Anticipatory Driving for a Robot-Car Based on Supervised Learning

Irene Markelić[1], Tomas Kulvičius[1], Minija Tamošiunaite[2], and Florentin Wörgötter[1]

[1] Bernstein Center for Computational Neuroscience, University of Göttingen, Bunsenstrasse. 10, 37073 Göttingen, Germany
{irene,tomas,woergoetter}@bccn-goettingen.de
http://www.bccn-goettingen.de
[2] Vytautas Magnus University, Kaunas, Lithuania
{m.tamosiunaite}@if.vdu.lt
http://www.vdu.lt

Abstract. Using look ahead information and plan making improves human driving. We therefore propose that also autonomously driving systems should dispose over such abilities. We adapt a machine learning approach, where the system, a car-like robot, is trained by an experienced driver by correlating visual input to human driving actions. The heart of the system is a database where look ahead sensory information is stored together with action *sequences* issued by the human supervisor. The result is a robot that runs at real-time and issues steering and velocity control in a human-like way. For steer we adapt a two-level approach, where the result of the database is combined with an additional reactive controller for robust behavior. Concerning velocity control this paper makes a novel contribution which is the ability of the system to react adequatly to upcoming curves.

Key words: 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 every day 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 help us decide which actions to take in response to upcoming events. We can either act immediately or prepare for taking certain actions, which reduces reaction time. For this reason we propose that an autonomously driving car should also dispose over such abilities as using look ahead and plan making. The advantages are that it can a) react to upcoming events and b) can cope with short lacks of sensory information and c) could use these plans for making predictions about its own state which can be used for higher-level planning. For a more thorough list of the advantages of action sequence generation in general see [1].

A conventional approach to autonomously driving cars using cameras as passive distal sensors is to extract as much relevant information as possible from the data stream. With this information a model of the environment is constructed wich is used to calculate optimal trajectories by applying laws of physics.

Where this is a precise and successfully used strategy, (e.g. [2, 3]) it also comes with some flaws. a) For the model construction all information must be built in a priori, thus the designer must consider all contingencies. b) The model construction is computationally expensive even with modern hardware, but a fast driving vehicle must process information in real time.

Therefore, a model-free approach seems attractive and as a consequence numerous machine learning mechanisms have been applied to this problem e.g.: artificial neural networks (ANNs) [4], reinforcement learning [5], genetic algorithms [6], neuro-fuzzy controller [7], and case based reasoning [8], to name but a few. Often artificial neural networks are used with a supervised learning algorithm [4, 9, 10], where there is a trend towards the usage of neuro-fuzzy controllers. Supervised learning in this case is convenient because it is often easy to produce large sets of training data from an expert. Although not all applications make use of this, ANNs can in principle solve two problems. a) they can perform feature selection *and* extraction (in other words they make conventional computer vision obsolete), and b) they can approximate control laws, by establishing a functional relationship between in- and output from training data. One very successful application in the autonomous driving domain is Alvinn (Autonomous Land Vehicle in a Neural Network) [4]. The pixel values of a downsampled camera image were fed into a multilayer network that was trained using a backpropagation algorithm. It was implemented for steer control in a real car and the performance was demonstrated by a 90 mile drive where velocity was controlled externally and steering by the network. One difficult issue in this project, which is inherent to all learning-from-human-exemplars scenarios, was how to deal with unknown situations. Since during training the system is only shown how to do the "right" thing some situations of the state space are never encountered. If the system in autonomous mode for some reason faces such a situation it must extrapolate from the training data, which leaves unclear if an appropriate action can be generated. In the Alvinn approach this problem was treated by artificially producing data for "bad" situations, i.e. being off the road, and training the network with these bad exemplars. This data was obtained using image projections, so the system would see the scenary as if being off the road, and the steering value was accordingly shifted in order to bring the car back to the center of the road.

In this paper we also use supervised learning for teaching a robot-car to drive, but we equip the system with the ability to predict sequences of actions. According to the taxonomy of [11] this work is most related to the category of *sequential decision making*, which is defined by choosing actions $a_j, .., a_{j+k}$ according to the last previous situation(s), $s_i, .., s_{i+j}$ and a goal state, G, where $k > 1$ and (in our case) $i = 0$, but see Discussion. We achieve this in a very simple way which requires only little image preprocessing, and thus a priori knowledge.

