarity we compute the correlation coefficient of the two signals (human and prediction). For steering and acceleration prediction we obtain for s03 0.99 and 0.81, for s05 0.93 and 0.73, and for s06 0.97 and 0.67. Thus, all predictions are very good, however the steering signal is better approximated than acceleration, which is as expected from the computed SNR in section 6.4.6.1.

For testing the performance on an unknown track we train the PAR with all available data except that from the track we test for. The result is shown in Fig. 6.9. We chose s05, which contains a very sharp turn (see Fig. 6.10) leading to two typical phenomena discussed below. The resulting steer and acceleration predictions in comparison to the human signal are shown in Fig. 6.9. The steering prediction is very close to the human signal, but the acceleration is less reliable. In particular it can be seen from the plotted acceleration prediction in Fig. 6.9 that the sharp curve (which is entered around time step 200, compare Fig. 6.10(a), is accompanied by a deceleration in the human signal (between time step 220 and 380) and that this is reflected nicely by the algorithm. However, before and after the deceleration part the predicted signal differs considerably from the human data.

This phenomenon results from the fact that street parts with a lower curvature allow a great variance in the driver's choice of speed depending on hard-to-access variables including "mood" and "intention", etc. where curves, especially sharp curves, significantly restrict the driver's behavior and thus make it better predictable. We therefore conclude that acceleration prediction with the presented method is only useful for curve negotiation. The second observation is that there are unwanted peaks in the predicted acceleration curve. This happens because the system sometimes hooks on to a different, similar looking road segments in the PAR, as can be seen from Fig. 6.10(b). Although the lanes are correctly extracted the look-ahead is not sufficient to distinguish this situation properly from almost straight streets. We found that our driver sometimes reacted to upcoming curves up to 8 seconds before entering them, requiring the system to have a similar look-ahead to correctly predict the driver. Humans can see this far and make out even very faint features of the road layout, which leads to such early reactions, computer vision systems, however, cannot. The detected lanes at such distances and the detected lane segments will, for smooth and less descriptive

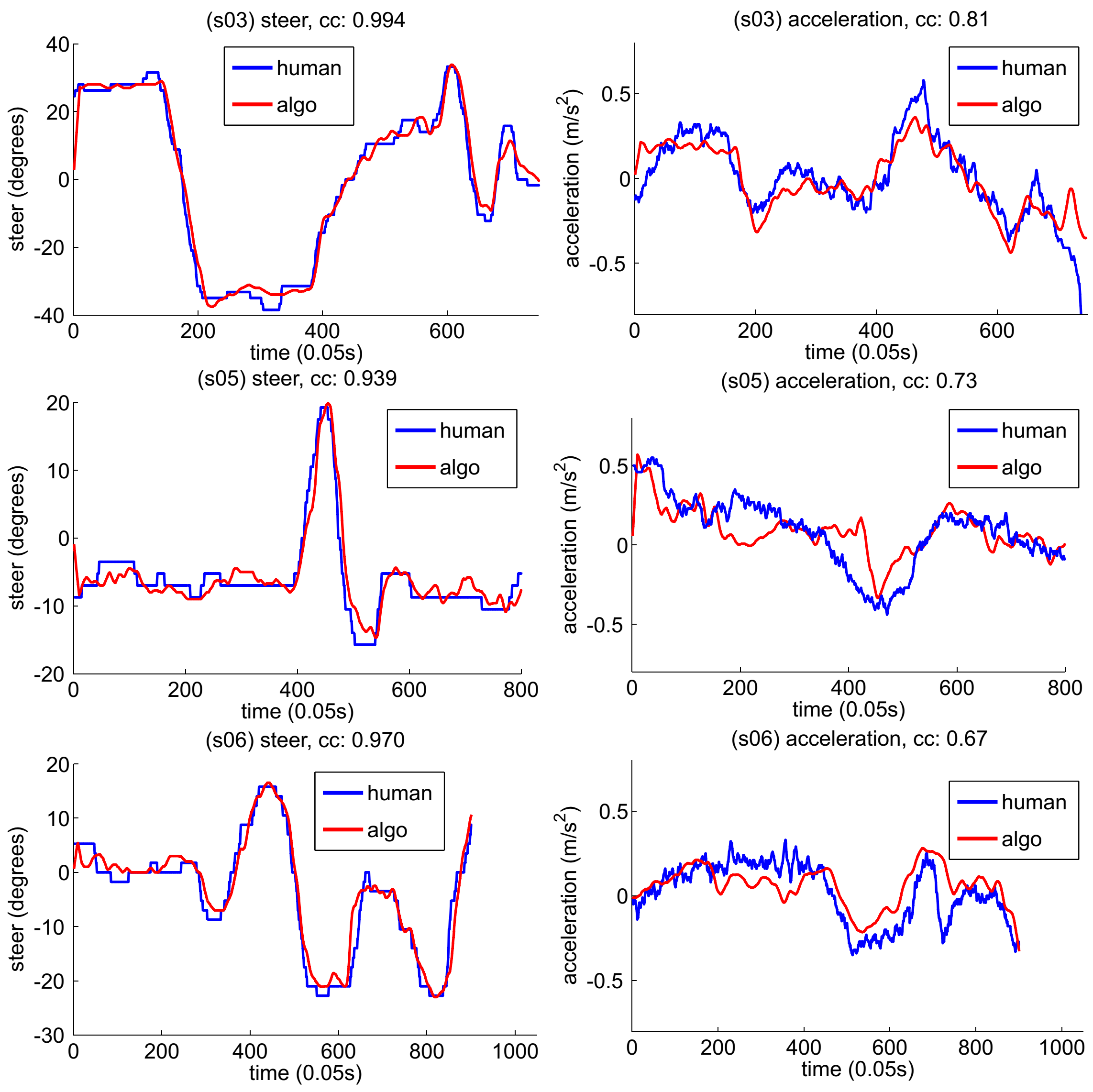
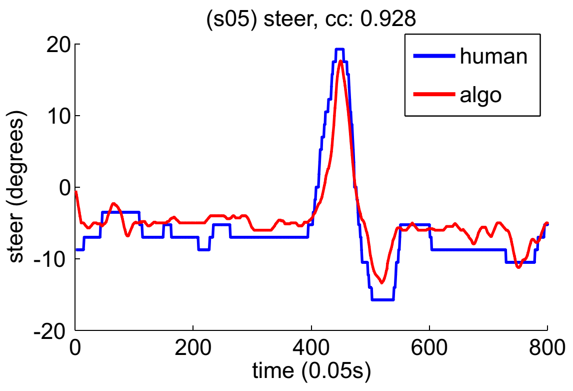


Figure 6.8: Results for steering (left column) and acceleration (right column) prediction for known tracks. "cc" denotes the correlation coefficient value between the human generated signal and the synthesized one.

situations, remain ambiguous leading to false matches which causes these peaks.

According to the algorithmic flow shown in Fig. 6.7 critical cases are a frequently undetected street, as well as the case that the PAR did not contain a good enough match. In these cases it is possible to work off a previously generated action plan, but only as long such a plan is available. For the presented tests the street detection rates were high: for s03 100%, for s05 96%, and for s06 98%. However, the case that no PAR match was retrieved occurred relatively frequently: for s03 in 32%, for s05 in 12%, and for s06 in 39% of all cases. For testing the case of driving on an



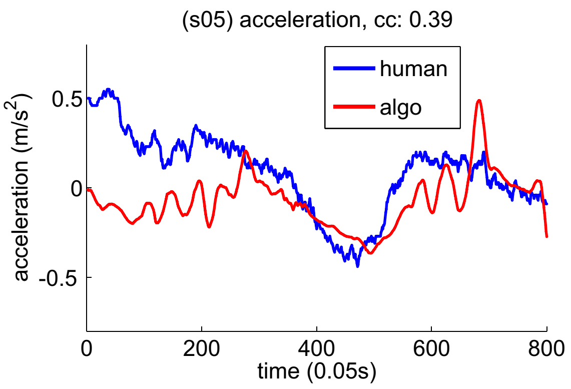
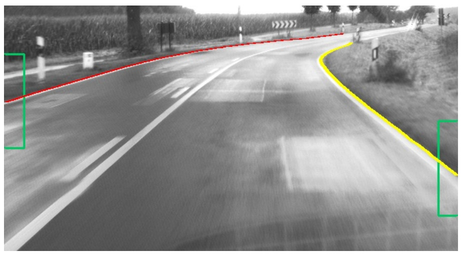
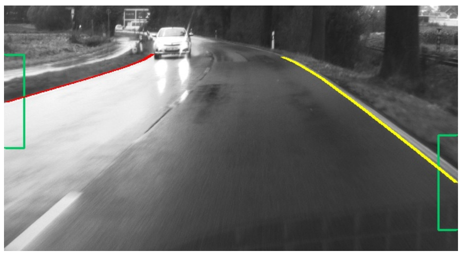


Figure 6.9: Results for steering (left) and acceleration (right) prediction on an unknown track.



(a)



(b)

Figure 6.10: (a) Entering a sharp turn (in s05) at t = 200. (b) Too short detected lanes.

unknown street on s05 no PAR match was retrieved in 39%. Despite these high rates it can be seen from the Figs. 6.8 and 6.9, that for the entire duration action signals were reliably produced. It is an important aspect of the system that its ability to predict sequences adds considerable robustness to its performance.

During the final review meeting of the DRIVSCO project the system was demonstrated successfully observed by three independent international reviewers, see EU-Newsletter [2010]. The driver from which the training data was obtained, i.e. who taught the system, drove the first 20 minutes of the long tour and back, where “back” corresponds to an unknown track situation as curves are reverted. All components were shown to work reliably together at the desired speed, the lane detection worked even under challenging weather conditions where sunshine after rain caused reflections. (A TV report about the DRIVSCO system is available on-line [NDR, 2009].)

6.6 Discussion and Conclusion

We have presented the DRIVSCO system, which learns basic human driving. Its imitation performance on known and unknown track(s) was evaluated and shown to work reliably for steering and in - in the case of curve negotiation - for acceleration prediction, too. In addition it contained a novel form of data driven IMO detection. Thus both, the learning algorithm as well as the IMO detection, did not rely on predefined models which made the system widely applicable.

We demonstrated the use of the system output (steering and IMO detection) for supporting the driver, where this system may have future potential, namely as intelligent driver assistance system automatically adapting to individual driving behavior.

In contrast to most related work DRIVSCO is an integrated system implemented on a real car, in real-time, realized by a multi-threaded, parallel CPU/GPU architecture which uses data from real driving – not simulation. The learning algorithm is deliberately simple and thus easy to understand and implement. In the early stages of the project different methods based on feed-forward and radial-basis-function networks were tested, however not achieving the performance of the lazy learning approach presented here. Furthermore, lazy learning offers the advantage that all information remains human interpretable, which is convenient for error analysis as opposed to learning algorithms which transform information into subsymbolic knowledge which is, for example, the case with neural networks. The system predicted *sequences* of expected human behavior as opposed to a moment-to-moment control. This made it a) more stable, e.g. in case of unreliable or lacking sensory input it could use predictions as a fall-back plan, and b) allowed for proactive control, i.e. warnings could be issued based on the predicted behavior instead of the current one.

