Anticipatory Driving for a Robot-Car Based on Supervised Learning

Irene Markelić[1], Tomas Kulviĉius[1], Minija Tamosiunaite[2], and Florentin Wörgötter[1]

[1] Bernstein Center for Computational Neuroscience, University of Göttingen, Bunsenstrasse. 10, 37073 Göttingen, Germany
{irene,tomas,worgott}@bccn-goettingen.de
http://www.bccn-goettingen.de

[2] Vytautas Magnus University, Kaunas, Lithuania
{m.tamosiunaite}@if.vdu.lt
http://www.vdu.lt

Abstract. Prediction and Planning are essential elements of successful human driving, making them equally important for autonomously driving systems. Many approaches achieve planning based on built-in world-knowledge. However, we show how a learning-based system can be extended to planning, needing little a priori knowledge. A car-like robot is trained by a human driver by constructing a database, where look ahead sensory information is stored together with action *sequences*. From that we achieve a novel form of velocity control, based only on information in image coordinates. For steering we employ a two-level approach in which database information is combined with an additional reactive controller. The result is a trajectory planning robot running at real-time, issuing steering and velocity control commands in a human manner.

Keywords: anticipatory behavior, example based learning, robot car driving, longitudinal control, lateral control, learning from experience.

1 Introduction

Automated system control is important in industry and has many applications for everyday life. For example, autonomously driving cars have the potential to increase safety and reduce costs. In driving, planning plays an important role. Look ahead information helps us decide which actions to take in response to upcoming events. We can either act immediately or prepare ahead of time for taking certain actions, thus reducing reaction time. For this reason we propose that an autonomously driving car should also be equipped with such capabilities as using look ahead and plan making, which is what we call anticipatory driving. The advantages are that it can a) react to upcoming events, b) cope with short lacks of sensory information, and c) use these plans for making predictions about its own state, which is useful for higher-level planning. For a more thorough list of the advantages of action sequence generation in general, see [1].

In this paper we focus on the task of lane following, which is a basic skill in autonomous driving. Lane following is a visuomotor skill, i.e. one in which visual sensory input must be transferred into appropriate motor output. Many approaches rely on predefined control laws which require a map of the environment in Cartesian coordinates, the known state of the plant, and possibly the known states of other entities, e.g. other cars. Thus, the work consists of a) identifying the necessary control law(s), b) identifying the model for the plant as well as other desired object models, and c) transforming the (relevant) visual input into the required map. In this type of approach most knowledge, such as *what* is expected to be sensed (object models), *how* it is sensed (sensor models), and how to *act* upon it (control laws), is built int the system a priori. Examples of approaches matching this description to a great extent are [2,3,4,5]. We refer to them as "model-based". They have the property that forward simulations of the system are possible by using the state estimation process and the control law(s). This can be used for planning algorithms like path planning. Despite many advantages, the massive dependency on built-in world-knowledge presents a bottleneck. Everything that a system might need to act upon cannot be predefined.

As alternatives, machine learning based approaches to lane following can be implemented. A prominent example is ALVINN [6,7,8,9], where actions of a human driver were associated with concurrent visual input from a camera via a neural network. The inputs to the network were the pixel values of the (downscaled) camera image and the output was an appropriate steering angle. Velocity control was handled by a human. Two important points to note are: 1) the system learned to take the correct actions not by explicit control laws and state estimators, but instead based only on the provided examples, and 2) no transformation of the visual input into another representation was required, thus no conventional image processing (e.g. feature extraction, or reconstruction of 3d-information) was necessary since the visual input was directly mapped to a motor command. Further examples of machine learning based approaches are: [10] using reinforcement learning, and [11] using genetic algorithms. We refer to these approaches as "learning-based". A shortcoming of these systems is the lack of an explicit mechanism for planning, making them dependent on continuous sensor input.

Our goal is to utilize the most advantageous quality of the learning-based approaches, i.e. not having to rely on built-in knowledge, and to extend the method with an explicit planning component. Path planning in model-based approaches can be achieved by using the Cartesian map of the environment, a model of the system to be controlled, and a control law. How can one plan a path if all these items are not available? We solve this problem by equipping our system with very simple mechanisms, that are thought to play a role in human learning, too. Precisely, we give it the ability to make associations and to store and retrieve data (memory). First, a reactive controller is obtained from human control data, by associating visual information concerning the nearby street trajectory with a steering command. Second, a planner learns to associate visual information about the entire observable street trajectory with action sequences. We show how this leads to robust lateral and longitudinal control of the robot, and how it also works in open-loop situations, i.e. when no sensory input is available. Our goal is not to compete with current state-of-the-art autonomous driving systems, which are quite advanced and also generally make use of many additional sensors besides visual ones, instead, we intend to present an alternative to many current approaches relying on task-specific knowledge.