The approach is based on a database construction and lazy-learning [12]. The underlying idea is that a system that repeatedly perceives visual input followed by certain actions (from an expert) should be able to correlate this [13]. But instead of linking only one action to the visual input we use many of them. Concerning steer we also face the above explained problem of having to deal with unknown situations. Instead of intricately synthesizing data, which also puts additional constraints on the architecture (it requires camera calibration), we adopt a two-level approach, where we combine an additional reactive controller with the plan constructed from the database output. The process that generates this plan is referred to as planner. This improves robustness of the system and is justified by the assumption that human steering is a combination of using look ahead and reactive control [14].

The structure of the paper is as follows: In the section Experimental Setup we describe the means for realizing the approach. In Methods we explain planner, reactive controller and their combination, in Results we show the system performance and we discuss our approach in Conclusion.

2 Experimental Setup

Experiments are carried out in an indoor environment on a four-wheeled robot (a modified VolksBot [15] of 50 cm x 60 cm size) with two motors, one for driving the wheels on each side, thus, using differential steering. The robot is equipped with a monochrome firewire camera operating at approx. 20Hz, comp. 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 motors and camera. In a cyclewise fashion it acquires a camera image, sends it to the TV, and waits for a control input from a 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, if 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, (comp. 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^{v}_{t}, and a^{st}_{t}. In Fig. 1D we show a sketch of the parcours on which we trained the robot.

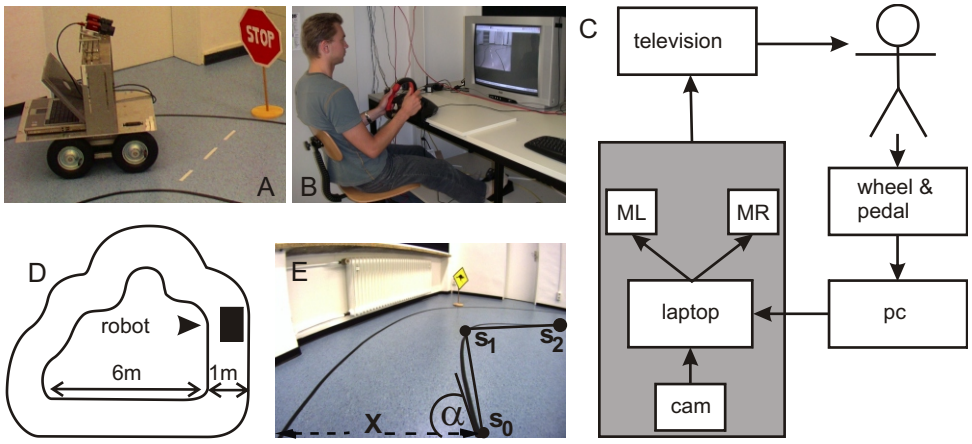

Fig. 1. A: A car-like robot. B: Control station. C: Information flow in the experimental setup: cam denotes camera, and ML and MR left and right motor, the shaded area indicates the robot. D: Sketch of the track used for training the robot. E: Short and long term visual information. x and α are defining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

3 Methods

For vehicle control steer and speed regulation is necessary. Steer control is considered to be a two-level process [14] using short-term information and look ahead, and we assume that speed control is based on look ahead information. We use the word "short-term" to denote relevant visual information that is temporally close to the vehicle; it also means being physically close to it; and "look ahead" for visual information that is relevant in the future, i.e. further away from the vehicle. As explained we use two modules 1) planner and 2) reactive controller (RC), where the planner is in charge of processing look ahead information and generating action plans, i.e. sequences for steer and speed control, and the RC maps short-term information to a single steer control value. The final steer 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 planner.

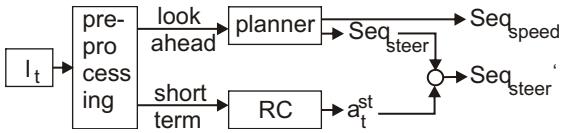

Fig. 2. The system setup. I_t denotes the image frame at time t and Seq a sequence of actions.