The system has been specifically designed for motor-ways and country roads, hence driving situations with limited context. To apply imitation learning to more difficult driving situations (e.g. city) appears currently infeasible as driving decisions are in these cases far too diverse and state-action descriptions would become too complex. Furthermore, we observed that acceleration signals are very non-descriptive when driving in uncritical situations (e.g. straight road) because drivers followed their mood. This contributes strongly to the high variance observed in the acceleration data. As a consequence longitudinal control (or warning) becomes only useful whenever a driver is forced to drive with less leeway (e.g. in front of sharp curves). This notion is important when considering the psychological acceptance of individualized driving aids. One of their central features must be to not unnecessarily interfere with the driver. Hence, in uncritical situations systems should remain silent and acceleration should remain in the hands of the driver.

To improve the presented system one should furthermore consider to extend the system's look-ahead, beyond that of with machine vision. This can be achieved, for example, by integrating GPS information and digital maps.

One interesting aspect concerning industrial applications is the potential use of this system for night driving support. Under bad illumination conditions the human's sensing process is obviously limited, however, by using infrared light the system's sensing is less affected, given that the lane detection process is adequately adapted to the new circumstances. Thus, the system can use its knowledge about driving acquired during the day to support the human in the more difficult situation of night driving.

Chapter 7

Discussion

"The difficulty lies, not in the new ideas, but in escaping the old ones, which ramify, for those brought up as most of us have been, into every corner of our minds."
John M. Keynes

The main chapters of this thesis are each accompanied by their own discussion section and the reader is referred to these for detailed aspects of the particular topic of a chapter. In this section we discuss the presented work in a broader context with respect to the stated hypothesis in section 1.5.

First, we summarize the main parts of this thesis and we finish by pointing out to future work.

7.1 Summary

By means of the practical application of learning anticipatory driving from a human supervisor this work investigated the hypothesis that principles of human skill acquisition together with economic pressure may lead to alternative methods for system control.

A review of human skill learning as far as relevant for this work was given, containing the important distinction between early and late learning phase and we proposed the use of the presented chunking mechanism for the given task. We also gave a short overview on important approaches to autonomous driving and presented other existing systems of which we described three in detail as representatives of a certain followed paradigm (control theoretic, machine-learning based, and a hybrid approach).

The proposed chunking approach was realized for the robot by constructing a database using human training data and sensory input extracted by a dedicated lane marker extraction method which we developed. The resulting system was shown to successfully capture the driving behavior concerning steering and speed control and to be able to produce action plans while making little system requirements.

We tested whether the mainly reactive system we designed could be augmented by means of forward models which are suggested to be acquired during the early phase of skill learning and to form part of the basis for higher cognitive skills. This work was conducted in the laboratory only. We found that a simple internal representation could be acquired which improved the system behavior. The predictive capacities by coupling the forward model with a controller (the resulting plans are sequences of state-act pairs and we denoted them planCs, compare section 5) were found limited in comparison to those of the chunking approach which we called planPs (sequences of actions, also compare section 5). Therefore, we concluded that biological systems either have the means to handle increasing uncertainties, or they do not and instead use only short-term predictions on this low sense-act level but compile information into higher-level representations. The latter is conform with the suggestion that chunks are hierarchically organized.

Finally, we transferred the chunking approach to the real car for which we also used our (slightly modified) lane marker detection method since an implemented alternative method from the literature could not be used. The resulting steering control was good but the acceleration prediction limited, mainly due to missing look-ahead which was difficult to achieve by using vision only and comparably little data.

The resulting system (robot and car) differs from others in that it a) does not require metric information while b) augmenting a reactive system with the capacity to generate action plans. These are in the form of action sequences (planPs). Furthermore, we did not make use of simulations like much other work in the field but worked with either a robot or the real car. Thus, the system was forced to cope with challenges like real-time requirements and noise at all stages.

7.2 Discussion

According to our hypothesis we wanted to study in how far mechanisms underlying skill acquisition can be used for developing alternative approaches to system control. In addition, we postulated the consideration of economical pressure which we argued to be important for shaping efficient processing mechanisms. We suggested that taking this approach (mechanisms underlying skill acquisition while taking an economical stance and no simulations) may be beneficial in three aspects: It should lead to the acquisition of a real-world working system, it may enable us to draw conclusions concerning the biological model, and it may lead to alternative system control. As summarized in the previous section, all this could be achieved to varying degrees. The main result is a driving system which differs much from other reported approaches in that it is not conventionally programmed, but learns from provided examples and requires very little given knowledge apart from the provided dedicated street detector. The most important distinction however is that our system is able to produce action plans without requiring metric information. (This also answers the question we stated in section 1.4 whether this is possible.) It is also the main difference from the described ALVINN system (see section 2.2.3) which is similar to ours in that aspect that it is also a black-box approach and does not require metric information either. However, ALVINN generates reactive moment-to-moment control (steer only) where here we produce the said action plans (steer and speed/acceleration). To the author's knowledge this is a unique achievement so far not found in the literature on autonomous driving and therefore confirms our chosen approach.

We mostly used very simple methods, e.g. we saved data in a database to realize the memory of the system, we used straight-forward retrieval mechanisms to realize information retention, and also the lazy learning techniques were mathematically not advanced. All used methods fulfilled their purpose while allowing a good understanding and thus debugging of the resulting control which is very advantageous for the development of a complete system. In addition, it shows that simple means can be sufficient for solving a difficult problem. Now, that the overall idea was shown to work, each module can of course be elaborated or replaced by more sophisticated ones. For example, the memory of the system may be realized more efficiently by a preceding clustering of the input data. However, more important from our point of view is the investigation of problems identified during this work (explained in the following) which are more profound than the instantiation of particular sub-components.

Apart from the positive experiences we made with the taken approach we also encountered shortcomings which we want to critically discuss in the following. There are two particular limitations, one of which concerns the problem of imitation learning and adaptivity and the other obstacle avoidance.

7.2.1 Imitation Learning and Adaptivity

A re-occurring theme in this work was reactive and deliberative behavior. We discussed the distinction between both and the question whether and how reactive behavior can be augmented or transferred into the more advanced deliberative kind, see section 1.4. Although we emphasize that our system can produce action plans, it is important to note that these are still acquired in a reactive fashion and not by reasoning by which we here mean the coupling of internal models, see section 3.2.3. We worked out the distinction between the two types of action plans obtained with both mechanisms which we denoted planCs (sequences of state-action pairs obtained by reasoning) and planPs (sequences of actions obtained with a stimulus-response mechanism), see section 3.2.3, and we found that planCs are richer in information since they include information about states expected to be encountered, but more expensive to achieve and also less robust concerning error propagation, see section 1.4. Concerning the lacking state information in planPs, we argued that they are implicitly given since chunks are only formed after long practice, i.e. after a behavior is already well executed, see section 3.2.3. Where this may be correct and a mapping from states to actions (policy learning, see section 3.3) can be very effective as we showed, lacking the knowledge of the next desired state leads to a serious shortcoming as explained in the following.

Without knowledge about the next desired state the system cannot compare the encountered one with the desired state and consequently not evaluate the conducted action. If it cannot evaluate the action it cannot adjust it either. Thus, without this crucial information the system only mimicks the observed behavior, but does not truly imitate it. By mimicking we mean that it chooses the action that the teacher would chose in a similar situation, but it cannot adapt its behavior to changing situations. For example, after some driving the car's brakes may wear-off. Where a human will realize that the braking pedal must be pressed harder, our system will simply repeat the meanwhile inappropriate action.

In summary, although our system can produce action plans these plans differ from those obtained by reasoning in the sense that they do not contain knowledge about the states which should be encountered. Our system is therefore limited to mimicking because it cannot evaluate its conducted actions. To improve this, i.e. to endow it with the missing information about desired states, literature on developmental research and imitation learning suggests the learning of inverse models, which in turn can be learned from already acquired forward models [Meltzoff, 1988; Meltzoff and Moore, 1997]. Thus, our work in chapter 5, where we were concerned with learning a visual forward model for the detected lane marker may be useful for improving this issue, i.e. to obtain an inverse model which contains a desired next state.

7.2.2 Obstacle Avoidance

Another current limitation we find is that in its current realization the system cannot easily solve the task of obstacle avoidance. To avoid an obstacle in a controlled way, i.e. with the aim to reach a desired location, in this case the street, the system can either plan a path around the obstacle and again to the street or it reactively avoids the obstacle and then decides on a moment-to-moment basis for the right actions. The first option is an example of path planning and the second (related) option an example of navigating and spatial orientation. Both problems appear almost trivial given a 2- or 3D environmental metric map. However, without such a map we face several difficulties.

Concerning the moment-to-moment approach the system must know "where" the street is with respect to itself. The arising question is how "where" is represented. As long as the desired location is directly perceivable this is unproblematic. However, this situation is unrealistic, since the vehicle during the avoidance motion turns away from the street. In mathematics, coordinate systems are used to precisely specify locations. For biological systems place cells were suggested to fulfill this task [O'Keefe and Nadel, 1979]. Locations perceived visually may be encoded by snapshot views [Cartwright and Collett, 1982; Collettand and Collett, 2000] and relations between positions may be encoded by sense-act rules, e.g. by attaching the actions that transform the perception of location A into the perception of location B, [Franz et al., 1998]. An intuitive example would be the expression of the relation (distance) between A and B by the number of steps (and direction) needed to traverse from A to B. This has been shown to lead to complex navigation behavior without the need of metric information, [Franz et al., 1998]. (As a side-remark one may notice the similarity between the idea of encoding paths by means of sense-act pairs to the notation of a forward model. In both cases there is a transition from a given state and an action to a next state.)

Concerning the second option, i.e. the pre-planning and following of an entire path, the system must be able to update the computed plan after the conduction of some action. How this update occurs depends on the representation of the path. It may be given in action space, discretized by fixed-length time slices, in which case it suffices to pop off the topmost action. However, this is difficult to achieve. More realistic is a plan given in sensory space which is translated into action commands on a moment-to-moment basis. In this case the changed path must be updated in sensor coordinates. Traditionally, this can be achieved by computing the resulting motion of the camera (rotation and translation) in 3D-Euclidean space and applying the resulting coordinate transformation to the image with the (possibly imagined) path. This can be used to incrementally obtain a metric map of the environment and in fact is used for vision-based map building and SLAM (simultaneous localization and mapping) algorithms [Thrun et al., 2005, chapter 5]. However, it is a white-box procedure which requires to provide the system with the a priori knowledge which we tried to avoid here. Vision-oriented biological systems also have only access to perspectively distorted information, thus this problem must be treatable by other means, too. As we already saw from the previous paragraph, cognitive maps may be acquired.

Summing up the last two paragraphs, we explained how solving the seemingly simple problem of (controlled) obstacle avoidance with the taken approach quickly lead to the fundamental question of spatial cognition. Where this is a highly relevant research topic it is questionable if learning cognitive maps is an appropriate mechanism for applications like advanced driver assistance systems. On the other hand, there is current research investigating appearance-based SLAM, which is inspired by the idea of cognitive map learning, [Cummins and Newman, 2008], thus, it might be just a matter of time until such methods are elaborated enough that they are attractive for commercial use, too.