As described our system is capable of generating speed control as well. This concept has been much less investigated than steering, at least for approaches that do not make use of environmental maps, for which a sensor model is necessary. Simpler controls also exists, such as Adaptive Cruise Control (ACC) systems, which use radar or laser to slow down the vehicle when detecting an obstacle in front, or Intelligent Speed Adapters and Limiters, ISAs and ISLs [12], which adjust, or limit, a vehicle's speed according to the given mandatory limits. Other approaches determine speed, with the help of a leading vehicle [13]. More related to this work are [14] and [15], which employ fuzzy neural networks trained on human control data to anticipate curves and regulate speed accordingly. In contrast to our approach, that work was done using simulations, and single actions per timestep were generated instead of action plans.

The structure of the paper is as follows: In the Experimental Setup section we describe the means for realizing this approach. In the Methods section we explain planner, reactive controller, and their combination, followed by their evaluation in the Results section. In the Discussion section we discuss our work and shortly compare it to predictive control based on Kalman filtering [16].

2 Experimental Setup

Experiments are carried out in an indoor environment on a four-wheeled robot (a modified VolksBot [17] of 50 cm x 60 cm size) with two motors, one for driving the wheels on each side (differential steering). The robot is equipped with a monochrome firewire camera operating at approx. 20 Hz, see. Fig. 1A. The laboratory setup simulates a street environment, where the driver can control the robot from a special station, see Fig. 1B. Here, one can see "through the robot's eyes" by means of a TV on which we display the camera output. The driver can manipulate the robot's actuators using a steering wheel and pedal set where the communication between human control output and robot sensory input is realized via a peer-to-peer architecture. A laptop placed on the robot, is connected to the camera and motors. In a cyclical fashion the robot acquires a camera image, sends it to the TV, and waits for a control input from the desktop computer connected to the steering wheel and pedal set. The control, or action input, is a steer and a velocity command, for which we use the following notation: a^{st} denotes the steer signal, and a^{v} velocity. Both signals take numerical values with $a^{st} \in [-128, 128]$ related to the steering angle and $a^{v} \in [-512, 512]$ related to the voltage sent to each motor. Throughout this text, we skip the superscripts *st* or *v*, when referring to both action signals. They are generated by the human and sent to the robot-laptop via the desktop computer, which in turn passes them to the motors of the robot, (see Fig. 1C). Thus, every communication cycle defines a discrete timestep t where every incoming image frame I_t is related to the corresponding control a_t^v, and a_t^{st}. In Fig. 1D, we show a sketch of the track on which we trained the robot.

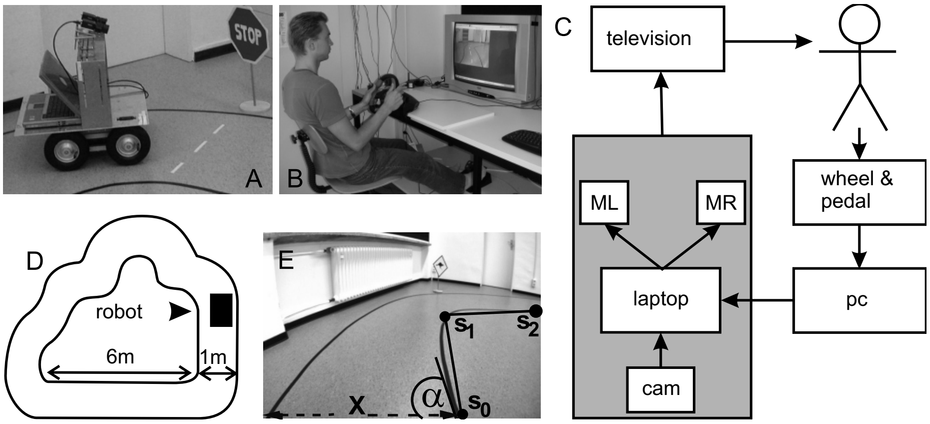

Fig. 1. A: A car-like robot. B: Control station. C: Information flow in the experimental setup: cam denotes camera, and ML and MR left and right motor, the shaded area indicates the robot. D: Sketch of the track used for training the robot. E: Short and long term visual information. x and α define short term information for the reactive controller and s_0, s_1, s_2, the corner points of the polygonized lane boundary, define the long term information for the planner.

During the supervised learning the robot associates visual information with human actions. This visual information is derived from the right street lane boundary that we detect in each image in real-time. We developed a simple and fast algorithm based on conventional edge detection (Canny [18]) which returns the detected boundary as an ordered 2d-curve.

3 Methods

Regulation of steering and speed are necessary for vehicle control. Steering control is considered to be a two-level process [19] using short-term and look ahead information, whereas we assume speed control to be based only on look ahead information. 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. As explained we use two modules, a reactive controller (RC) and a planner, where the former maps short-term information to a single steering control value, and the latter is in charge of processing look ahead information and generating action plans, i.e. sequences for steering and speed control. The final steering command is a combination of planner and RC output. This setup is visualized in Fig. 2. In the following we describe both modules starting with the reactive controller.