3.1 Planner

As mentioned before, the idea is that a system should be able to correlate experienced action sequences with visually perceived situations. When exposed to similar situations it should remember the previously conducted action manouvers. For example, if the system observes a right turn it should know from experience that it needs to steer to the right in the observable future and not to the left. This example also shows that even if this plan is not very exact, it already provides guidance into the right direction. We realize this idea by building a database, where the system stores triplets containing a perceived situation description along with the according 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. This leads to the following steps: a) database construction, b) database query at runtime, and c) calculating control sequences at retrieval time.

a) For the database construction a visual state or situation description, $\boldsymbol{s}$, is needed comprising look ahead. For that purpose we use a polygonized approximation of the right street lane, thus $\boldsymbol{s} = [s_0, s_1, ..s_l]$, where s_i, with $0 \leq i \leq l$, are the corner points of the polygon. For detecting the lane we develop a simple and fast algorithm based on conventional edge detection (Canny [16]) which returns an ordered 2d-curve, that is then polygonized using the Douglas-Peucker method [17]. Note, that the vertices of the vector $\boldsymbol{s}$ are also 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 of an extracted lane can be seen in Fig. 3 and 5a. It is a rough description of the observed street which contains look ahead information, but not explicitly extracted detailed information.

To each $\boldsymbol{s}_t$ according 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 $Seq_{speed} = [a^{v}_t, a^{v}_{t+j1}, .. a^{v}_{t+n}]$. 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 an experimentally determined value, which is:

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

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 either 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.

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, that 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 $\boldsymbol{\omega}_{i+1} < \boldsymbol{\omega}_i$, precisely 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

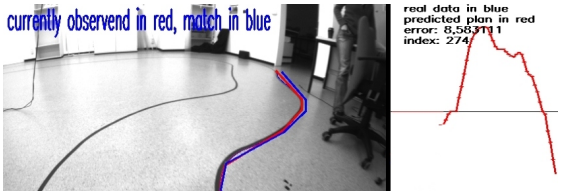

Fig. 3. Screenshot example of the planner operating mode. Left: The observed street (red) is compared to the database entries and the best match is returned (blue). Right: The assigned steering sequence of the best match.

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 exactly the same images are retrieved for a second time. In other words, how can we generalize using the database output? Postponing this step until retrieval time is typical for lazy-learning algorithms that often employ local learning [12, 18]. Here, we adopt this idea and test two different ways of using the query information: 1) We use values from *one* retrieved sequence until a better match is found or it ends, 2) we use values from *all* (or the latest N) retrieved sequences, (comp. Fig. 4A and B). 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, since it is based on simple averaging.

We begin with building an intuition for the DIFF method: The first step is to construct a sequence $\tilde{a}$ from the "raw" database response as shown in Fig. 4A. For that we query the database every timestep and keep a retrieved sequence, let's call it Seq_{best} as long as we do not find another better match, that is as long

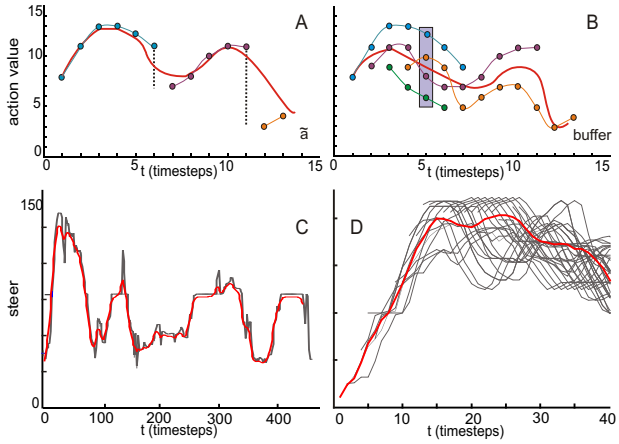

Fig. 4. Visualizing the data DIFF (A) and AVG (B) operate on. Points of same color denote actions from the same retrieved sequence. The red line indicates the final values obtained by each method. A) The basis for DIFF, the sequence $\tilde{a}$, constructed from single sequences, which leads to jump discontinuities between the single parts as indicated by the dashed vertical lines. B) The basis for AVG, a buffer, holding many sequences, the colored rectangle denotes the vector v, see text for more information. C) Shows real data from DIFF and D) from AVG. Here the gray lines indicate sequences.