7.2.3 Evaluation of the Taken Approach

The DRIVSCO project aimed at a driving system, however, it allowed the investigation of many versatile questions which are addressed below and as such was an ideal testbed.

Traditional approaches to system control have known limitations and alternatives for their improvement are necessary. Humans and many other animals derive much of their performance from the ability of skill acquisition and therefore its investigation is useful for arriving at a control different from known and established concepts as we could show with this work. Related to this idea are biologically inspired reactive robots which were shown to perform amazingly life-like, and while implementing very simple control strategies they were able to overcome serious limitations of traditional AI, [Brooks, 1991]. Where these are already good examples of alternative and successful control, an important problem is their augmentation with the ability of deliberative behavior. Based on our work in chapter 5 we conclude that this may be possible by means of hierarchical chunking and internal model learning.

A difficulty we encountered with the suggested approach is that skills build on other, already available skills or knowledge. For example, spatial orientation may rely on a cognitive map which was obtained earlier. However, the more knowledge or skills that must be learned the more data and time is needed. And even with this available it cannot be guaranteed that all desired functions can easily be acquired since it may simply not be known how. For example, acquiring such fundamental skills like depth perception is an open research issue (e.g. O'Regan and Noë [2001]). It may even be that for a proper grounding a very strict bottom-up approach must be pursued, e.g. started with tactile sensors before moving towards vision. This idea is highly interesting for basic research in robotics and developmental sciences. For example one major European project, RoboCub, investigates such a bottom-up approach by means of a robot resembling a three year-old child, [Nosengo, 2009]. However, for commercial systems bottom-up approaches like this do not make much sense. In addition, if the capacity of a robot depends on its experience, the question arises how experience can be transferred such that not every robot must undergo the same learning process.

On the other hand, it may be possible to decouple sufficiently well-understood modules that originated from such research. Their application scope may be limited, however within this scope they may still work very efficiently, such that they can be used for particular purposes. For example, the chunking approach we presented may not be usable for obstacle avoidance, but it was shown to work very well for steering sequence prediction. And this particular task was solved efficiently, too. It was independent from the used system, which was shown by the ease with which we could transfer it from the robot to the car which differ very much in their technical realization, for example, the robot has a differential drive whereas the car has movable axles. The method made little system requirements (a single uncalibrated, undistorted, gray value camera) and did not require the programming of a white-box solution Therefore, we conclude that the method is very general, versatile, and cheap and thus interesting for commercial applications, even more so, if the problems regarding speed control for the real car can be improved since this is something which does not exist on the car market yet (see also section 6).

In summary, whereas not all of aspects of our suggested approach may be usable for commercial systems, we consider its pursuit as advantageous and promising, since it allows us to gain knowledge not only concerning alternative robot control, but also the identity of human beings and other animals.

7.3 Outlook and Future Work

The questions we suggest for further investigation are motivated by the described limitations of this work.

We stated that extracting the intention of an observed action is crucial for true imitation learning (in contrast to mimicking) to achieve adaptive and deliberative behavior. It is also useful for developing systems which can safely and efficiently interact with humans. To our best knowledge only few approaches to this relatively new topic exist ([Ziebart et al., 2008; Ramachandran and Amir, 2007; Ng and Russel, 2000; Abbeel and Ng, 2004; Meltzoff and Moore, 1997]) and therefore we suggest it for future research.

As stated above (see section 7.2.2), intentions may be acquired by inverse models which in turn can be obtained by forward models. Apart from its use for intention learning, visual forward models are also required for tracking algorithms which improve the system perception and thus also the performance. Therefore, learning (visual) forward models possibly without the necessity of undoing the perspective effect is another important topic which we already started to tackle in chapter 5.

The next problem we suggest concerns the fundamental problem of extending reactive systems with deliberative behavior which was considered to be realizable by means of acquiring internal representations in form of forward models. These predictors should be as accurate as possible and as we argued before may be improved by hierarchical chunking. Thus, the automatic formation and acquisition of hierarchical chunks and also their composition should be studied.

Finally, we propose to further investigate the acquisition of visual depth perception. This is a fundamental skill required for many visuo-motor applications, for example reaching and grasping. Many automated vision systems consist of fixed cameras which compute depth from stereo, which is a traditional white-box approach. In accordance with Gibson's view (Gibson [1979]) we propose that learning and understanding depth (and thus better performing systems) should be studied with systems that have the ability to move. This way it can be investigated if experience grounded in sense-act rules indeed allows to arrive at a usable representation of depth. An interesting work related to this topic was done by Hoffmann [2007] who let a robot learn to judge if it would pass through a visually perceived gap by means of mental simulations only. For that no information other than the robot's own experience was required, in particular no metric information, e.g. the size of the gap or the width of the robot, indicating the power of this "enacting" approach.

Appendix A

Glossary

A.1 Action

An action can alter the environment of an agent and/or its state. In classical AI actions correspond to the notion of operators, and in control theory and robotics actions correspond to inputs or controls (often abbreviated as *u*).

A.2 Agent

The learner or decision-maker [Sutton and Barto, 1998].
"[. . .] an umbrella term if no distinction between humans, animals, and robots is intended" [Pfeifer and Scheier, 1999, p. 646]. The control-theoretic counterpart is the plant, compare A.13.

A.3 Behavior

"What an autonomous agent is observed doing. Always the result of an interaction of an agent with its environment" [Pfeifer and Scheier, 1999, p. 646].

A.4 Control / Controller

A controller realizes the desired response of a system according to some input [Dorf and Bishop, 2007].

A.5 Control Law

The rules implemented by the controller.

A.6 Dynamic Model (Robotics)

A dynamic model describes the motion of an object considering the causing force (compare also kinematic model A.11). For example, a mathematical pendulum that is described by its deflection angle θ, can be described by a dynamic model which is derived from the equation of motion given below (by solving for $\theta(t)$):

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{g}{l}\sin\theta = 0, \tag{A.1}$$

with t indicating time, l the length of string, and g acceleration due to gravity.

A.7 Forward Model (Robotics)

A mechanism which takes a current state and an action to predict the subsequent state.

A.8 Internal Models

Consisting of forward- and inverse models and suggested to play an important role in motion control [Wolpert et al., 1995].

A.9 Internal Representation

It was suggested that deliberative behavior in animals is based on internal representations [Clark and Grush, 1999; Grush, 2004; Hesslow, 2002, 1994] which are considered to be mappings of experienced entities or processes into forward models.

A.10 Inverse Model

A mechanism which takes a current state and a desired next state and predicts the action which (and possibly the probability with which it) leads from the current to the desired next state.

A.11 Kinematic Model (Robotics)

A kinematic model describes the motion of an object without considering the force that caused that motion. Usually this involves the computation of positions or angles, using velocities and accelerations (compare also dynamic model A.6).

A.12 Planning

“[...] any computational process that takes a model as input and produces or improves a policy for interacting with the modeled environment” [Sutton and Barto, 1998, p. 228]. Where the environment is anything which is not the agent. And a model of the environment is “anything that an agent can use to predict how the environment will respond to its actions” [Sutton and Barto, 1998, p. 227].

A.13 Plant

A system or process to be controlled. The control theoretic equivalent to the notion of an agent as used in AI and machine learning, compare A.2.

A.14 Policy

A term used in reinforcement learning to describe a mapping from states to actions, or to action probabilities, usually with the goal to achieve a desired behavior. In this sense it is equivalent to the notion of a controller.

A.15 State

The state of a system is defined by a state vector, which contains the minimal set of independent variables describing the properties of interest of the modeled system.

A.16 State Estimation

The process of determining the state of a system based on given observations. Usually based on Markov chain models and Bayesian reasoning. State estimation is important for optimal filtering, predictions and feedback control. Famous examples are the Kalman- and the Particle-filter,

Appendix B

Method Details

Here, we give detailed descriptions on several methods used in this work ordered according to their occurrence in the text.

B.1 Artificial Neural Networks

Artificial neural networks can be used as general function approximators and as such are very useful for (nonlinear) system identification.

Inspired by biological neural networks they consist of single computing units, the (artificial) **neurons**, which take in information, typically one or several numerical values and pass this information through their **transfer-** or **activation function**. Thus processed, the information may be passed on as input to further neurons in which case there is said to be a **connection**. Hence, connections between neurons resemble the synapses in biological networks. Each connection has an assigned weight which is the analogon to the strength of a synapse. One possible taxonomy of networks is according to their topology. Herein, the probably most prominent network class is the **multilayer perceptron**, MLP. MLPs are networks which are organized in layers of neurons, with an **input** and an **output layer**, and possibly also intermediate layers which are called **hidden**, an example is shown in Fig. B.1. Such a network in which information is passed in one direction only, from the input to the output layer is called **feed-forward**. (Networks with self-connecting neurons are called **recurrent**.)

ANNs can be used for regression or classification tasks. The former describes the learning of a function (function fitting) from given examples, and the latter the learning of a categorization of given input data. A regression task can sometimes be converted into a classification task by interpreting each possible output value as a separate class. This requires as many output neurons as there are possible output values. For example, let $f(x) \mapsto [2, 5, 8]$ be the function to be learned: For a regression task one output neuron is required which can take on all the values of the target

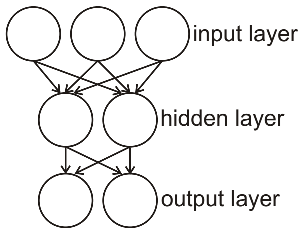


Figure B.1: Example of a three-layer MLP. Circles resemble neurons and arcs synapses, i.e. connections.

set, i.e. 2, 5, and 8. Formulated as classification task three output neurons are necessary each of which represents one entry of the target set as “class”. The activation of the first output neuron may indicate the “class” 2 of the target set, the second output neuron the “class” 5, and of the third the “class” 8.

ANNs can be trained such that a given input results in a desired output, where in- and output are usually given as vectors. Depending on the type of used network several training algorithms are available, e.g. the famous backpropagation algorithm for feed-forward networks Rumelhart et al. [1986], or its improved version, the resilient-propagation algorithm, RPROP [Riedmiller and Braun, 1993]. Common to all training algorithms is to adjust the connection weights such that the error of the network is minimized, where the error E is determined as given in Eq. B.1.

$$E = \frac{1}{2}\sum_{x \in D}(f(x) - \hat{f}(x))^2, \tag{B.1}$$

where D is the set of training samples Mitchell [1997][compare p.237].

MLPs are examples of eager-learning methods (see section 3.3.2), i.e. during training they approximate globally over the given samples, in contrast to radial basis functions, see section B.2.

Further information on ANNs can be found e.g. in Mitchell [1997, chapter 4, p. 81] or Nelles [2002, chapter 11, p. 239].

B.2 Radial Basis Function Networks

Radial basis function networks, RBFs, can be used for interpolation between given (multidimensional) samples. Thus, an RBF can be used to locally estimate the function-value $f(x)$ given the input x, based on l training samples of the form $(f(x_i), x_i)$, whith $1 \leq i \leq l$.

