Sensorless but not Senseless: Prediction in Evolutionary Car Racing

Hugo Marques, Julian Togelius, Magdalena Kogutowska, Owen Holland and Simon M. Lucas
Department of Computer Science
University of Essex
Colchester, CO4 3SQ, UK
{hgmarq, jtogel, mkogut, owen, sml}@essex.ac.uk

***Abstract*— In this paper we try to develop predictors in order to drive a simulated car around a track without the most recent sensor data. In order to test the predictive abilities of our car we developed two experiments: one where the sensor data was interrupted for a certain time and another where the sensor data is constantly delayed by a certain amount. The predictors are based on neural networks, and we compare backpropagation and evolutionary computation as methods of training these. In the end we found that predictors with good driving performance do not sample the set of predictors which minimize the prediction error in the sensors.**

I. Introduction: Prediction and Imagination

Prediction has been acknowledged to play a possibly great role in controlling complex biological bodies in dynamical and often hostile environments [1] and [2]. Take the example of a monkey swinging from branch to branch in a tree: despite the complex interaction between all its body parts, the muscles actuating them and the speed at which this happens, the body moves smoothly. From an engineering point of view such systems cannot afford waiting for sensor feedback in order to decide which muscle activations should happen next. This suggests the presence of an additional system which removes the large computational overhead; the forward model mechanism used in control theory seems to fit the profile. In nature forward-models have been confirmed

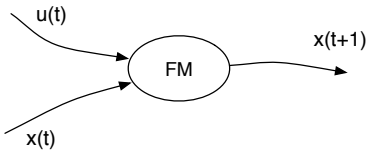

Fig. 1. Given a set of motor commands - *u(t)* - and the current state - *x(t)* - the forward model generates a prediction of the outcome - *x(t+1)*.

to exist in frogs and are generally accepted to be part of the motor control in mammals (there is still some discussion as to whether or not they exist insects [3] and [4]).

A. Prediction in Biologically Inspired Artificial Intelligence

If relatively simple animals such as frogs are capable of prediction, it is possible that the mechanism itself is simple. The study of these minimal and simple, yet paradoxically, complex behaviours goes back (at least) to the founding fathers of Ethology, e.g [5]. The concept of studying minimal behaviours stripped of superfluous details also have supporters in the field of cybernetics e.g. [6], advocates of the subsumption architectures [7], in the dynamical systems approach to studying minimally cognitive behaviours [8] and by practitioners of evolutionary methods e.g. [9]. The crux of the argument is that since it took billions of years to evolve these simple behaviours the focus should be on those rather than on the apparently more complex ones which have only been around for the last few tens of thousands years or so.

B. Background

The main source of inspiration for our experiments with prediction is Ziemke *et al.* who take a bottom-up approach for investigating the properties of prediction/anticipation and internal simulation (for details on experiments see [10] and [11]). The overall aim of those experiments was to try and find the elusive mechanism that allows an agent to make predictions and act according to these. The theoretical backdrop to the experiments involve the emulation theory (see Grush [12] and Hesslow [11]) which explain how an 'inner world' can exist in a simple agent. All the experiments mentioned, used a Khepera simulator and most of them some kind of variation of recurrent artificial neural networks (ANNs) trained with evolutionary algorithms (EAs). The results, related to prediction, were partially successful. Although the controllers evolved to navigate correctly in a given environment, the results were somewhat lacking when it came to navigating "blindfolded" i.e., when all sensors were switched off and the robots had to rely solely on their "inner world" representations of the environment to navigate.

II. Evolutionary Car Racing

In evolutionary car racing (ECR) evolutionary algorithms are used to create and tune controllers, sensors or other parameters for racing cars, in simulation or physical reality. Only a limited amount of work has been done on ECR, all quite recently; see [13] for a more comprehensive review. Previous work has focused on:

- Investigating various controller architectures and sensor representations [14]
- finding ways of developing ANNs with both general and specialised driving skills that can proficiently race a variety of tracks, as well as specialized controllers that perform very well on particular tracks.

- Used competitive co-evolution to develop car controllers capable of racing against, and avoiding (or sometimes initiating) collisions with competitors on the same track[15].

The obvious applications in computer games aside and the future of autonomous cars, our work is also in line with research in the field of evolutionary robotics. The way in which ECR differs from most other evolutionary robotics research is that racing a car around a track is a more complicated control problem than driving a differential-drive robot. It becomes harder still when more than one car and more than one track is considered. The addition of features raises the bar further by increasing the complexity of the environment.