as $\epsilon_{best} \leq \epsilon_t$, where ϵ_{best} denotes the difference measure ϵ affilliated with Seq_{best} and accordingly ϵ_t the difference affiliated to the current database query. To acknowledge that a good match that was found a few timesteps ago is less well suited at the current timestep we discount Seq_{best} by adding a decay factor λ to it at every timestep, with $\lambda = 5$. If j denotes the number of timesteps that we are using a given Seq_{best}, we set $\tilde{a}_t$ to Seq_{best_j}, which is just formally expressing that we use consecutive action values from a given best action sequence. Important to note is, that we can also use $future$ values, i.e. $\tilde{a}_{t+1} = Seq_{best_j+1}$, $\tilde{a}_{t+2} = Seq_{best_j+2}$, ..., $\tilde{a}_{t+n-j} = Seq_{best_n-j}$, with n being the length of Seq_{best}. Note that this way of keeping a match as long as a better one is found, can lead to jump discontinuities between those parts that stem from different sequences, (comp. 4A). The second step is applying the difference equation (4) on $\tilde{a}$. The equation realizes smooth transitions from one found action segment to another, but based on future information, i.e. not only $\tilde{a}_t$ but, $\tilde{a}_{t+i\tau}$, comp. equation (4), from which new action sequences can be generated that are not contained in the database. In Fig. 4C an example of DIFF on real data is shown. The formulas are:

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

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

where a_t is representing either a_t^{st} or a_t^v, and n is the length of the currently used sequence. The variable τ is a constant determining the sampling frequency on the current sequence. It influences how fast "jump" from one segment 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. The denominator decelerates the growth of the function if the previous action was already close to the maximum, thus counteracting the emergence of too extreme actions, it is determined by G, a constant, which we set to 10, and $\alpha_i = e^{\frac{-i^2}{\sigma^2}}$ a decay term which discounts the influence of future values of $\tilde{a}$, and σ is a constant, which we set to 4. Finally a_{max} is the maximum steer or speed value from all the stored sequences in the database.

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 as visualized in Fig. 4B. 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. Thus:

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

3.2 RC

One potential problem with the database approach is, that it will most probably only work well on streets similar to what is contained in the database. We tested this assumption by letting the robot drive on an unknown track[3] and indeed, in sharp turns it looses the street. The reason is that the system has never experienced such sharp turns and as a consequence it does not contain appropriate entries in the database. That means that the planner provides action sequences that belong to more shallow curves, thus the amplitude in steering is less then required for the current sharp turn. Simple averaging cannot transform low amplitude signals into a high amplitude signal, in other words the extrapolation capacity of the planner is poor. Since for a real car observable street trajectories are very repetitive this problem can be solved by ensuring that the database contains diverse enough entries which are collected during a training phase. To achieve a robust driving behavior during this training phase as a fall-back strategy we add a second controller. It takes only a short stretch of the observable street into account, we call it short-term situation, and maps a single action to it $s_t \mapsto a_t$. One can think of this as stimulus-response or reactive system. We decided to add such a system to the robot control and to combine it with the planner output.

The reactive controller, RC, is 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

[3] This is the same track the robot was trained on, but the opposite direction.

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 assigned human actions (from the training set) to the state space, comp. 5A and to fill the empty spaces, that is to generalize to unknown situations we use k-nearest neighbour, shown in 5B. 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.

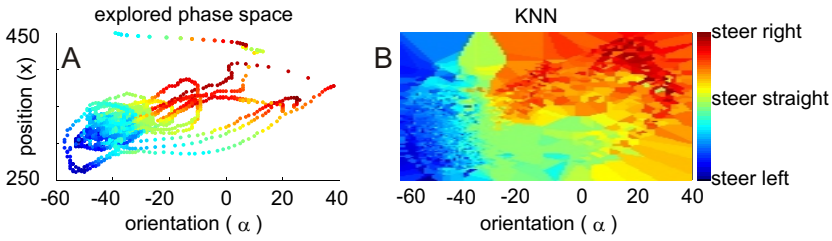

Fig. 5. A: The acquired policy from the supervisor. B: The interpolated policy using k-nearest neighbour.