Each neuron of an RBF network is assigned a vector x_u of the same length as the input vectors accepted by the network, where u denotes the u’th neuron. With this an RBFs local estimate $\hat{f}(x)$

of a given input vector x of length n is as given in Eq. B.2.

$$\hat{f}(x) = \omega_0 + \sum_{u=1}^{k} \omega_u K_u(\delta(x_u, x)), \tag{B.2}$$

where k is the total number of neurons, and δ being typically an Euclidean distance-function as given in Eq. B.3

$$\delta(x_u, x) = \sqrt{\sum_{r=1}^{n} (x_{u_r} - x_r)^2}, \tag{B.3}$$

and K being the kernel-function, i.e. the RBF neuron's transfer function which is normally defined by a Gaussian function as given in Eq. B.4.

$$K(\delta(x_u, x)) = exp\frac{-\delta^2(x_u, x)}{2\sigma^2}. \tag{B.4}$$

The offset ω_0 is the bias an adjustable parameter.Thus, an RBF network works in three stages, first the distance of the input vector x to all RBF neurons is computed, and second each neuron's output by passing the previously computed distance $(\delta(x_u, x)$ through the neuron's transfer function K. The third stage is the computation of the final output which is realized as a linear combination of the individual neuron outputs and a weight attached to each neuron output, ω_u, plus the bias ω_0. Hence, each neuron of an RBF acts as a similarity detector that gives an output of 1 or close to 1 if the given input vector x is the same or similar to the neuron (i.e. the neuron's assigned vector x_u). The more similar a neuron is to the given input the larger its output, i.e. the closer to 1, and thus the greater its influence on the final output value.

The variance used for the kernel function determines the selectivity of the network. E.g. if the variance is low (close to zero) only very similar neurons will be activated, i.e. giving an output close to 1. The higher the value of the variance the more neurons will be activated and thus involved in computing the final output. This usually allows for a better generalization performance of the network.

The training of an RBF network consists of two stages: determining the number of neurons and then determining the appropriate weights such that the desired output is obtained. There are various methods for that, compare Mitchell [1997, p. 242], or Nelles [2002, p. 269-279].

RBFs are examples of lazy learning methods, i.e. they generalize locally during retrieval time while keeping the given samples, in contrast to other types of ANNs B.1.

Further information on RBFs can be found e.g. in Mitchell [1997, chapter 8, p. 238] or Nelles [2002, chapter 11, p. 264].

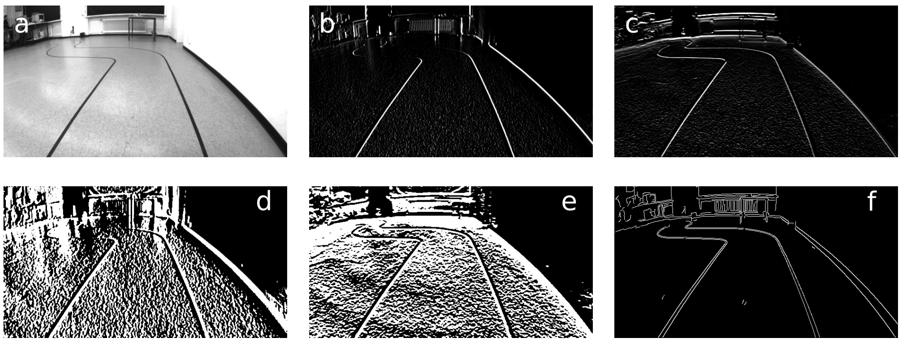


Figure B.2: (a) Original image. (b) and (c) horizontal and vertical derivative obtained with the given 3×3 kernels k_x and k_y from Eq. B.6. (d) and (e): horizontal and vertical derivative obtained with a kernel of size 7×7. (f) The image after applying the Canny filter (see B.6) to it.

B.3 Image Derivatives and Sobel Operator

The Sobel operator ([Sobel and Feldman, 1973]) consists of two discrete differentiation operators k_x and k_y which are used to calculate the (approximate) derivatives of an image in horizontal (k_x) and vertical direction (k_y), i.e:

$$\frac{\partial I}{\partial x} = k_x \otimes I \qquad \frac{\partial I}{\partial y} = k_y \otimes I, \tag{B.5}$$

with $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$ denoting the image derivatives and $\otimes$ convolution. Typical kernels are:

$$k_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad k_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{B.6}$$

Different kernel sizes can be used and also different kernel values. The obtained derivatives for the above kernels (Eq. B.6) are shown in Fig. B.2 (b) and (c), and for a different kernel of size $7x7$ in B.2(d) and (e). The bigger the kernel size the more robust is the filter to noise and less sensitive to details.

Other operators for computing image derivatives are for example Scharr [Jähne et al., 1999] and Laplace [Marr, 1982] filters. Image derivatives are needed to calculate the image gradient, (B.4).

B.4 Image Gradient

If an (intensity) image is interpreted as a two-dimensional (discrete) function assigning an intensity value to each pixel coordinate defined by x and y, i.e.

$$I(x, y) \mapsto [0, 255] \tag{B.7}$$

then the gradient of I is:

$$\nabla I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}), \tag{B.8}$$

with ∂x and ∂y being the partial derivatives of I in the horizontal and vertical direction. They can be computed for instance by applying the Sobel operator (B.3) to the image.

B.5 Pixel Orientation

The orientation of a pixel with coordinates x, y is the orientation of the image gradient (B.4), i.e.

$$o(x, y) = arctan(\frac{\delta y}{\delta x}), \tag{B.9}$$

with $o(x, y)$ being the orientation of the pixel and δy and δx the derivatives of the pixel in horizontal and vertical direction (i.e. shorthand for $\frac{\partial I}{\partial x}$ and $\frac{\partial I}{\partial y}$).

Usually the pixel orientation is discretized, such that each pixel can take one of the four directions 0°, 45°, 90°, and 135°.

B.6 Canny Filter for Edge Detection

The Canny filter [Canny, 1986] is a classic computer vision method for image edge detection, based on two thresholds and precomputed image gradients. It comprises multiple stages:

- Smoothing the image with a Gaussian filter to reduce noise.
- Gradient calculation of the image, e.g. using the Sobel operator (B.3), (but also other operators can be used) for subsequent processing.
- Non-maximum surpression to thin thicker lines, using the previously computed gradient information.
- Edge tracing using hysteresis also using the previously computed gradient information.

The hysteresis requires two thresholds, t_0 and t_1, with $t_0 \leq t_1$. All pixels with an absolute gradient value larger or equal than the higher threshold are considered to be definitely edge pixels and those with an absolute gradient value lower or equal than the lower threshold value are made non-edge pixels. Those between the thresholds are made edges dependent on their vicinity to the previously identified definite edge pixels. An example result of the Canny filter is shown in Fig. B.2 e).

B.7 Active Contours/Snakes

Active Contours or "Snakes" [Kass et al., 1988], are curves (or surfaces) that, by minimizing a predefined energy function, can lock onto regions of interest in an image, usually the outlines of some object. The energy function is normally defined as in Eq. B.10.

$$E_{total} = E_{in} + E_{ex}, \tag{B.10}$$

where E_{in} denotes an internal and E_{ex} an external energy. The latter serves to push the curve towards interesting image structures, such as edges, and the former encourages smooth curve shapes. Examples for internal and external energies for a curve $C(t)$ as in Eq. B.11

$$C(t) = (x(t), y(t)), \qquad 0 \leq s \leq 1, \tag{B.11}$$

are given by equations B.12 and B.13.

$$E_{in}(C(t)) = \alpha|\frac{\delta C}{\delta t}|^2 + \beta|\frac{\delta^2 t}{\delta^2 s}|^2, \tag{B.12}$$

$$E_{ex}(C(t)) = -(|\delta x(C(t))|^2 + |\delta y(C(t))|^2), \tag{B.13}$$

where $\delta x(C(t))$ and $\delta y(C(t))$ denote the horizontal and vertical image gradient at $C(t)$ (see B.4). The curve can be adjusted to the sought contour by iteratively minimizing the energy, for example by using dynamic programming.

An initial contour is usually placed roughly at where the sought object outline is, thus some a priori knowledge is required. Known difficulties with active contours are getting stuck in local minima or hooking on to undesired objects.

B.8 Splines

Splines [DeBoor, 1978] are piecewise defined parametric polynomials that provide an efficient way of mathematically describing complex shapes (e.g. self crossing curves and surfaces).

The **degree** of the used polynomial defines the degree of the spline, and the **order** of a spline is

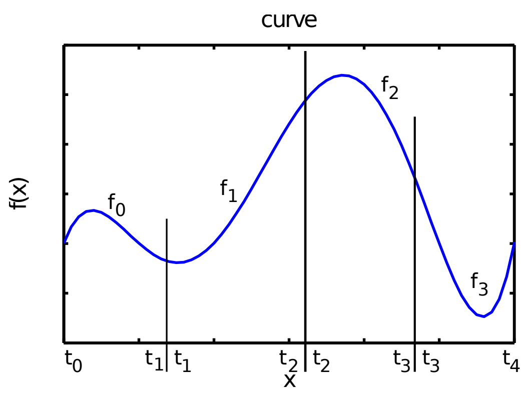


Figure B.3: A spline defined on the interval $[a = t_0, b = t_4]$. Each subinterval $[t_i, t_i + 1]$ with $[0 \leq i \leq 4]$ is described by a polynomial f_i of a fixed degree.

equal to the number of coefficients of the polynomial, i.e. one more than the degree. For example, cubic splines, which are most commonly used in computer graphics, are based on polynomials of degree 3 and have an order of 4. (Cubic splines are especially attractive since they result in curves with minimal curvature.) The joints of the polynomial pieces are called **knots**. If these are evenly spaced the spline is said to be **uniform**, otherwise it is called **non-uniform**. The vector containing all knots of a spline is called the **knot vector**. For example, a spline C might be defined on an interval $[a, b]$ and map to $\mathbb{R}$, see Eq. B.14.

$$C := [a, b] \mapsto \mathbb{R}. \tag{B.14}$$

Each piece of the spline, i.e. each polynomial f_i, is then defined on a subinterval $[t_i, t_{i+1}]$, with $0 \leq i \leq k$, such that

$$[a, b] = [t_0, t_1] \cup [t_1, t_2] \cdots \cup [t_{k-1}, t_k] \tag{B.15}$$

$$a = t_0 \leq t_1 \leq t_2 \cdots t_k = b \tag{B.16}$$

with k denoting the total number of subintervals, compare Fig. B.3 where $k = 4$. The joints t_i are the knots and the vector $[t_0, t_1, ..t_k]$ is the knot vector. (Note, that a spline may map to any dimension $\mathbb{R}^n$ with $n \in \mathbb{N}$ and is not restricted to $\mathbb{R}$ as in this example.) To be able to describe space curves, i.e. curves that can map back on themselves, e.g. circles, splines are expressed via parametric polynomials. A new variable $u \in [0, 1]$ is introduced and the interval $[t_i, t_{i+1}]$ normalized by setting

$$u = (t_i - x)/(t_{i+1} - t_i), \tag{B.17}$$

with x being the independent variable of the polynomial.