We have previously argued that car racing is a promising environment for evolving complex general intelligence, because the task of navigating a basic track is relatively simple to learn, but can gradually be made more and more complex almost without limits; beating Michael Schumacher at his own game would require path planning, anticipation/prediction, opponent modeling, and complex knowledge about engine, tyre and surface characteristics, as well as fast and appropriate reactions.

A. Motivations

Being able to predict the future is beyond doubt a beneficial quality to possess. That goes for most agents, but especially for an agent whose job it is to get from A to B as quickly as possible. In a (potentially) changing world with (potentially) competing opponents, such predictive skills would be highly sought after. Having evolved general predictive capabilities would probably also mean that the agent's performance improved across a wide range of situations and environments. The trick, so to speak, is to evolve this general predictor. Because of these benefits, we find it both a worthwhile and important issue to focus our experiments on.

1) The Real World VS Simulation: The justification for using simulations of the real world rather than doing the experiments in the real world has been discussed widely e.g. [9] and [16]. Whilst in agreement with the proponents of embodiment and situatedness [7] it may seem strange to advocate the use of simulations. Fortunately, it has been shown that if the task is well-specified, the dynamics of the agent and environment thoroughly understood, then the transition from simulation to real-world is successful [17]. Contrary to behaviour-based robotics which require evolving and adding behaviours in real-time in the real world, the evolutionary robotics approach takes advantage of the simulator which allows testing of many different behaviours, morphologies and types of environments in a much shorter period of time and without the added technical difficulties that hardware poses. Although both approaches are based on the goals and knowledge of the researcher; the goals and knowledge can be implemented much more indirectly using the evolutionary robotics approach. So that the end result can be said to be less biased by the human designer if prior assumptions are minimised [9]. Further arguments in favour of using evolutionary techniques are outlined in [16].

2) Theoretical and Practical Motivations: The main motivations for this paper are two-fold: From the theoretical point these experiments allows us to apply the theory that biological organisms are capable of prediction using forward-models. The practical requirements of driving a physical car involves appropriate handling of faulty sensors as well as the delay in them (e.g. caused by a webcam-controller loop [14]). To achieve this we used evolutionary computation and back-propagation networks in order to devise predictors in different ways so that they could than be compared with each other. Our aim is to address the following questions:

1) Can a predictor be evolved, using a forward model and evolutionary algorithms?
2) How well can sensory data be predicted?
3) What would a good predictor look like? Would it be related to the average error in the sensor prediction?
4) How differently will the predictors based on back-propagation and evolutionary algorithms be? Will they present different characteristics? Which of them will work best?

III. Methods

The experiments reported in this article were done in a slightly updated version of the simulator used in [15].

A. Simulation Environment

The 2-dimensional simulator is intended to, qualitatively if not quantitatively, model a standard radio-controlled (R/C) toy car measuring 20*10 pixels (app. 17*8.5 cm) in an arena with dimensions 400*300 pixels (app. 3*2 m), and a track delimited by solid walls.

A track consists of a set of walls, a chain of waypoints, and a set of starting positions and directions. The car is added to a track in one of two starting positions, with corresponding starting direction. Both the position and direction of the car are subject to small random perturbations when the car is added. The waypoints are used for fitness calculations.

The dynamics of the car are based on a reasonably accurate mechanical model, taking into account the small size of the car and bad grip on the surface, but is not based on any actual measurements [18][19]. While the dynamics of the car itself are fairly straightforward, the collision handling has been subject to much tuning and exception-handling in order to get a behaviour that feels right for the human player and cannot easily be exploited in an unintended way by the evolutionary algorithm.

B. Sensors

The car experiences its environment through three types of sensors: the speed sensor, the waypoint sensor, and a number of wall sensors. The speed sensor is simply the speed of the car. The waypoint sensor gives the difference between the car's current orientation and the angle to the next waypoint (but not the distance to the waypoint). When pointing straight

to a waypoint, this sensor thus outputs 0, when the waypoint is to the left of the car it outputs a positive value, and vice versa. As for the wall sensors, which are meant to be abstract

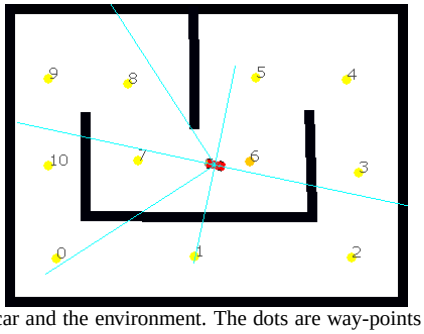

Fig. 2. The car and the environment. The dots are way-points and the lines protruding from the car indicate the sensors and their range

"range-finders" similar to sonars or IR sensors, each sensor has an angle (relative to the orientation of the car) and a range, between 0 and 200 pixels. The output of the wall sensor is zero if no wall is encountered along a line with the specified angle and range from the centre of the car, otherwise it is a fraction of one, depending on how close to the car the sensed wall is. A small amount of noise is applied to all sensor readings, as it is to starting positions and orientations.