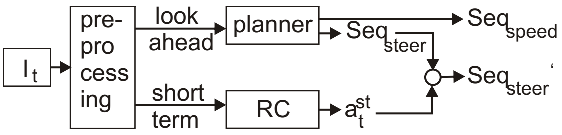

Fig. 2. The system setup. I_t denotes the image frame at time t and Seq a sequence of actions.

3.1 The Reactive Controller

The purpose of the reactive controller is two-fold. It is supposed to correct the planner if necessary, and it is used in the case that no sufficiently well suited plan is contained in the database. It is also learned from human actions and designed as follows: We define the immediate future (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 x-axis at the bottom line of the image, see Fig. 1E. To acquire the supervisor's policy with respect to these parameters we assign human actions (from the training set) to the state space (see 3A). To fill the empty spaces, generalizing to unknown situations, we use k-nearest neighbor, shown in 3B. Of course, other approximation methods can be used instead. Note that this simple approach results in an extremely fast controller, only requiring the time necessary for looking up a steering signal in a matrix.

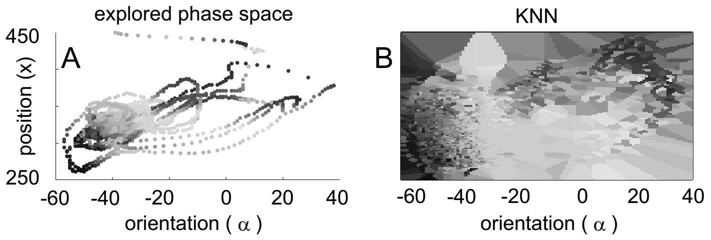

Fig. 3. A: The acquired policy from the supervisor. B: The interpolated policy using k-nearest neighbour. Different gray values denote different steering angles.

3.2 Planner

We follow the idea that a system should be able to associate experienced action *sequences* with visually perceived situations. When exposed to similar situations it should remember the previously conducted action maneuver. For example, if the system observes a right turn, it should remember that in the past it always conducted a similar sequence of actions after it had seen the right turn. Thus, right turns should be directly associated with right turns in the action space. Even if this associated action plan is not completely exact, for example the steering amplitude would not be exactly correct for taking the turn, it still provides guidance in the desired direction. We realized this idea by building a database, wherein the system stores triples containing a perceived situation description along with the corresponding sequence of steering and velocity actions. When driving in autonomous mode, the system queries the database with the currently perceived situation and receives (remembers) the assigned action plans. Based on these retrieved plans, it computes current and, if necessary, future actions. Thus, the following steps are necessary: a) database construction, b) database query at runtime, and c) control sequence calculation at retrieval time.

a) For the database construction a visual state or situation description, $\boldsymbol{s}$, is needed, comprising look ahead information. For that purpose we use a polygonized approximation of the right street lane, such that $\boldsymbol{s} = [s_0, s_1, ..s_l]$, where s_i, with $0 \leq i \leq l$, are the corner points of the polygon. The polygonization is done using the Douglas-Peucker method [20]. Note that the vertices of the vector $\boldsymbol{s}$ are ordered, i.e. s_0 is the first vertex at the bottom of the image and s_l describes the last vertex on the 2d-curve. The vector length l can vary. An example is shown in Fig. 1E and 4. It is a rough description of the observed street which contains look ahead information, but not explicitly extracted information like curvature or path length.

To each $\boldsymbol{s}_t$ corresponding control sequences are assigned. Control sequences are ordered series of actions, $Seq_{steer} = [a^{st}_t, a^{st}_{t+1}, ..a^{st}_{t+n}]$, and $Seq_{speed} = [a^v_t, a^v_{t+1}, ..\ a^v_{t+n}]$. The length n of a given sequence is supposed to resemble the number of actions that are executed while following the observed trajectory at a given timestep. That is, if only a short stretch of the street is visible we only assign a short action sequence to it and vice versa. Since we do not know exactly how many actions correspond to the observed street we use the experimentally determined value:

$$n = \lfloor \frac{1}{8} \sum_{i=1}^{l} |s_{i-1} - s_i| \rfloor . \qquad (1)$$

A triple $(\boldsymbol{s}_t, Seq_{steer}, Seq_{speed})$ is stored in the database, unless a similar entry is already available, (i.e. $\epsilon \leq 10$, see below and equation 2). The database is complete if a predefined number of entries is reached, or no more triples are added by the routine. We denote the total amount of database entries as K. Thus, Seq^k_{steer}, with $1 \leq k \leq K$ is the steering sequence of the $k'th$ database entry. If we are referring to Seq_{steer} and Seq_{speed} interchangeably we skip the subscripts.