To construct a spline of a desired shape the user must provide sufficient information to allow the calculation of the coefficients of the spline segments. (Such segments are also sometimes referred to as spans). If the curve maps to $\mathbb{R}^n$ this may be a list of vectors of dimension n that the curve is supposed to pass through. In the special case of $n = 2$ these vectors are points in the plane and are referred to as **control points**. The number of control points must be equal to or larger than the order of the spline (otherwise the spline coefficients cannot be computed). There are two types of splines, those that interpolate the given control points and those that approximate them. Accordingly, the former are called interpolating and the latter approximating splines. Examples for interpolating splines are (cubic) Hermite-, Cardinal- and Catmull-Rom-splines (see section B.8.1), examples for approximating splines are Bezier curves, B-splines (see section B.8.2), and NURBS.

Given the required information a spline segment can be conveniently expressed via simple matrix multiplication:

$$C(u) = UMX^T, \qquad u \in [0, 1], \tag{B.18}$$

where X is the vector containing the provided information , U is a vector containing the independent variable of the used polynomial of particular degrees, and M is the so called **blending matrix** which is specific for each spline and can be obtained by solving for the coefficients of the used polynomial by imposing certain constraints on the curve. The expressions

$$b_1 = Um_1 \tag{B.19}$$

$$b_2 = Um_2 \tag{B.20}$$

$$\dots \tag{B.21}$$

$$b_e = Um_e \tag{B.22}$$

are called the **basis functions** of the spline, with $m_1, m_2 \dots m_e$ denoting the column vectors of M and e the **order** of the spline. Another notation often found is

$$C(u) = \sum_{i=1}^{n} x_i B_i(u), \tag{B.23}$$

here the basis functions are $B_i(u)$ and x_i are the n information provided by the user.

B.8.1 Cubic Hermite, Cardinal, and Catmull-Rom/Overhauster Splines

The cubic Hermite-spline (or cspline) is an interpolating spline using polynomials of degree 3, hence cubic, that represents each segment in Hermite form, i.e. instead of 4 control points it uses two control points and two tangents, one at each point. Points and tangent vectors are provided by the user. A spline segment is thus defined by:

$$C(u) = U_c M_c X_c^T, \tag{B.24}$$

$$U_c = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}, \tag{B.25}$$

$$M_c = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \tag{B.26}$$

$$X_c = \begin{bmatrix} P_1 & P_2 & r_1 & r_2 \end{bmatrix} \tag{B.27}$$

with P_i denoting a control point i and r_i a tangent at control point i.

B.8.1.1 Derivation of M_c

In the following we exemplary show how matrix M_c is derived. Since a polynomial of degree 3 is defined by

$$p(u) = au^3 + bu^2 + cu + d \tag{B.28}$$

$$= UC_x^T \tag{B.29}$$

$$U = \begin{bmatrix} u^3, u^2, u, 1 \end{bmatrix} \tag{B.30}$$

$$C_x = \begin{bmatrix} a & b & c & d \end{bmatrix} \tag{B.31}$$

with u denoting the independent variable and a, b, c, and d coefficients, the curve for a particular spline segment can be derived by solving for the 4 coefficients. For that we have to define additional constraints. In case of this particular spline we require that the spline curve interpolates the knots, i.e. the given control points. Since $u \in [0, 1]$ we obtain the two equations:

$$p(0) = P_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} C_x \tag{B.32}$$

$$p(1) = P_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} C_x. \tag{B.33}$$

Furthermore the derivatives at the knots are equal to r_1 and r_2, thus

$$\frac{dp}{du} = \begin{bmatrix} 3u^2 & 2u & u & 0 \end{bmatrix} C_x \Rightarrow \tag{B.34}$$

$$r_0 = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} C_x, \tag{B.35}$$

$$r_1 = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} C_x. \tag{B.36}$$

Written in matrix form we obtain Eq. (B.37):

$$\begin{bmatrix} p_1 \\ p_2 \\ r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} C_x, \tag{B.37}$$

the matrix in this expression we refer to as B. Thus, $M_c = B^{-1}$.

If the tangents are not specified by the user but calculated from the control points in the following way, see Eq. (B.38):

$$r_i = a(P_{i+1} - P_{i-1}) \tag{B.38}$$

we speak of **Cardinal-splines**. Thus, Cardinal-splines are a subset of cubic Hermite-splines. Note, that the magnitude of the tangent is defined by the parameter $a \in [0, 1]$, which is also referred to as **tension**. A spline with a high tension has higher curvature.

Furthermore, if the parameter of the Cardinal-spline is set to 0.5 the curve is called **Catmull-Rom** [Catmull and Rom, 1974] or **Overhauster- spline**, thus Catmull-Rom-splines (or shorter CR-splines) are again a subset of Cardinal-splines. They can be obtained by the following expression (the derivation can be found in any standard textbook about splines, e.g. [Salomon, 2006]):

$$c(u) = 0.5 \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & 3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}, \tag{B.39}$$

with P_i denoting the i'th spline control point. Note, the segment is only interpolating P_i and P_{i+1}. To obtain a curve that interpolates P_{i-1} and P_{i+2}, one can set $P_{i-1} = P_i$ and $P_{i+1} = P_{i+2}$.

Properties shared by all these curves are: They pass through the given control points, (thus interpolating) and there they are C^0 continuous. The slope of the CR-spline in the control points is per definition given by Eq. (B.40).

$$m_i = (P_{i+1} - P_{k-i})/2 \qquad \text{slope of CR-spline at } i\text{'th control point.} \tag{B.40}$$

B.8.2 B-Spline

A spline curve that is based on B-spline basis functions is called a B-Spline. It is an approximating spline i.e. the curve does not interpolate the control points, which in this context are called De-Boor points. Its definition is given in Eq. (B.41).

$$C(u) = \sum_{i=0}^{n} P_i B_{i,e-1,\tau}(u), \tag{B.41}$$

where P_i is the *i′th* control point and $n+1$ the number of given control points (which must be equal to or larger than the spline's order e). $B_{i,e-1,\tau}$ is the B-spline basis function which is defined on the knot vector τ given in Eq. (B.42) and recursively defined in Eq. (B.44).

$$\tau = [t_0, t_1, \cdots t_{n+e}], \text{ where } t_j \leq t_{j+1}, \text{ and } 0 \leq j \leq n+e. \tag{B.42}$$

$$B_{j,0,\tau}(u) := \begin{cases} 1, & t_j \leq u \leq t_{j+1} \\ 0, & else \end{cases} \tag{B.43}$$

$$B_{j,d,\tau}(u) := \frac{u - t_j}{t_{j+d} - t_j} B_{j,d-1,\tau}(u) + \frac{t_{j+d+1} - u}{t_{j+d+1} - t_{j+1}} B_{j+1,d-1,\tau}(u), \qquad d \geq 0 \tag{B.44}$$

Here, $d = e - 1$ is the degree of the curve.

A uniform cubic B-spline can be expressed via the following matrix notation:

$$C_i(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix} \text{for } 0 \leq u \leq 1. \tag{B.45}$$

Important properties of B-splines are that they are always within the convex hull of their control points and that the change of a control point effects the curve shape only locally. Precisely this means that changing the point P_i influences the shape of the curve only within the interval determined by the number of control points equal to the order of the spline around it. For example, if the spline is cubic, i.e. the order $e = 4$, and the point P_i is changed, the curve shape is only altered within the span determined by the points P_{i-1}, P_i, P_{i+1}, and P_{i+2}.

Appendix C

Miscellaneous

C.1 Parameter Table

Chapter 4 contains the specification of various parameters which we here summarize.

Table C.1: Overview of fiddle parameters and their used values in chapter 4.

parameter	**used value**	**reference**
allowed gap width between two lane segments	40 pixels	4.2.2.4
orientation threshold for lane construction	$\pi/2$	4.2.2.4
standard deviation (σ) for Kalman filter for lane tracking	26.5	4.2.2.5
process variance ($\mathbf{Q}$) for Kalman filter for lane tracking	0.01	4.2.2.5
side length K for $k - cos$ curvature	6	4.2.2.6
threshold for determining discontinuity using $k - cos$ curvature	0.5	4.2.2.6
ω for weighting the similarity of detected street lanes	20, 10, 5, 5 for the first four ω entries and 1 for all remaining ones	4.3.2.2
$thresh$_v for computing PAR entry similarity (visual)	10	4.3.2.2
$thresh$_st for computing PAR entry similarity (steer)	20	4.3.2.2
$threshR$_v, upper threshold for accepting PAR retrievals (visual)	200	4.3.2.3
$threshR$_st$_{\text{past}}$, upper threshold for accepting PAR retrievals (steer)	100	4.3.2.3
the parameter configuration used for DIFF	$\sigma = 4$, $G = 10$; $n = 10$; $\tau = 1$	4.3.2.4.1.
canny parameters for night driving	lower threshold 100, higher threshold 350	4.3.3.6

C.2 Pseudocode

Algorithm 2: ConstructCurveSegments (master function for curve segment construction)

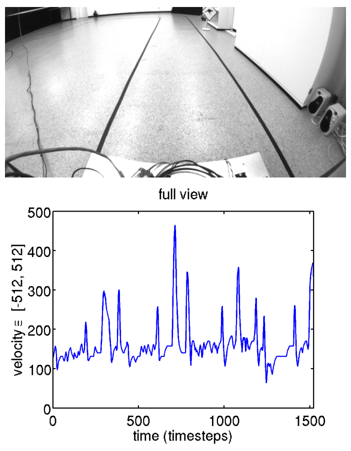
```
Input: edgeImage, orientationImage
Output: curve[] //list of curve segments
currentPixel ⟵ edgeImage.bottomLeft;
ctr ⟵ 0;
while currentPixel < edgeImage.upperRight do
    if currentPixel == edgePixel then
        currentPixel = “start”; //label pixel as startpixel
        currentPixel = “read”; //label pixel as having been read
        curve[ctr].add(currentPixel);
        orientation ⟵ currentPixel.orientation;
        traceNeighbours(currentPixel, orientation, curve[ctr],“leftToRight”);
        //slave function
    else
        currentPixel = ”end”; //label pixel as endpixel
        currentPixel + +;
        ctr + +;
```

Algorithm 3: traceNeighbors (slave function for curve segment construction)

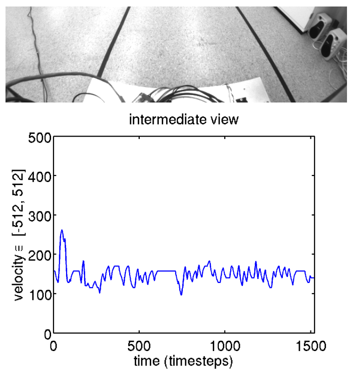
```
Input: currentPixel, orientation, curve[ctr], flag "leftToRight" or "rightToLeft"
Output: curve[ctr] //a traced curve segment
if flag == "leftToRight" then
    set neighbors according to Fig. 4.4(a)
else
    set neighbors according to Fig. 4.4(b)
//check neighbors
if neighbor1 == edgePixel then
    if neighbor1 == edgePixel and neighbor1 ! = "read"
    and neighbor1.orientation == orientation then
        neighbor1 = "read"; //label pixel as having been read
        curve[ctr].add(neighbor1);
        //add the pixel to the curve
        //recurvsive call with this new curve-pixel, its orientation, and
        //desired  tracing-direction
        traceNeighbours( neighbor1, orientation, curve[ctr], "leftToRight");
else
    //check second neighbor
    ⋮
else
    //check third neighbor
    ⋮
else
    //relax orientation criterion and check all three neighbors again, now with the new
    orientation
    ⋮
```

C.3 Velocity Control is Dependant on Distal Information

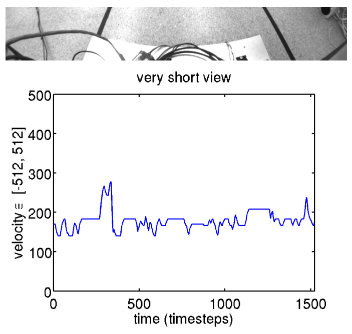
Human velocity control is dependent on distal information. This can be seen by the following experiment: In the indoor scenario the human driver was instructed to follow the track as fast as possible while being shown different cutouts of the recorded images which are shown in the upper row of Fig. C.3. During the first trial the driver was shown the entire recorded image frame (full view), next only the lower half of the recorded image was shown, (intermediate view), and in a third trial even this view was reduced such that the driver could only see a very short part of the lane at the image bottom, (very short view). The produced velocity profile is shown in the lower row of Fig. C.3 underneath the corresponding image view.