C. Controller Architecture

The controllers and the predictors in the experiments below are based on ANNs. More precisely, we are using multilayer perceptrons with three neuronal layers (two adaptive layers) and *tanh* activation functions. A controller network has nine inputs: one fixed input with the value 1, one speed input in the approximate range [0..3], one input from the waypoint sensor, in the range [-Π..Π], and six inputs from wall sensors, in the range [0..1]. All networks have two outputs, which are interpreted as driving commands for the car. If the first output is above 0.3, this is interpreted as the forward command, less than -0.3 means backward, and anything else means neutral. The output of the second neuron means steer left if below -0.3, right if above 0.3, and straight forward otherwise.

In previous papers the sensor parameters were evolved, but in the current experiments each car is equipped with six sensors at fixed angles and ranges. This is done in order to ensure consistency in the representation of the state of the car; the state vector, as presented to the controller, predicted by predictors and stored in delay queues is simply an array of nine real numbers, as described above.

D. Network Training Procedure

In order to use supervised learning to train the network we gathered the state of the car and the control signal performed at each timestep during 70000 time steps. The order in which the samples were presented was always randomized before training the network. In order to test the results of the network we used a log file with 700 time steps gathered independently from the samples just mentioned. The network was always trained using a fixed amount of training sample presentations (7 million), which means that when trained with 7 examples the backpropagation algorithm ran for one million epochs and when trained with 70000 it ran for 100 epochs.

E. Evolutionary Algorithm

When evolving a controller or a predictor, the genome is a fixed-length array of floating point numbers, encoding the weights of the connections in the ANN.

A fairly standard EA is used: 50 genomes (the elite) are created at the start of evolution. At each generation, one copy is made of each genome in the elite, and all copies are mutated by adding a random number drawn from a gaussian distribution with mean 0 and standard deviation 0.1 to every value in the genome. After that, a fitness value is calculated for each genome, and the 50 best individuals of all 100 form the new elite. All evolutionary runs last 100 generations.

F. Fitness Function

There are two fitness functions, one for getting as quickly around the track as possible and the other for the prediction.

1) Performance Fitness Function: The fitness for performance is calculated as the number of waypoints the car has passed, divided by the number of waypoints in the track, plus an intermediate term representing how far it is on its way to the next waypoint, calculated from the relative distances between the car and the previous and next waypoint. A fitness of 1.0 thus means having completed one full track within the alloted time. Waypoints can only be passed in the correct order, and a waypoint is counted as passed when the centre of the car is within 30 pixels from the waypoint. In the experiments reported below, each car was allowed 700 time steps (enough to do two to three laps on most tracks in the test set) and fitness was averaged over three trials. On the track used in this paper, the best fitness achieved so far is slightly higher than 3.0, with the best evolved controllers narrowly outperforming the best human drivers.

2) Predictor Fitness Function: We also evolved predictors for prediction ability. The fitness evaluation was done by letting the original controller drive the car for 700 time steps (with unimpaired sensor data) and comparing the last prediction of the predictor with the real sensor state. Fitness was thus defined as the negative of the sum of errors.

IV. Experiments

In order to test our hypotheses, two extensions of the basic car racing task were devised. A number of experiments were performed on these two tasks, investigating the performance and behaviour of the same controller combined with different predictors. The predictors are multilayer perceptrons with 11 inputs and 9 outputs, which take as an input the sensor vector at time *t*, the action taken by the controller at time *t*, and returns a prediction of the real sensor vector of the car at time *t+1*.

A. Tasks

Imagine driving in an unlit tunnel, in a car whose headlights has the irritating habit of flickering on and off, sometimes staying off for several seconds at a time. Or imagine remotely controlling a vehicle based on an unreliable image feed that blacks out from time to time. These scenarios are the motivation for the intermittent task. The delay task, on the other hand, is motivated by the delay in car state information reaching the controlling computer from an overhead webcam in some of our real-world experiments.

1) Intermittent: Technically speaking, both tasks change the sensor model of the car. In the intermittent task, there are two possible states

- The normal state, where current sensor information is presented to the controller, or
- The blackout state, with all sensors off.