3.3 Combination of Planner and RC

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 the robot's currrent state. Theoretically this should be indicated by ϵ, the error measure of the database query. If no sufficiently good match to the currently observed image is contained in the database the system performance should decrease and this must be correlated with the value of ϵ. To test if this is really the case we let the robot drive on the known and unknown track. To have a value that describes system performance other than ϵ we measure the lateral deviation of the robot from the right lane boundary. One could argue, that we could use this value straight away for measuring system performance, then we would not need to consider ϵ, however, a change of this value does not necessarily mean that the system is in a critical state, e.g. it also occurrs if the supervisor likes cutting curves. Nevertheless, in addition to ϵ it gives us at least some information about the system's state, so we can evaluate ϵ. The result is shown in Fig. 6F. We can observe that indeed a high error and a higher deviation from the street lane coincide. At the end of the plot, where ϵ is highest, the robot lost track of the street.

Having confirmed the assumption that ϵ indicates the system performance we can now combine RC and planner as a function of ϵ, $f(\epsilon) \rightarrow \omega_c$ with $\omega_c \in [0, 1]$. The smaller ϵ the more we want to rely on the planner, the larger the more we

consider RC output: $steer = \omega_c * RC + (1 - \omega_c) * planner$ with $f(\epsilon) = e^a$, and $a = \frac{(\epsilon - \epsilon_{tolerable})}{150}$, we set $\epsilon_{tolerable}$ to 700.

4 Results

To test the algorithms DIFF, AVG and RC we produced training and test data. For that we let a human drive the robot eight laps (always in the same direction) on our parcours in the lab. 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 prediction, i.e. generating a_t, which is shown in Fig. 6A-E. For velocity prediction with AVG we found best results when averaging over the last $N = 20$ buffer entries and for steer the last $N = 10$. It can be seen that all three methods capture the human behavior, where AVG and DIFF give smooth output and RC is comparably jerky. Conventional smoothing (low pass filtering) can only be applied to a certain degree, since it introduces a time delay.

The error for speed prediction on average is higher than for steer, which is explainable, since 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, where the depicted speed can be considered to be constant. We also compare the methods by plotting the squareroot 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. It can be seen that there is no statistical difference between AVG and DIFF. The higher error in RC can be explained by its jerkiness.

As this is a quantitative comparison it is necessarily offline and it does not prove if the system behavior would also be acceptable if the controllers were placed 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.

Concerning lateral control the combination of planner and RC is supposed to improve the system performance. In case of an unfamiliar environment that is not found in the database the RC should still be able to issue appropriate steering signals, however, without the ability to plan ahead. We trained the robot in one direction, and since in our setup the track is circular the robot is almost exclusively exposed to turns into the same direction, in our 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 that the latter captures the human signal better than RC or planner output alone. We show an excerpt of this drive in Fig. 8 were 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 to the robot. The amplitude of the signal is very important. Over- or understeering without correction would lead 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 steer 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 turns due to the explained reason. Using RC it can stay on track, which is as expected, however, the behavior is less smooth. Finally, when using the combination it manages to stay on the track and it drives smoothly on the known parts, which constitutes the majority of the encountered situations.

To evaluate the performance of the system concerning sequence prediction which 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, but does not lead to better results in generating single actions, as shown above. Again we test quantitatively and qualitativly. For the former 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 that was elicited by the human at that time. We sum the difference over the entire test set and plot it in Fig. 7B and C. We also included RC[4] in this plot, mainly for comparison. It can be seen that RC's predictive capacity is very low - as expected, and that AVG's predictive capacity is high, 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 the longer we predict, which is also as expected.

For qualitative testing we let the human drive and suddenly block the view. This can be interpreted as a short sensor "black-out", which might occurr due to technical problems. We measure the timesteps that the human can stay on the street without the visual feedback. For that we use a constant speed (during human performance, not for the robot), since otherwise the probands tend to stop the robot, which is probably not a bad strategy. Furthermore we decide to block the view shortly before a curve, which means, that a real change in actions is required. We repeate this with three more probands, where 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 is successfully using its generated plan and it does it similar to the trainer. It also shows that the less well trained humans loose the track quickly.

[4] 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 $t = 30$.