(a)



(b)



(c)

Figure C.1: Top row: Three different views ((a) full view, (b) intermediate, and (c) very short view) shown to the driver while following the indoor-track as fast as possible. Bottom row: The accordingly produced velocity profiles.

It can clearly be seen, that the velocity profile of the full-view-trial differs in amplitude in comparison to the other two profiles which both show a less differentiated signal. Thus, it can be concluded that human velocity control depends on distal information.

Bibliography

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*. ACM Press.

Aha, D. W., editor (1997). *Lazy Learning*, volume 11 of *Artificial Intelligence Review*, chapter Editorial, pages 7–10. Kluwer Academic Publishers.

Aloimonos, J., Weiss, I., and Bandopadhay, A. (1987). Active vision. *International Journal on Computer Vision*, 1:333–356.

Alt, H. and Godau, M. (1995). Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91.

Aly, M. (2008). Real time detection of lane markers in urban streets. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 7–12.

Anderson, J. R., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, 111(4):1036–1050.

Apostoloff, N. and Zelinsky, A. (2003). Robust vision based lane tracking using multiple cues and particle filtering. In *IEEE Intelligent Vehicles Symposium*.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483.

Baddeley, A. (2002). Is working memory still working? *European Psychologist*, 7(2-3):85–97.

Bajcsy, R. (1988). Active perception. In *IEEE Proceedings*, volume 76, pages 996–1006.

Ballard, D. (1991). Animate vision. *Artificial Intelligence*, 48:1–27.

Barsalou, L. W. (2009). Simulation, situated conceptualization, and prediction. *Philos Trans R Soc Lond B Biol Sci.*, 364:12811289.

Bastian, A. J. (2006). Learning to predict the future: the cerebellum adapts feedforward movement control. *Current Opinion in Neurobiology*, 16(6):645–649.

Bernstein, N. A. (1996). *Dexterity and its Development*. Lawrence Erlbaum Associates.

Berthenthal, B. I. (1996). Origins and early development of perception, action, and representation. *Annual Review of Psychology*, 47:43:431–459.

Bertozzi, M. and Broggi, A. (1996). Real-time lane and obstacle detection on the system. In *IEEE Intelligent Vehicles, 1996*, pages 213–218.

Bertozzi, M. and Broggi, A. (1998). Gold: A parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing*, 7(1):62–81.

Blake, A. and Isard, M. (2000). *Active Contours*. Springer.

Blakemore, S. J., Wolpert, D., and Frith, C. (2000). Why can't you tickle yourself. *NeuroReport*, 11:11–16.

Bottou, L. and Vapnik, V. (1992). Local learning algorithms. *Neural Computation*, 4:888–900.

Braess, H. H. and Reichart, G. (1995). Prometheus: Vision des intelligenten automobils auf intelligenter strasse? versuch einer kritischen wrdigung. *ATZ. Automobiltechnische Zeitschrift*, 97:200–205.

Broggi, A. and Berte, S. (1995). Vision-based road detection in automotive systems: A real-time expectation-driven approach. *Journal of Artificial Intelligence Research*, 3:325–348.

Broggi, A., Fascioli, A., and Bertozzi, M. (1999). *The Argo Autonomous Vehicle: The Experience of the ARGO Autonomous Vehicle*. World Scientific Pub Co.

Brookhuis, K. and de Waard, D. (1999). Limiting speed, towards an intelligent speed adapter (isa),. *Transportation Research Part F: Traffic Psychology and Behaviour*, 2:81–90.

Brookhuis, K. A., de Waard, D., and H.Janssen, W. (2001). Behavioural impacts of advanced driver assistance systems-an overview. *European Journal of Transport and Infrastructure Research*, 1(3).

Brooks, R. A. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2):3–15.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, 47:139–160.

Byrne, R. W. and Russon, A. E. (1998). Learning by imitation: A hierarchical approach. *Behavioral and Brain Sciences*, 21:667–684.

Canny, J. F. (1986). A computational approach to edge detection. *IEEE Trans. Pattern Anal. Machine Intell.*, 8:679–698.

Cartwright, B. A. and Collett, T. S. (1982). How honey bees use landmarks to guide their return to a food source. *Nature*, 295:560 – 564.

Cartwright, B. A. and Collett, T. S. (1983). Landmark learning in bees. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 151:521–543.

Catmull, E. and Rom, R. (1974). *A Class of Local Interpolating Splines in Computer Aided Geometric Design*. Academic Press.

Charness, N. (1976). Memory for chess positions: Resistance to interference. *Journal of Experimental Psychology: Human Learning and Memory*, 2(6):641–653.

Charness, N., Reingold, E. M., Pomplun, M., and Stampe, D. M. (2001). The perceptual aspect of skilled performance in chess: Evidence from eye movements. *Memory and Cognition*, 8:1146–1152.

Chase, W. G. and H. A. Simon, H. (1973). *Visual information processing*, chapter The minds eye in chess, pages 215–281. New York: Academic Press.

Chen, S., Billings, S. A., and Grant, P. M. (1990). Non-linear system identification using neural networks. *Int. J. Control*, 51:1191–1214.

Churchland, P., Ramachandran, V. S., and Sejnowski, T. (1994). *A Critique of Pure Vision*. MIT Press.

Clark, A. and Grush, R. (1999). Towads a cognitive robotics. *Adaptive Behavior*, 7:5–16.

Collett, T. (1996). Insect navigation en route to the goal: multiple strategies for the use of landmarks. *Journal of Experimental Biology*, 199(1):227–235.

Collettand, M. and Collett, T. S. (2000). How do insects use path integration for their navigation? *Biol. Cybern.*, 83:245259.

Coughlin, J. F., Reimer, B., and (2009)., B. M. (2009). Driver wellness, safety & the development of an awarecar. White paper, AgeLab, MIT.

Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storag capacity. *Behavioral and Brain Scienes*, 4:87–185.

Cummins, M. and Newman, P. (2008). FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. *The International Journal of Robotics Research*, 27(6):647–665.

Curby, K. M. and Gauthier, I. (2009). To the trained eye: Perceptual expertise alters visual processing. *Topics in Cognitive Science*, online:1–13.

DARPA (2010). online: www.darpa.mil.

Daugman, J. (1985). Uncertainty relation for resolution in space, spatial-frequency, and orientation optimized by two-dimensional visual cortical filters. *J. Opt. Soc. Am. A-Opt. Image Sci. Vis.*, 2(7):1160–1169.

Dautenhahn, K. and Nehaniv, C. L., editors (2002). *Imitation in Animals and Artifacts*. MIT Press.

Daw, N. D., Niv, Y., and Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, 8:1704 – 1711.

DeBoor, C. (1978). *A Practical Guide to Splines*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K.

DeGroot, A. D. (1949). Het denken van den schaker. *Noord Hollandscher*.

DeGroot, A. D. (1978). *Thought and Choice in Chess*. Mouton Publishers.

Demiris, Y. and Meltzoff, A. (2008). The robot in the crib: a developmental analysis of imitation skills in infants and robots. *Infant and Child Development*, 17:43–53.

Desmurget, M. and Grafton, S. (2000). Forward modeling allows feedback control for fast reaching movements. *Trends in Cognitive Sciences*, 4(11):423 – 431.

Dickmanns, E. (2002). The development of machine vision for road vehicles in the last decade. In *Intelligent Vehicle Symposium, IEEE*.

Dickmanns, E. (2007). *Dynamic Vision for Perception and Control of Motion*. Springer.

Dickmanns, E. D. (1998). Vehicles capable of dynamic vision: a new breed of technical beings? *AI*, 103:49–76.

Dickmanns, E. D. and Graefe, V. (1988). Dynamic monocular machine vision. *Machine Vision and Applications*, 1:223–240.

Diplecs (2010). Official diplecs website. online: http://www.diplecs.eu/.

Donges, E. (1978). A two-level model of driver steering behaviour. *Hum Factors*, 20:691707.

Dorf, R. C. and Bishop, R. H. (2007). *Modern Control Systems (11th Edition) (Pie)*. Prentice Hall.

Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10:112–122.

Eberly, D. (2010). online: www.geometrictools.com/index.htm.

Eisat (2010). Eisat website. online: http://www.mi.auckland.ac.nz/EISATS.

Ericsson, K. A. and Kintsch, W. (1995). Long-term working memory. *Psychological Review*, 102:211–245.

Ericsson, K. A. and Lehmann, A. C. (1996). Expert and exceptional performance: Evidence of maximal adaptation to task constraints. *Annual Review of Psychology*, 47(1):273–305.

EU-Newsletter (2010). online: ftp://ftp.cordis.europa.eu/pub/fp7/ict/docs/fet/fetnl06_en.pdf.

Feigenbaum, E. A. and Simon, H. A. (1962). A theory of the serial position effect. *British Journal of Psychology*, 53:307–320.

Feigenbaum, E. A. and Simon, H. A. (1984). Epam-like models of recognition and learning. *Cognitive Science*, 8:305–336.

Ferrari, P. F., Bonini, L., and Fogassi, L. (2009). From monkey mirror neurons to primate behaviours: possible 'direct' and 'indirect' pathways. *Philos Trans R Soc Lond B Biol Sci.*, 364:2311–23.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

Fiser, J. and Aslin, R. N. (2005). Encoding multielement scenes: statistical learning of visual feature hierarchies. *J Exp Psychol Gen*, 134(4):521–537.

Fitts, P. and Posner, M. (1967). *Human Performance*. Monterey.