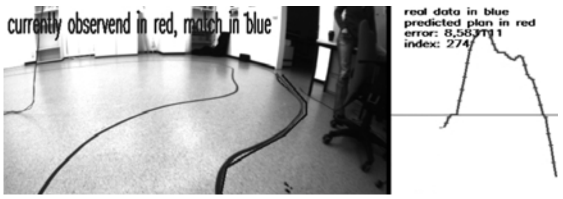

Fig. 4. Screenshot example of the planner operating mode. Left: The observed street (gray, originally red) is compared to the database entries and the best match is returned (black, originally blue). Right: The assigned steering sequence of the best match.

b) For the retrieval step, we need a metric to determine the difference ϵ between the extracted vectors, $\boldsymbol{s}$, which describe the street ahead. We use a weighted euclidean distance between vectors of same length l, normalized by l. The weighting enforces similar curves to be those that are especially similar in the beginning, which is the part of the street that is closest to the robot:

$$\epsilon = \frac{1}{l} \sum_{i=0}^{l} \boldsymbol{\omega}_i \sqrt{(\boldsymbol{s}_{q_i} - \boldsymbol{s}_{db_i})^2}, \tag{2}$$

where $\boldsymbol{s}_{q_i}$ denotes the i'th element of the queried vector and $\boldsymbol{s}_{db_i}$ the i'th element of a vector in the database, $\boldsymbol{\omega}$ is a vector containing weights where $\boldsymbol{\omega}_{i+1} < \boldsymbol{\omega}_i$ (we used 20, 10, 5, 5 for the first four $\boldsymbol{\omega}$ entries and 1 for all remaining ones). Equipped with such a database, the robot can use its current visual input for making queries. The return values are: 1) the difference ϵ to the best found match and 2) Seq_{steer} and Seq_{speed} that were assigned to it.

c) The action sequences from the database retrieval contain valuable information, not only for the current timestep t but also for $t+1$, $t+2$, ..., $t+n$. However, the database output as such only corresponds to the observed street to a certain degree. How can we drive on unknown streets? Even on the same track it is unlikely that identical images are retrieved multiple times. In other words, how can we generalize using the database output? Here, we postpone this generalization step until retrieval time which is typical for lazy-learning algorithms ([21,22,23,24]).

In principle there are two different ways in which the action sequences obtained from the database query can be used:
1) an action plan can be computed based on *single* retrieved sequences, or
2) based on *all* (or the latest N) retrieved sequences.

For the former we propose a method that we refer to as DIFF, because it is based on a difference equation, and for the latter a method that we refer to as AVG, because it is based on simple averaging. We will find that both methods yield comparable results, and because AVG is much simpler to implement we will only use this method later. Even so, we believe that chaining single action sequences together is an important concept that should be considered as well. For this reason we include a description of the DIFF method.

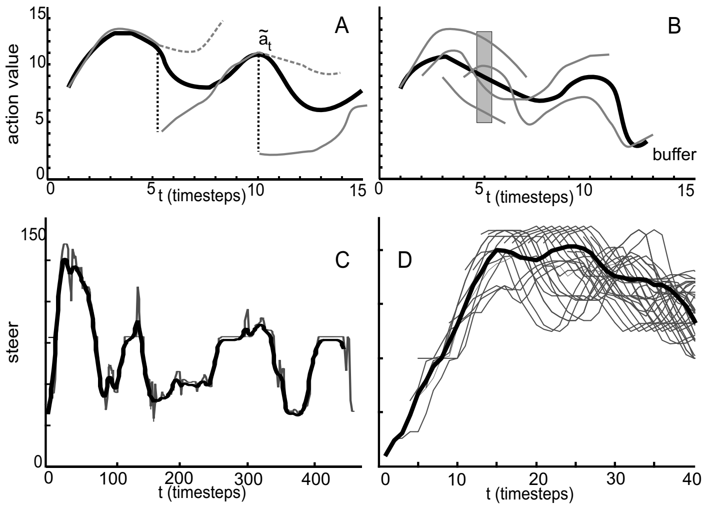

Fig. 5. A) and B) visualize schematically the data DIFF (A) and AVG (B) operate on. A) The solid gray lines denote parts of single retrieved action sequences that are used until a better match is found, which is indicated by the vertical slashed lines. As can be seen, there is no smooth transition from one retrieved action sequence to the next. The proposed difference equation smoothly joins two sequences together by taking into account future values from the previous sequence, which are depicted by the gray slashed line segments. In the equation we refer to these as $\tilde{a}$. As an example, we took $t = 10$ as the current timestep. Thus, $\tilde{a}_{t+i}$ denotes the action value from the previous sequence i timesteps ahead. The resulting action plan is drawn as the thicker black line denoting the function $a(t)$. B) The AVG method uses values from the last N retrieved action sequences, which must be held in a buffer and which are drawn as thin gray lines. The new action sequence is computed by simply taking the average from the action values in the buffer at each timestep, indicated the drawn rectangle, which denotes the vector v (see text). C) and D) show real data examples for DIFF (C) and AVG (D). Again, the thin gray lines denote retrieved sequences and the thick black line denotes the new sequence returned by each method.