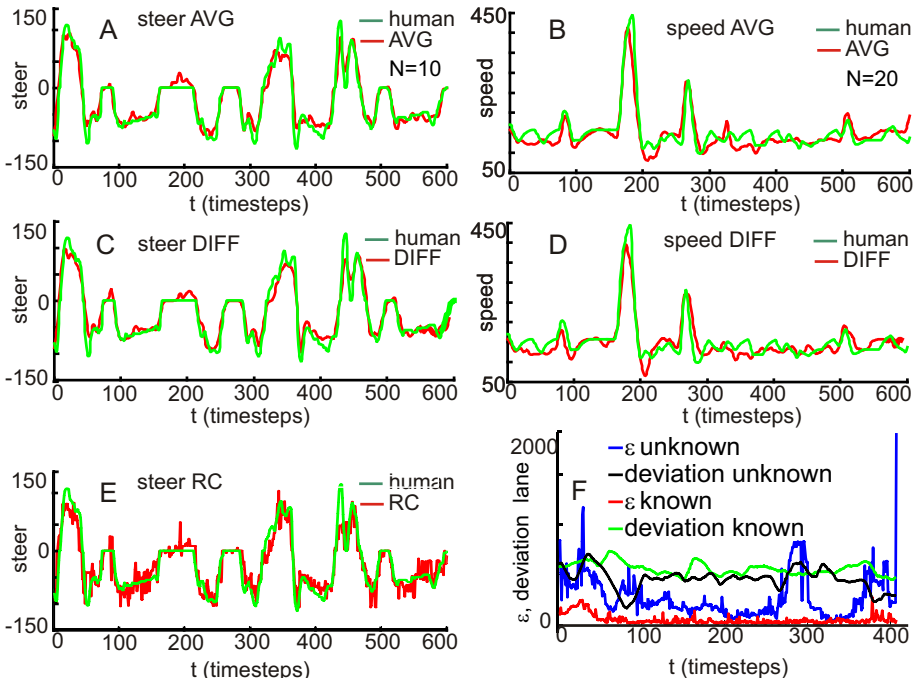

Fig. 6. A and B: Performance of AVG on generating a_t^{st} and a_t^v. "N" is the amount of entries in the buffer over which was averaged. C and D: Performance of DIFF on generating a_t^{st} and a_t^v. E: Performance of RC on generating a_t^{st} F: Correlation between the quality of database retrieval and system performance. Plots labeled as known/unknown denote data from driving on a known or unknown track. "deviation" is the measured lateral deviation of the robot from the right lane boundary, and ϵ is the measured database retrieval error. It can be observed that a high ϵ value results in a higher deviation of the system from the right lane.

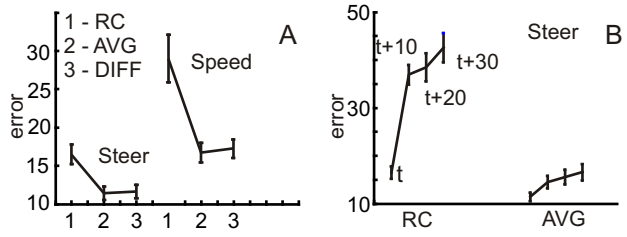

Fig. 7. A: Comparison of AVG, DIFF and RC for steer prediction for a_t. B: Comparison of RC and AVG for steer sequence prediction.

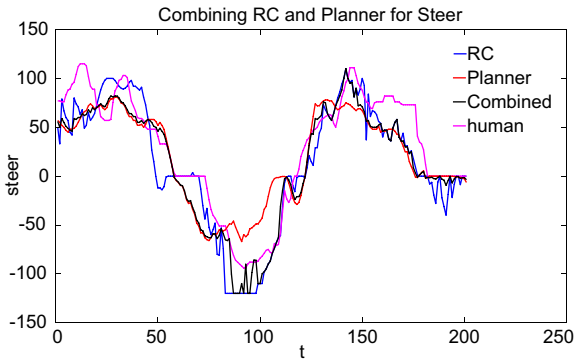

Fig. 8. Comparison of the combined signal to RC, Planner, and human output, where the robot was driven by the human. At around $t = 100$ it can be seen how the combined signal improves the Planner output by being drawn closer to the RC signal.