Flanagan, J. R. and Wing, A. M. (1997). The role of internal models in motion planning and control: Evidence from grip force adjustments during movements of hand-held loads. *J. Neurosci.*, 17(4):1519–1528.

Francis, B. and Wonham, W. (1976). The internal model principle of control theory. *Automatica*, 12(5):457 – 465.

Franz, M., Scholkopf, B., H.A., M., , and H.H., B. (March 1998). Learning view graphs for robot navigation. *Autonomous Robots*, 5:111–125(15).

Freyhof, H., Gruber, H., and Ziegler, A. (1992). Expertise and hierarchical knowledge representation in chess. *Psychological Research*, 54:1430–2772.

Freymann, R. (2008). *Motion and Vibration Control*, chapter Driver Assistance Technology to Enhance Traffic Safety, pages 71–81. Springer Netherlands.

Gallistel, C. R. (1980). *The Organization of Action: A New Synthesis*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Gat, E. (1997). On three-layer architectures. In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, editors, Artificial Intelligence and Mobile Robots. MIT/AAAI Press.

Gautama, T. and Van Hulle, M. (2002). A phase-based approach to the estimation of the optical flow field using spatial filtering. *IEEE Transactions on Neural Networks*, 13(5):1127–1136.

Gerstmayr, L., Mallot, H. A., and Wiener, J. M. (2008). A minimalistic model of visually guided obstacle avoidance and path selection behavior. In *Proceedings of the international conference on Spatial Cognition VI*, pages 87–103, Berlin, Heidelberg. Springer-Verlag.

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates.

Glover, S. (2004). Separate visual representations in the planning and control of action. *Behavioral and Brain Scienes*, 27:3–78.

Gobet, F. and Chassy, P. (2009). Expertise and intuition: A tale of three theories. *Minds Mach.*, 19(2):151–180.

Gobet, F. and Clarkson, G. (2004). Chunks in expert memory: evidence for the magical number four ... or is it two? *Memory (Hove, England)*, 12(6):732–747.

Gobet, F. and Simon, H. A. (1996). Templates in chess memory: A mechanism for recalling several boards. *Cognitive Psychology*, 31:1–40.

Goldbeck, J. and Huertgen, B. (1999). Lane detection and tracking by video sensors. In *Proc. IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems*, pages 74–79.

Graybiel, A. M. (1998). The basal ganglia and chunking of action repertoires. *Neurobiol Learn Mem*, 70(1-2):119–136.

Greenwald, A. G. (1970). Sensory feedback mechanisms in performance control: With special reference to the ideomotor mechanism. *Psychological Review*, 77:73–99.

Grush, R. (2004). The emulation theory of representation: Motor control, imagery, and perception. *Behavioral and Brain Sciences*, 27:377442.

Halsband, U. and Lange, R. K. (2006). Motor learning in man: a review of functional and clinical studies. *J Physiol Paris*, 99(4-6):414–424.

Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Hesslow, G. (1994). Will neuroscience explain consciousness? *Journal of Theoretical Biology*, 171(1):29 – 39.

Hesslow, G. (2002). Conscious thought as simulation of behaviour and perception. *Trends in Cognitive Sciences*, 6(6):242–247.

Hikosaka, O., Nakamura, K., Sakai, K., and Nakahara, H. (2002). Central mechanisms of motor skill learning. *Curr Opin Neurobiol*, 12(2):217–222.

Hoch, S., Schweigert, M., Althoff, F., and Rigoll, G. (2007). The bmw surf project: A contribution to the research on cognitive vehicles. In *Proceedings of the 2007 Intelligent Vehicles Symposium*.

Hoffmann, H. (2007). Perception through visuomotor anticipation in a mobile robot. *Neural Networks*, 20(1):22 – 33.

Hoffmann, H. and Möller, R. (2004). Action selection and mental transformation based on a chain of forward models. In Schaal, S., Ijspeert, A., Billard, A., Vijayakumar, S., Hallam, J., and Meyer, J.-A., editors, *From Animals to Animats 8 (Proc. 8th Intl. Conf. on the Simulation of Adaptive Behavior)*, pages 213–222. MIT Press.

Hommel, B., Müsseler, J., Aschersleben, G., and Prinz, W. (2001). The theory of event coding (tec): a framework for perception and action planning (with commentary). *Behavioral and Brain Sciences*, 24(5).

Hutchinson, S. and Corke, G. D. H. P. I. (1996). A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670.

Huttenlocher, D. P., Klanderman, G. A., and Rucklidge, W. A. (1993). Comparing images using the hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):850–863.

Hwang, C. H., Massey, N., Miller, B. W., and Torkkola, K. (2003). Hybrid intelligence for driver assistance. In *FLAIRS*.

Jähne, B., Scharr, H., and S.Krkel (1999). *Handbook of Computer Vision and Applications*. Academic Press.

Jeannerod, M. (2006). *Motor Cognition*. Oxford University Press, USA.

Jordan, M. I. (1996). *Handbook of Perception and Action: Motor Skills*, chapter Computational aspects of motor control and motor learning. Academic Press, 1996.

Jordan, M. I. and Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354.

Julier, S. and Uhlmann, J. (1997). A new extension of the kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls, Orlando, FL.*

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transaction of the ASMEJournal of Basic Engineering*, pages 33–45.

Kandil, F. I., Rotter, A., and Lappe, M. (2009). Driving is smoother and more stable when using the tangent point. *Journal of Vision*, 9:1–11.

Karni, A., Meyer, G., Rey-Hipolito, C., Jezzard, P., Adams, M. M., Turner, R., and Ungerleider, L. G. (1998). The acquisition of skilled motor performance: Fast and slow experience-driven changes in primary motor cortex. In *Proc Natl Acad Sci U S A. 1998*, volume 95, page 861868.

Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331.

Kawato, M. (1999). Internal models for motor control and trajectory planning. *Curr Opin Neurobiol*, 9(6):718–727.

Kawato, M., Furukawa, F., and Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185.

Kim, Z. (2006). Realtime lane tracking of curved local road. In *Proceedings of the IEEE Intelligent Transportation Systems Conference*.

Klatzky, R. L. (1998). Allocentric and egocentric spatial representations: Definitions, distinctions, and interconnections. In *Spatial Cognition, An Interdisciplinary Approach to Representing and Processing Spatial Knowledge*, pages 1–18, London, UK. Springer-Verlag.

Koch, I. and Hoffmann, J. (2000). Patterns, chunks, and hierarchies in serial reaction-time tasks. *Psychological Research*, 63:2235.

Koffka, K. (1999). *Principles of Gestalt Psychology*. Routledge Chapman & Hall.

Kwasnicka, H. and Dudala, M. (2002). Neuro-fuzzy driver learning from real driving observations. In *Proceedings of the Artificial Intelligence in Control and Managamnent*.

Latombe, J.-C. (1990). *Robot Motion Planning*. Springer.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

Lavoie, P. (2002). nurbs++. online: http://sourceforge.net/projects/libnurbs/.

Lee, D. N. (1976). A theory of visual control of braking based on information about time-to-collision. *Perception*, 5:437–459.

Leibe, B., Cornelis, N., Cornelis, K., and Van Gool, L. (2007). Dynamic 3d scene analysis from a moving vehicle. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1423–30, Minneapolis.

Leibe, B., Schindler, K., Cornelis, N., and Van Gool, L. (2008). Coupled object detection and tracking from static cameras and moving vehicles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(10):1683–1698.

Li, L., Wang, F.-Y., and Zhou, Q. (2006). Integrated longitudinal and lateral tire/road friction modeling and monitoring for vehicle motion control. *Intelligent Transportation Systems, IEEE Transactions on*, 7(1):1–19.

Lindgren, A. and Chen, F. (2007). State of the art analysis: An overview of advanced driver assistance systems (adas) and possible human factors issues. In *Human Factors and Economic Aspects on Safety. Swedish Network for Human Factors Conference*.

Liu, L. G., Woergoetter, F., and Markelic, I. (2010). Combining statistical hough transform and particle filter for robust lane detection and tracking. In *Proc. IEEE Intelligent Vehicles Symposium, 2010 (submitted)*.

Luft, A. R. and Buitrago, M. M. (2005). Stages of motor skill learning. *Molecular Neurobiology*, 32(3):205–216.

Mallot, H. A. and Basten, K. (2009). Embodied spatial cognition: Biological and artificial systems. *Image and Vision Computing*, 27(11):1658 – 1670. Cognitive Systems: Perception, Action, Learning.

Mallot, H. A., Blthoff, H. H., Little, J. J., and Bohrer, S. (1991). Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biol Cybern*, 64(3):177–185.

Mammar, S., Glaser, S., and Netto, M. (2006). Time to line crossing for lane departure avoidance: a theoretical study and an experimental setting. *Intelligent Transportation Systems, IEEE Transactions on*, 7(2):226–241.

Markelić, I., Kjæ-Nielsen, A., Pauwels, K., Jensen, L. B. W., Chumerin, N., Vidugiriene, A., Tamosiunaite, M., Hulle, M. V., Krüeger, N., Rotter, A., and Wörgötterm, F. (2010). The driving school system: Learning automated basic driving skills from a teacher in a real car. *IEEE Transactions on Intelligent Transportation Systems (submitted)*.

Markelic, I., Kulvicius, T., Tamosiunaite, M., and Wörgötter, F. (2008). Anticipatory driving for a robot-car based on supervised learning. In *ABiALS*, pages 267–282.

Marr, D. (1982). *Vision: a computational investigation into the human representation and processing of visual information*. W. H. Freeman, San Francisco.

Massaro, D. (1990). *Relationships between perception and action: Current approaches*, chapter An information-processing analysis of perception and action, pages 133–166. Springer.

McCall, J. and Trivedi, M. (2006). Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *Intelligent Transportation Systems, IEEE Transactions on*, 7(1):20–37.

McCall, J. C. and Trivedi, M. M. (2004). An integrated, robust approach to lane marking detection and lane tracking. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 533–537.

McNamara, T., Sluzenski, J., and Rump, B. (2008). Human spatial memory and navigation. In Byrne, J. H., editor, *Learning and Memory: A Comprehensive Reference*, pages 157 – 178. Academic Press, Oxford.

Meltzoff, A. N. (1988). *Social learning: Psychological and biological perspectives*, chapter The human infant as homo imitans, page 319341. Erlbau, Hillsdale, NJ.

Meltzoff, A. N. and Moore, M. K. (1997). Explaining facial imitation: A theoretical model. *Early Development and Parenting*, 6:179–192.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97.

Mitchell, T. M. (1997). *Machine Learning*. MIT Press and The McGraw-Hill Companies.

Montemerlo, M., Thrun, S., Dahlkamp, H., Stavens, D., and Strohband, S. (2006). Winning the darpa grand challenge with an ai robot. In *Proceedings of American Association of Artificial Intelligence*.

Moravec, H. (1982). The cmu rover. In *Proceedings of AAAI-82*, pages 377–380.

Moravec, H. (1990). The stanford cart and the cmu rover. In Cox, I. J. and Wilfong, G. T., editors, *Autonomous Robot Vehicles*, pages 407–41. Springer-Verlag.