As explained the DIFF method computes an action plan based on single action sequences obtained from database queries. At every timestep a database query is conducted and the returned sequence is compared to the one obtained in a previous timestep. If it is better, i.e. the affiliated error ϵ, which is returned together with the sequence by the database, is smaller than the error affiliated with the previous sequence, then the new result is kept and the old sequence is discarded and vice versa. (To acknowledge that a good match found a few timesteps ago, is less well suited at the current moment, we discount previous sequences by adding a discount factor, $\lambda = 5$, to their affiliated error.) The concept is visualized in 5A, where the gray line segments denote current action sequences. It can be seen that whenever a better match is found, there is a gap between two consecutive signals, indicated by the vertical, dashed lines in the figure. The purpose of the DIFF method is to create a smooth transition between two such signals by taking the future values (the dashed gray line segments in the figure) of the previous signal into account. The action plan is given by a_t as shown in 3 and 4.

$$a_{t+1} = a_t + \Delta a_t \tag{3}$$

$$\Delta a_t = \sum_{i=0}^{n-1} \alpha_i \frac{\tilde{a}_{t+i\tau} - a_t}{(1 + \frac{a_t}{a_{max}})G}, \tag{4}$$

where a_t is representing a steering or velocity command, i.e. either a_t^{st} or a_t^{v}, and n is the length of the sequence currently being used. We denote action values from sequences from the database retrieval at the current timestep with $\tilde{a}_t$, see Fig. 5A. Thus, future values, i.e. those values in the sequence at $t+1$, $t+2$, $t+..$, $t+n$ are given by $\tilde{a}_{t+i}$. The variable τ is a constant determining the sampling frequency on the current sequence. It influences how fast to move from one signal to another. If a low value is chosen, the resulting control sequence lingers longer in the vicinity of the previous segment before reaching the values of the new segment and vice versa. From the training data we know that the human used steering and speed commands that did not exceed a certain amplitude. These upper and lower limits, which we denote with a_{max} should not be exceeded by the system either. Hence, the denominator decelerates the growth of the action plan function if the previous action was already close to these known limits. It is determined by the constant G, which we set to 10, and $\alpha_i = e^{\frac{-i^2}{\sigma^2}}$ a decay term, which discounts the influence of future values given by $\tilde{a}$. The σ is a constant, which we set to 4. In Fig. 5A the action plan a_t is drawn in black. In 5C the result of the difference equation is shown for real data.

We now turn to the second method, AVG, which also makes a query every timestep, but in contrast to DIFF keeps the returned sequence of each retrieval in a buffer. To determine an action value at a given timestep, we simply compute the average on the action values from the latest N retrievals, which are contained in a vector $\boldsymbol{v}$ as shown in the figure. The value N is usually also the number of values that the average is computed from. Thus:

$$a_t = \frac{1}{|v|} \sum_{i=0}^{|v|-1} v_i. \tag{5}$$

An example for this method is shown in Fig. 5B, and a result computed on real data in Fig. 5D.

3.3 Combination of Planner and Reactive Controller

The next step is the combination of RC and planner. The RC should correct the planner in critical, i.e. unfamiliar situations. Therefore, a measure is needed that informs about this state. We find that an appropriate measure is the error ϵ returned from the database query. If no sufficiently good match to the currently observed image is contained in the database the system performance decreases and this correlates with the value of ϵ.

We can now combine RC and planner as a function of ϵ, $f(\epsilon) \rightarrow \omega_c$ with $0 \leq \omega_c \leq 1$. The smaller ϵ the more we want to rely on the planner, the larger ϵ the more we consider RC output:

$$steer = \omega_c * RC + (1 - \omega_c) * planner, \tag{6}$$

$$\text{with } f(\epsilon) = e^a, \text{and } a = \frac{(\epsilon - \epsilon_{tolerable})}{150}, \tag{7}$$

where we set $\epsilon_{tolerable}$ to 700.

4 Results

To test the algorithms DIFF, AVG and RC, training and test data is produced from eight laps of human driving on the described track in the lab (always in the same direction). The database is constructed from five of these laps and the remaining data is used as test set. First we consider the performance of a single action generation for the current timestep, i.e. a_t. For velocity prediction with AVG we found best results when averaging over the last $N = 20$ buffer entries and for steering the last $N = 10$. The result is shown in Fig. 6A-E. It can be seen that all three methods capture the human behavior, where AVG and DIFF give smooth output and RC is comparably jerky.

We further compare the methods by plotting the root of the summed squared error between algorithmic output and human signal, ($error = \sqrt{(algorithm_{out} - human_{out})^2}$). This error and confidence interval (95%) are plotted in Fig. 7A for steer and 7B for speed. It can be seen that there is little difference between AVG and DIFF. The higher 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. This is understandable because there is more variance in the human velocity data than in the steering data. Consider for example the velocity plot in Fig. 6B or C between timesteps 300 and 500 on the x-axis. The depicted speed signal in this intervall can be considered to be constant, however, the small deviations between human and synthesized signal accrue to a relatively large error.