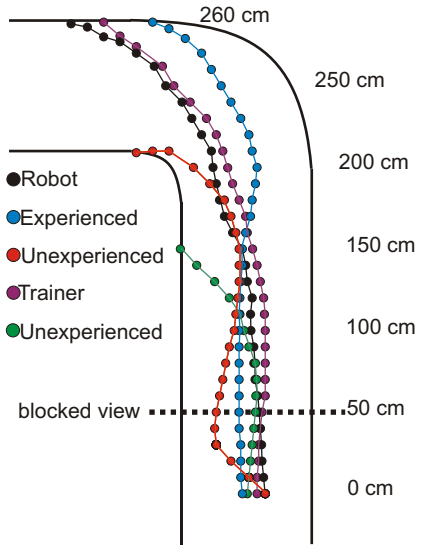

Fig. 9. Top view on part of the track. Shown is the driven trajectory of the probands and the robot, sampled every 10 centimeters. The vertical line denotes where the view was blocked.

5 Discussion

We presented a robot-car that learns predictive driving from a human supervisor and visual sensory data. Predictive means, that it can generate action sequences and that it can 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 control in a human-like way where it can also deal with missing visual input due to the sequence generation. We described the underlying controller which is based on the combination of a reactive signal and a generated plan, which leads to better performance than each single signal alone. The system is simple and apart from the image preprocessing, where we extract the right lane boundary, it makes few assumptions about the problem domain which makes it relatively general.

The image processing is an important issue. As mentioned above neural networks can be used for that. However, their processing is opaque, i.e. it is difficult to analyze what they are actually doing [19]. On the other hand particular influential features are often not known a priori and finding them is difficult. We consider our work to be a compromise between these two extremes. Extracting the lane requires some work beforehand but it allows the construction of a transparent controller. On the other hand it is only a rough description of the street ahead and we do not need to know, and do not have to extract, more specific features like the so-called tangent-point[5], which is often considered to be highly influential in curve negotiation [20], the distance to an upcoming curve etc. Therefore, for this approach no stereo information, rectified or undistorted images are necessary.

For function approximation a database has some advantages compared to a neural network. Besides being transparent, it stores the expert knowledge of the supervisor, which can be used as is or for interpolation between examples. A neural network used as global approximator would smooth over these support points, which in this case is undesired. Also the training set for the network must be chosen with caution, if overrepresenting one event the net might forget about others. If only storing examples in a table there is no need to consider this. On the other hand a network is very small compared to the size of the database which makes it more efficient for storage. But todays large storage capacities and fast access decrease the importance of this aspect. More generally speaking this is actually not a question of database or network but rather concerns the differences between global and local approximation, which is discussed in [18] and [21]. Hence other local learning techniques can replace the database if desired.

One might assume, that the RC is sufficient as a controller, also for action sequence generation, since we can apply the RC on the currently observed situation s_j to get a_j and based on this predict s_{j+1}. If we do this repeatedly we get an action sequence. The prediction step could be done either by learning the transition matrix based on the training data, or by calculating it using knowledge of the robot and applying image projections. We do not do this because,

[5] Where the driver's gaze touches the lane in a curve.

in the latter case, we a) loose the advantages of a model-free approach, and b) we need knowledge about the camera parameters which makes the setup less robust and also contributes to the computational costs. If we use probabilities we make predictions based on predictions of which the accuracy decays very fast. In fact we tried using a feedforward multilayer perceptron and the results were discouraging.

As said in the introductory part, our work is related to *sequential decision making*, according to [11]. For convenience we restate the definition here: "$s_i, .., s_{i+j}; G \mapsto a_j, .., a_{j+k}$ That is, given $s_i, .., s_{i+j}$ and a goal state G we want to choose action a_j through a_{j+k} that might lead to that goal." Apart from the fact, that we do not have an explicit goal, we similarly consider the current situation s_{i+j} to obtain an action sequence. However, this action sequence is not used directly, but for constructing either $\tilde{a}$ in case of DIFF, or the buffer used for constructing $\boldsymbol{v}$ from which we then compute the $a_j, ..a_{j+k}$, in case of AVG. Thus, we never compute actions based directly on past situations, but instead consider *predictions* from the past. Using predictions from the past, differs from using past situations, since in the latter case a system might become inert, not able to react fast enough to new situations. Predictions, however, even if they are old, are based on the observable future and are thus, more flexible. Of course, if the observed situation changes this would not hold anymore. But usually a right curve does not turn into a left curve from one moment to another.