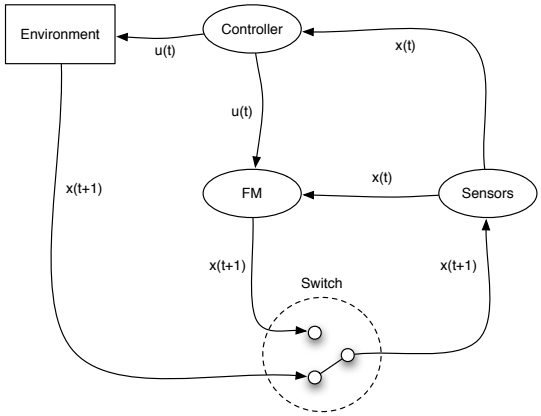

Fig. 3. The switch mechanism

In order to decide what signal reaches the controller we used the concept of switching proposed in [20](see figure 3). This consists of using the signal directly from the sensors whenever that signal is available and using the output of the forward model otherwise. At each timestep, the sensor model has a probability of 0.2 to go from the normal to the blackout state, and will then stay in the blackout state for a number of time steps drawn from a uniform distribution with a maximum of 10 or 20 time steps. While in the blackout state, the information received by the controller depends on whether there is a predictor present or not, and on the experimental setup. In the case without a predictor, the controller input is a vector of all zeroes, or the last real sensor vector before the start of the blackout. When a predictor is present, the controller input is the output of that predictor, which in turn receives input from the action and sensor vector at time *t*, be it the real information or the output of previous prediction, and tries to predict state *t+1*.

2) Delay: In the delay task, the controller never receives the current sensor data. Instead, when no predictor is present, the controller input is the current sensor vector from a number of *n* time steps ago, *n* equals 3 in most of the experiments. When a predictor is present, at each timestep the controller receives the end result of the predictor being run *n* times, starting with the real sensor vector and action at time *t-n*, and then its own predictions for time steps *t-n+1* to *t*.

B. Training And Evolving For Prediction And Fitness

Throughout the majority of these experiments, both the same controller and method for evolving it is used as described in [13]. When the sensor data is not altered in any way, this controller completes on average 2.84 laps on the chosen track, with a standard deviation of 0.007. This means that while other controllers drive somewhat faster, this is a very robust controller.

We chose to compare predictors created through three different methods: backpropagation, evolving for prediction ability, and evolving for driving ability. For the backpropagation training, a log of sensor data from a number of runs of the above controller with unaltered sensor data is used as training data. When evolving for prediction, the predictor is not affecting the control of the car, and the fitness value is the negative mean prediction error. Evolving for driving ability means measuring fitness as described in section III-E, with the predictor being evaluated indirectly.

For all of these methods, we systematically varied the size of the hidden layer in the predictor networks, with all methods being tested for networks of 5, 10, 15, 20 and 25 hidden neurons. We also varied some parameters for the other predictor creation methods: training with backpropagation was done with training sets of 7, 70, 700, 7000 and 70000 data points; evolving for performance on the intermittent task was done with maximum blackout lengths of 10 and 20 time steps; and evolving for performance on the delay task was done with delay lengths of 3 and 6 time steps.

C. The Intermittent Task

We trained 250 predictors using backpropagation modifying both the amount of training samples used (7, 70, 700, 7000, and 70000) and the amount of neurons in the hidden layer (5, 10, 15, 20, 25). We created 10 predictors for each data set size and number of neurons in the hidden layer. The results are presented in table I where we can see for example that for maximum blackout of 8, the fitness generally increases with the number of samples used in the training while the error decreases. In II we can see that the 50 controllers evolved for prediction had mean prediction error of 0.046 and fitness 0.077. In addition we can also see the mean results of the predictors evolved for maximum blackout time of 10 and 20 time steps.

Figure 4 contrasts the performance of three different predictors trained with backpropagation and the two non-predictor conditions (where the last sensor vector is retained during the blackout and where the sensors report zero during the blackout) for all maximum blackout lengths between 0 and 30 time steps. The three predictors consist of the predictor with the lowest prediction error of all the trained predictors, the predictor with the highest prediction error, and the predictor with the highest track fitness. It is clear that the controller without any predictor does not do very well

TD	AF	sd	min	max	ME	C
7	1.055	0.373	0.332	1.906	0.142	0.180
70	0.965	0.264	0.495	1.481	0.028	0.311
700	1.192	0.253	0.542	1.632	0.006	0.184
7000	1.398	0.194	0.842	1.930	0.005	0.188
70000	1.415	0.144	0.915	1.687	0.005	-0.072