As this is a quantitative comparison, it is necessarily offline, and does not prove that the system behavior would also be acceptable if the controllers were used inside the closed-loop setup, i.e. when the generated action of the controller 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, i.e. it stays on the track. The jerkiness of the RC output also results in a jerky lateral behavior. However, due to the inertia of the robot it is less strongly visible than what could be expected from the plotted signal.

Next we test wether or not the combination of planner and RC indeed improves the system performance as supposed. In case of an unfamiliar street environment that is not represented in the database, 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 our setup track is circular the robot

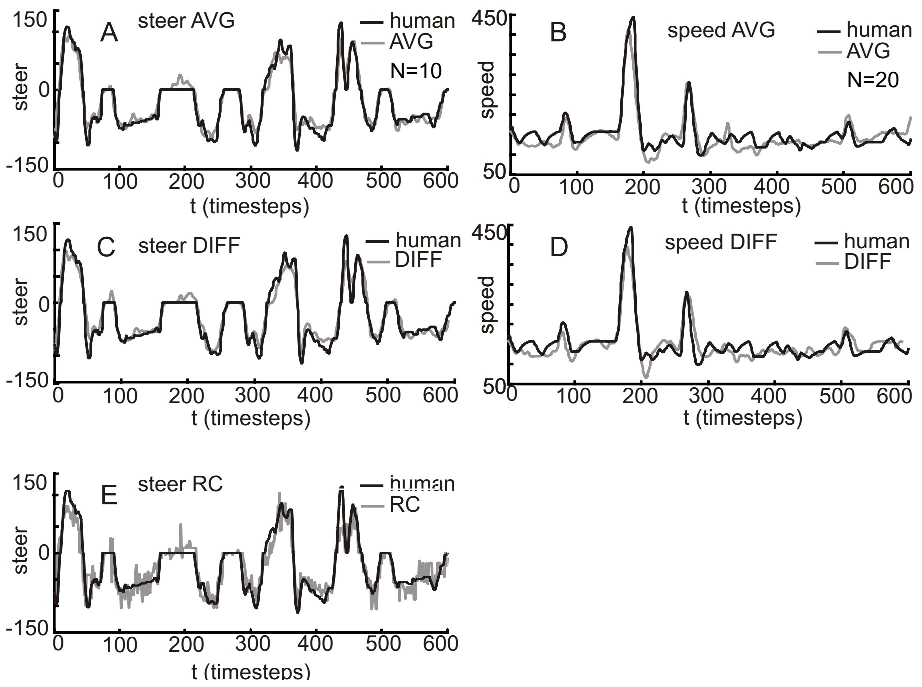

Fig. 6. A and B: Performance of AVG on generating a_t^{st} and a_t^v. "N" is the amount of entries in the buffer over which was averaged. C and D: Performance of DIFF on generating a_t^{st} and a_t^v. E: Performance of RC on generating a_t^{st}.

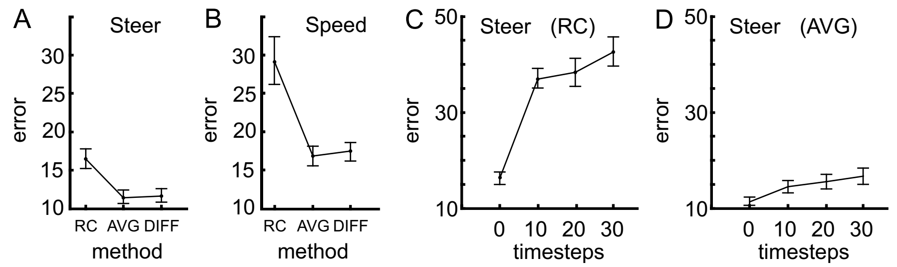

Fig. 7. A: Comparing the performance of AVG, DIFF and RC for steering generation for a_t. The plotted error is the root of the summed squared difference between the human action signal and the signal generated by each method. B: Comparing the performance of the methods for speed generation. C: The quality of steer predictions of RC for t, t+10..t+30 timesteps ahead. D: The quality of steer predictions of AVG for t, t+10..t+30 timesteps ahead. As expected, the error for AVG is much less than for RC, which indicates the capacity of AVG for action prediction.

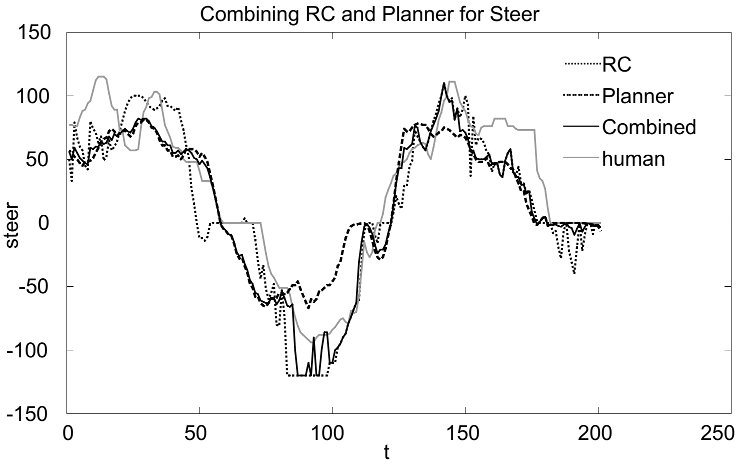

Fig. 8. Comparison of the combined signal to RC, Planner, and human output, where the robot was driven by the human. At around $t = 100$ it can be seen how the combined signal is better than the Planner output by being drawn closer to the RC signal.