This paper makes an original contribution concerning speed control, which has been much less investigated than steer. Simpler controls are Adaptive Cruise Control (ACC) systems that use radar or laser to slow down the vehicle when detecting an obstacle in front or Intelligent Speed Adapters and Limiters, ISAs and ISLs [22] that adjust or limit a vehicles speed according to the given mandatory limits. Other approaches determine speed given that there is a leading vehicle [23]. More related to this work are [7] and [24], where simulation has been used to train fuzzy neural networks using human training data to anticipate curves and regulate speed accordingly. Where these works have some nice properties, e.g. the linkage to a symbolic level due to the used fuzzy logic paradigm, both generate a single action per timestep. In the former the input to the fuzzy controller is based on experiments that investigated what processed information is acquired while driving. In our case knowledge of such specific parameters is not necessary.

References

1. Sun, R., Sessions, C.: Learning plans without a priori knowledge. Adaptive Behavior **8** (2000) 225–253
2. Dickmanns, E.: Dynamic Vision for Perception and Control of Motion. Springer (2007)
3. Gregor, R., Lutzeler, M., Pellkofer, M., Siedersberger, K., Dickmanns, E.: Emsvision: a perceptual system for autonomous vehicles. Intelligent Transportation Systems, IEEE Transactions on **3** (2002) 48–59

4. Pomerleau, D.: Alvinn: An autonomous land vehicle in a neural network. In: Advances in Neural Information Processing Systems 1, Morgan Kaufmann (1989)
5. 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. (2007) 645–650
6. Togelius, J., Lucas, S.: Evolving robust and specialized car racing skills. In: Proc. CEC 2006. Evolutionary Computation IEEE Congress on. (2006) 1187–1194
7. Partouche, D., Pasquier, M., Spalanzani, A.: Intelligent speed adaptation using a self-organizing neuro-fuzzy controller. In: Proc. IEEE Intelligent Vehicles Symposium. (2007) 846–851
8. Weng, J., Chen, S.: Autonomous navigation through case-based learning. In: Proc. International Symposium on Computer Vision. (1995) 359–364
9. Pomerleau, D.: Neural network based autonomous navigation. In: NAVLAB90. (1990) 558–614
10. Pomerleau, D.A.: Neural network vision for robot driving. In: The Handbook of Brain Theory and Neural Networks. M. Arbib (1999)
11. Sun, R., Giles, C.L.: Sequence learning - paradigms, algorithms, and applications. In Sun, R., Giles, C.L., eds.: Sequence Learning. Volume 1828 of Lecture Notes in Computer Science., Springer (2001)
12. Aha, D.W., ed.: Editorial. In: Lazy Learning. Volume 11 of Artificial Intelligence Review. Kluwer Academic Publishers (1997) 7–10
13. Florentin Wörgötter, B.P.: Temporal sequence learning, prediction, and control: A review of different models and their relation to biological mechanisms. Neural Computation **17** (2005) 245–319
14. Donges, E.: A two-level model of driver steering behaviour. Hum Factors **20** (1978) 691707
15. Volksbot: (http://www.volksbot.de)
16. Canny, J.F.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Machine Intell. **8** (1986) 679–698
17. 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** (1973) 112–122
18. Bottou, L., Vapnik, V.: Local learning algorithms. Neural Computation **4** (1992) 888–900
19. Pomerleau, D., Touretzky, D.S.: Analysis of feature detectors learned by a neural network autonomous driving system. In: International Conference on Intelligent Autonomous Systems. (1993) 572–581
20. Boer, E.: Tangent point oriented curve negotiation. In: Proc. IEEE Intelligent Vehicles Symposium. (1996) 7–12
21. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural Comput. **4** (1992) 1–58
22. Brookhuis, K., de Waard, D.: Limiting speed, towards an intelligent speed adapter (isa),. Transportation Research Part F: Traffic Psychology and Behaviour **2** (1999) 81–90
23. Tahirovic, A., Konjicija, S., Avdagic, Z., Meier, G., Wurmthaler, C.: Longitudinal vehicle guidance using neural networks. In: Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. (2005)
24. Kwasnicka, H., Dudala, M.: Neuro-fuzzy driver learning from real driving observations. In: Proceedings of the Artificial Intelligence in Control and Managamnent. (2002)