Mosteller, F. and Tukey, J. (1977). *Data Analysis and Regression: A Second Course in Statistics*. Addison-Wesley Reading, Mass.

NDR (2009). TV report about the DRIVSCO system (in German). online: http://www1.ndr.de/mediathek/index.html?media=ndsmag2340.

Nelles, O. (2002). *Nonlinear System Identification*, volume 13. Springer.

Ng, A. Y. and Russel, S. (2000). Algorithms for inverse reinforcement learning. In *Proc. ICML*.

Nilsson, N. J. (1969). A mobile automaton: An application of artificial intelligence techniques. In *IJCAI*, pages 509–520.

Nissen, M. J. and Bullemer, P. (1987). Attentional requirements of learning: Evidence from performance measures. *Cognitive Psychology*, 19(1):1 – 32.

Nissen, S. (2005). Neural networks made simple. *Software 2.0*, 2:14–19.

Nosengo, N. (2009). The bot that plays ball. *Nature*, 460:1076–1078.

O'Keefe, J. and Nadel, L. (1979). The hippocampus as a cognitive map. *Behavioral and Brain Sciences*, 2:487–533.

O'Regan, J. K. and Noë, A. (2001). A sensorimotor account of vision and visual consciousness. *Behav Brain Sci*, 24(5):939–73; discussion 973–1031.

Partouche, D., Pasquier, M., and Spalanzani, A. (2007). Intelligent speed adaptation using a self-organizing neuro-fuzzy controller. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 846–851.

Pasquier, M. and Oentaryo, R. J. (December 2007). Learning to drive the human way: a step towards intelligent vehicle. *International Journal of Vehicle Autonomous Systems*, 6:24–47(24).

Pauwels, K., Krueger, N., Lappe, M., Woergoetter, F., and van Hulle, M. (2010). A cortical architecture on parallel hardware for motion processing in real-time. *Journal of Vision (submitted).*

Pauwels, K. and Van Hulle, M. (2008). Realtime phase-based optical flow on the GPU. In *IEEE Conference on Computer Vision and Pattern Recognition, Workshop on Computer Vision on the GPU.*

Pfeifer, R. and Scheier, C. (1999). *Understanding Intelligence.* MIT Press.

Piaget, J. (1952). *The origins of Intelligence in Children.* International Universities Press.

Pomerleau, D. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1*. Morgan Kaufmann.

Pomerleau, D. (1990). Neural network based autonomous navigation. In *NAVLAB90*, pages 558–614.

Pomerleau, D. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97.

Pomerleau, D. (1995). Ralph: Rapidly adapting lateral position handler. In *IEEE Symposium on Intelligent Vehicles*, pages 506 – 511.

Pomerleau, D. A. (1999). Neural network vision for robot driving. In *The Handbook of Brain Theory and Neural Networks*. M. Arbib.

Posner, M. I. (1978). *Chronometric explorations of mind : the third Paul M. Fitts lectures, delivered at the University of Michigan, September 1976 / Michael I. Posner*. L. Erlbaum Associates ; distributed by the Halsted Press Division of Wiley, Hillsdale, N.J. : New York :.

Prinz, W. (1 June 1997). Perception and action planning. *The European Journal of Cognitive Psychology*, 9:129–154(26).

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the ieee*, volume 77.

Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning. In *international joint conference on artificial intelligence*, pages 2586–2591.

Reingold, E. M., Charness, N., Pomplun, M., and Stampe, D. M. (2001a). Visual span in expert chess players: Evidence from eye movements. *Psychological Science*, 12:48–55.

Reingold, E. M., Charness, N., Schulteus, R. S., and Stampe, D. M. (2001b). Perceptual automaticity in expert chess players: Parallel encoding of chess relations. *Psychonomic Bulletin & Review*, 8:504–510.

Richman, H. B., Staszewski, J. J., and Simon, H. A. (1995). Simulation of expert memory using epam iv. *Psychol Rev*, 102(2):305–330.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591.

Riedmiller, M., Montemerlo, M., and Dahlkamp, H. (2007). Learning to drive a real car in 20 minutes. In *Proc. Frontiers in the Convergence of Bioscience and Information Technologies FBIT 2007*, pages 645–650.

Rosenbaum, D. A., Carlson, R. A., and Gilmore, R. O. (2001). Acquisition of intellectual and perceptual-motor skills. *Annu Rev Psychol*, 52:453–470.

Rosenbaum, D. A., Kenny, S. B., and Derr, M. A. (1983). Hierarchical control of rapid movement sequences. *J Exp Psychol Hum Percept Perform*, 9(1):86–102.

Rosenfeld, A. and Johnston, E. (1973). Angle detection on digital curves. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2:875–878.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Sabatini, S., Gastaldi, G., Solari, F., Diaz, J., Ros, E., Pauwels, K., Van Hulle, M., Pugeault, N., and Krüger, N. (2007). Compact and accurate early vision processing in the harmonic space. In *International Conference on Computer Vision Theory and Applications*, pages 213–220, Barcelona.

Safelane (2008). collected publications of the safelane project. online: http://www.prevent-ip.org/en/public_documents/publications/ safelane_publications.htm.

Sakai, K., Hikosaka, O., and Nakamura, K. (2004). Emergence of rhythm during motor learning. *Trends Cogn Sci*, 8(12):547–553.

Sakai, K., Kitaguchi, K., and Hikosaka, O. (2003). Chunking during human visuomotor sequence learning. *Exp Brain Res*, 152(2):229–242.

Salomon, D. (2006). *Curves and Surfaces for Computer Graphics*. springer.

Sammut, C., Hurst, S., Kedzier, D., and Michie, D. (1992). Learning to fly. In *ML*, pages 385–393.

Simon, D. (2006). *Optimal State Estimation*. Wiley.

Simon, H. A. and Chase, W. G. (1973). Skill in chess. *American Scientist*, 61:394–403.

Simon, H. A. and Gilmartin, K. (1973). A simulation of memory for chess positions. *Cognitive Psychology*, 5:29–46.

Sivak, M. (1996). The information that drivers use: Is it indeed 90*Perception*, 25:1081–89.

Sobel, I. and Feldman, G. (1973). *A 3x3 Isotropic Gradient Operator for Image Processing*. Wiley.

Southall, B. and Taylor, C. J. (2001). Stochastic road shape estimation. In *Proceeding International Conference on Computer Vision, ICCV*, volume 1.

Sun, R. and Sessions, C. (2000). Learning plans without a priori knowledge. *Adaptive Behavior*, 8(3-4):225–253.

Sutton, R. S. and Barto, A. G. (1981). Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review*, 88:135–170.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

Tahirovic, A., Konjicija, S., Avdagic, Z., Meier, G., and Wurmthaler, C. (2005). Longitudinal vehicle guidance using neural networks. In *Computational Intelligence in Robotics and Automation, 2005. CIRA 2005.*

Taylor, C. J., Malik, J., and Weber, J. (1996). A real-time approach to stereopsis and lane-finding. In *Proc. IEEE Intelligent Vehicles*.

Thompson, W. and Pong, T. (1990). Detecting moving-objects. *International Journal of Computer Vision*, 4:39–57.

Thorpe, C., Herbert, M., Kanade, T., and Shafer, S. (1991). Toward autonomous driving: the cmu navlab. i. perception. *IEEE Expert*, 6:31–42.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L. E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P. (2006). Stanley: The robot that won the darpa grand challenge:. *J. Robot. Syst.*, 23(9):661–692.

Togelius, J. and Lucas, S. (2006). Evolving robust and specialized car racing skills. In *Proc. CEC 2006. Evolutionary Computation IEEE Congress on*, pages 1187–1194.

Tsugawa, S. (1993). Vision-based vehicles in japan: the machine vision systems and driving control systems. In *Industrial Electronics, 1993. Conference Proceedings, ISIE'93 - Budapest., IEEE International Symposium on*.

Turk, M. A., Morgenthaler, D. G., Gremban, K. D., and Marra, M. (1988). Vits-a vision system for autonomous land vehicle navigation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(3):342–361.

Ulleberg, P. and Rundmo, T. (2003). *Safety Science*, 41(5):427 – 443.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Dolan, J., Duggins, D., Ferguson, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T., Kelly, A., Kohanbash, D., Likhachev, M., Miller, N., Peterson, K., Rajkumar, R., Rybski, P., Salesky, B., Scherer, S., Woo-Seo, Y., Simmons, R., Singh, S., Snider, J., Stentz, A., Whittaker, W. ., and Ziglar, J. (2007). Tartan racing: A multi-modal approach to the darpa urban challenge. *Darpa Technical Report*.

Volksbot (2000). online: http://www.volksbot.de.

von Holst, E. and Mittelstaedt, H. (1950). The reafference principle. interaction between the central nervous system and the periphery. in selected papers of erich von holst: The behavioural physiology of animals and man, london: Methuen. (from german) 1 : 1 39-73. *The Behavioural Physiology of Animals and Man*, 1:39–73.

Wang, R. F. and Spelke, E. S. (2002). Human spatial representation: insights from animals. *TRENDS in Cognitive Sciences*, 6:376–382.

Wang, Y., Shen, D., and eam Khwang Teoh (1998). Lane detection using catmull-rom spline. In *IEEE International Conference on Intelligent Vehicles*.

Wang, Y., Teoh, E. K., and Shen, D. (2003). Lane detection and tracking using b-snake. *Image and Vision Computing*, 22:269–280.

Wasserman, P. (1993). *Advanced Methods in Neural Computing*. New York: Academic Press.

Wehner, R., Michel, B., and Antonsen, P. (1996). Visual navigation in insects: coupling of egocentric and geocentric information. *J Exp Biol*, 199(1):141–146.

Widrow, B. and Stearns, S. D. (1985). *Adaptive signal processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Wolpert, D. M., Ghahramani, Z., and Jordan, M. I. (1995). An internal model for sensorimotor integration. *Science*, 269:1880–1882.

Zhang, T. and Tomasi, C. (2002). On the consistency of instantaneous rigid motion estimation. *International Journal of Computer Vision*, 46:51–79.

Ziebart, B. D., Maas, A., Bagnell, J. A. D., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Proceeding of AAAI 2008*.

Curriculum Vitae

Name	Irene Markelić
Geburtstag	Juni 1979
Geburtsort	Datteln, (Deutschland, NRW)
Staatsangehörigkeit	Deutsch

1999–2002	Vordiplom in Computervisualistik (Informatik mit Schwerpunkt Bildverarbeitung und Bildsynthese), Universtiät Koblenz-Landau, Deutschland.
2002–2005	Diplom in Computervisualistik (Informatik mit Schwerpunkt Bildverarbeitung und Bildsynthese), Universtiät Koblenz-Landau, Deutschland.
(2002–2003)	Auslandssemester in Informatik an der Universität Granada, Spanien.
(2005–2005)	Diplomarbeit und Tätigkeit als studentische Hilfskraft am Fraunhofer Institut, Sankt Augustin, Deutschland.
2006–jetzt	Doktorandin und Wissenschaftliche Mitarbeiterin am *Bernstein Center for Computational Neuroscience*, Georg-August-Universität Göttingen