is 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 not part of its database. For a first evaluation we let the human drive the 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. 8, where at around timestep 100 on the x-axis 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 signals from the planner indicate 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.

As a second evaluation we let the robot drive on the unknown track using a) only the planner, b) only RC, and c) the combined signals. With the planner it drives smoothly but looses the track in difficult (high curvature) turns due to the explained reason that this situation is not represented in its database. Using RC it is able to stay on track as expected, however, the behavior is less smooth. Finally, when using the combination it drives smoothly on the known parts, which constitutes the majority of the encountered situations, and in addition it manages to stay on the track even during the described difficult turns. To evaluate the performance of the system concerning sequence prediction, which

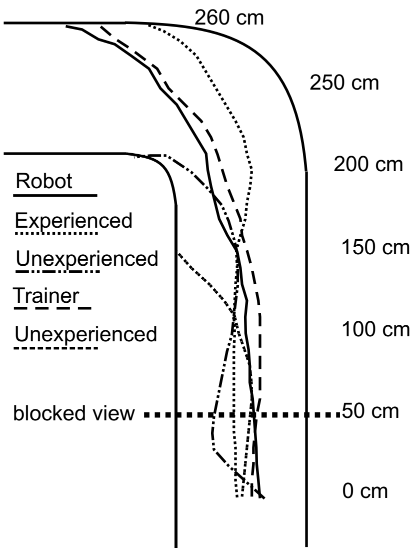

Fig. 9. Top view on part of the track. Shown is the driven trajectory of the drivers and the robot, sampled every 10 centimeters. The horizontal line denotes where the view was blocked.

is our main interest, we do not consider the DIFF method but only AVG, since DIFF is more complicated with more parameters to tune than AVG, yet fails to lead to significantly better results in generating single actions as shown above. Again we test quantitatively and qualitatively.

For the quantitative evaluation we apply AVG on the test set 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 test set and plot it in Fig. 7D. We also included RC[1] in this plot, mainly for comparison. This result is shown in Fig. 7C. 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 qualitative testing we abruptly blocked the human controller's view during driving. This can be interpreted as a short sensor "black-out", which might occur due to technical problems. We then measure the number of timesteps the human was able to stay on the street without visual feedback. For that we only let the human control steering. The speed signal is set to a constant value uninfluenced by the driver, (during human performance, not for the robot). This is done

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

because the drivers stop the robot during the experiment as soon as they cannot see the street anymore. Furthermore, we decide to block the view shortly before a curve, requiring a real change in actions. We repeat this with three more drivers: two are not trained in driving the robot, one intermediate driver, and the expert, who also generated the training data set for the robot. The result is shown in Fig. 9. It can be seen that the robot does perform the turn, which means that it successfully uses its generated plan and executes it it similarly to the trainer. It also shows that the less well trained humans lose the track quicker than the robot.

5 Discussion

We presented a robot-car that learns anticipatory driving from a human supervisor and visual sensory data. Anticipatory means that it learns to generate action sequences and to react to upcoming events, which is necessary for velocity control (e.g. speed must be decreased when approaching sharp turns). It runs at real-time and issues steering and velocity controls in a human-like way. Its planning capability allows it to cope with missing visual input.

In contrast to many current approaches to vehicle control, which are mostly model-based, very little a priori knowledge was required. Instead the system achieved its behavior by being equipped only with mechanisms for associative learning and memory.

First, a reactive controller associated short-term visual information with single actions from a human teacher. Following the idea that a system that repeatedly executes similar action sequences after observing similar images should be able to also associate these things, a planner learned to correlate observed street trajectories with subsequently performed action sequences. During performance the combined signal between reactive controller and planner was shown to lead to robust lane following behaviour. As described in the Results section, the combined signal is jerkier when relying on RC in unknown situations and smoother when using the planner in well known situations. This appears natural when considering that humans also produce smoother action sequences (in dancing for example) after training. In particular, it was not necessary to build a map of the environment from the visual sensor input to acquire action plans. Visual information could be processed directly in image coordinates. No sensor model was needed, thus it was not necessary to know the camera geometry or to undistort image frames. This makes this approach easy to implement and to use.

Concerning velocity control this work makes a novel contribution with respect to the work in autonomous driving that is not based on constructing environmental maps, namely the ability of the system to generate speed control based on the visually perceived upcoming curves.

Since this work is related to predictive control, we shortly compare it to methods usually used in this context. All of the cited model-based work in the Introduction uses state estimators for generating action control. As state estimators, a variant of the Kalman filter [16] is often used. Such a filter requires knowledge about the state-transition probability of the system, i.e. it must be known how the system's state changes under the influence of actions or time. Based on this, and possibly also on knowledge about the sensing process, a probable future state can be predicted. Then actions can be chosen with regard to the predicted future state of the system. If this is done repeatedly (like a mental simulation), action sequences can be obtained. The main difference between this way of achieving predictions and our method is that we skip the state prediction. We generate action predictions not by inferring them from a predicted state, but by memorizing entire sequences. We see two advantages in that: 1) it is faster, simply because the step of state generation is not necessary; 2) it is less prone to error, because fixed sequences are stored and do not have to be generated step by step based on predicted states that get more and more erroneous. Of course, not being able to predict future states is a disadvantage. For example, we cannot link multiple sequences together, which would be possible if we knew the state of the system after the execution of an action sequence. However, this approach could be extended to also predict future states.

Acknowledgments. This work was supported by the European Comission grant DRIVSCO.

References

1. Sun, R., Sessions, C.: Learning plans without a priori knowledge. Adaptive Behavior 8(3-4), 225–253 (2000)
2. Dickmanns, E.D., Graefe, V.: Dynamic monocular machine vision. Machine Vision and Applications 1, 223–240 (1988)
3. Turk, M.A., Morgenthaler, D.G., Gremban, K.D., Marra, M.: Vits-a vision system for autonomous land vehicle navigation. IEEE Trans. Pattern Anal. Mach. Intell. 10(3), 342–361 (1988)
4. 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., Mahoney, P.: Stanley: The robot that won the darpa grand challenge. J. Robot. Syst. 23(9), 661–692 (2006)
5. 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.R., Ziglar, J.: Tartan racing: A multi-modal approach to the darpa urban challenge. Darpa Technical Report (2007)
6. Pomerleau, D.: Alvinn: An autonomous land vehicle in a neural network. In: Advances in Neural Information Processing Systems, vol. 1, Morgan Kaufmann, San Francisco (1989)
7. Pomerleau, D.: Efficient training of artificial neural networks for autonomous navigation. Neural Computation 3(1), 88–97 (1991)
8. Pomerleau, D.: Neural network based autonomous navigation. In: NAVLAB 1990, pp. 558–614 (1990)

9. Pomerleau, D.A.: Neural network vision for robot driving. In: The Handbook of Brain Theory and Neural Networks. M. Arbib (1999)
10. Riedmiller, M., Montemerlo, M., Dahlkamp, H.: Learning to drive a real car in 20 minutes. In: Proc. Frontiers in the Convergence of Bioscience and Information Technologies, FBIT 2007, pp. 645–650 (2007)
11. Togelius, J., Lucas, S.: Evolving robust and specialized car racing skills. In: Evolutionary Computation IEEE Congress on Proc. CEC 2006, pp. 1187–1194 (2006)
12. Brookhuis, K., de Waard, D.: Limiting speed, towards an intelligent speed adapter (isa). Transportation Research Part F: Traffic Psychology and Behaviour 2, 81–90 (1999)
13. Tahirovic, A., Konjicija, S., Avdagic, Z., Meier, G., Wurmthaler, C.: Longitudinal vehicle guidance using neural networks. In: Computational Intelligence in Robotics and Automation, CIRA 2005 (2005)
14. Partouche, D., Pasquier, M., Spalanzani, A.: Intelligent speed adaptation using a self-organizing neuro-fuzzy controller. In: Proc. IEEE Intelligent Vehicles Symposium, pp. 846–851 (2007)
15. Kwasnicka, H., Dudala, M.: Neuro-fuzzy driver learning from real driving observations. In: Proceedings of the Artificial Intelligence in Control and Managamnent (2002)
16. Kalman, R.E.: A new approach to linear filtering and prediction problems. Transaction of the ASME Journal of Basic Engineering, 33–45 (1960)
17. Volksbot (2000), `http://www.volksbot.de`
18. Canny, J.F.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Machine Intell. 8, 679–698 (1986)
19. Donges, E.: A two-level model of driver steering behaviour. Hum Factors 20, 691–707 (1978)
20. Douglas, D.H., Peucker, T.K.: 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 (1973)
21. Aha, D.W. (ed.): Lazy Learning. Artificial Intelligence Review, vol. 11, pp. 7–10. Kluwer Academic Publishers, Dordrecht (1997)
22. Bottou, L., Vapnik, V.: Local learning algorithms. Neural Computation 4, 888–900 (1992)
23. Santamaria, J.C., Sutton, R.S., Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces. Adaptive Behavior 6, 163–217 (1997)
24. Smart, W.D., Kaelbling, L.P.: Practical reinforcement learning in continuous spaces. In: Proceedings of the Seventeenth International Conference on Machine Learning (2000)