TABLE I

STATISTICS FOR TRAINING WITH BACKPROPAGATION. THE HEADINGS ARE TD: TRAINING DATASET, AF: AVERAGE FITNESS, SD: STANDARD DERIVATION, MIN: MINIMUM FITNESS, MAX: MAXIMUM FITNESS, ME: MEAN ERROR, C: CORRELATION BETWEEN THE FITNESS AND THE ERROR

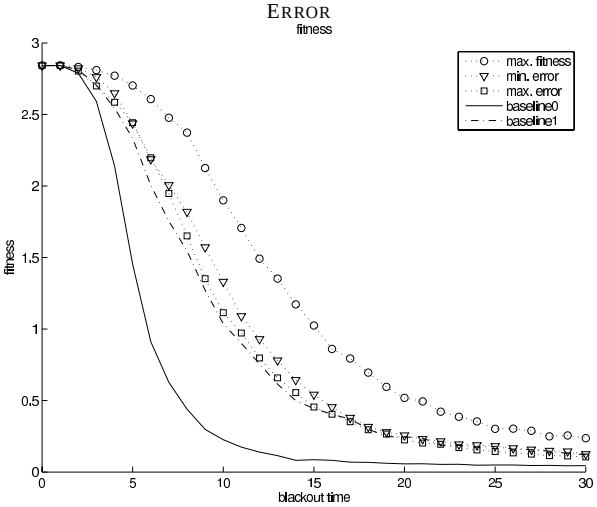

Fig. 4. Fitness results of predictors trained with backpropagation. Baseline0 - controller without prediction, baseline1 - controller runs with the last sensor input before blackout, min/max error - controller with lowest/highest prediction error, max fitness - predictor with highest fitness.

on the intermittent task for maximum blackout lengths above 8 or so. Most of the predictors trained or evolved for error minimisation make no improvement at all on this - some even decrease fitness. However, a few of those predictors manage to provide the controller with substitute sensor inputs that allow it to continue driving successfully in the absence of real data. The predictors evolved for performance are different. Almost all of them provide substitute sensor data that enable good driving. But useful as these sensor data are, they seem not to be predictions of what the real sensor data would have been at all, as the prediction errors for these predictors is very large. Figure 5 plots the performance of the best predictor evolved for performance with the maximum blackout length set 10 against the best predictor evolved with maximum blackout 20, the best predictor evolved for minimising the error, the predictor with the lowest error, and the two non-predictor conditions.

ET	AF	sd	min	max	ME	C
10	2.418	0.056	2.199	2.500	0.939	0.180
20	2.309	0.054	2.186	2.413	0.889	0.209
P	0.963	0.471	0.175	1.774	0.046	0.077

TABLE II

INTERMITTENT TASK: PERFORMANCE DURING BLACKOUT LENGTHS 10, 20 AND FOR PREDICTION. (ET: EVOLUTION TYPE AND P: PREDICTION. OTHER HEADINGS AS IN TABLE I)

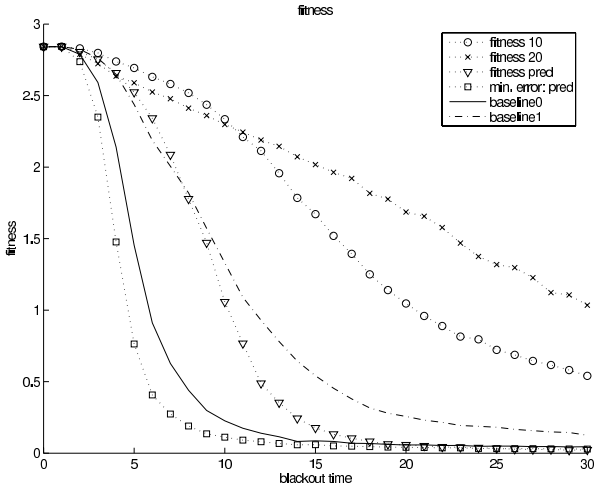

Fig. 5. Predictors evolved for performance

1) Behavioural Observations: Without a predictor, the car starts accelerating, turning left when sensor inputs are blocked. If the last sensation is kept at the start of a blackout the car, perhaps obviously, keeps doing exactly what it was doing before the blackout. The controller normally makes numerous small turns (a sort of wiggling motion) even on straight segments of the path. This often causes the car to end up driving into a wall even during straight sections of the track, when last sensations are retained. We also found that the wall bumping behaviour was popular amongst the majority of controllers trained for fitness; We were surprised to see how little influence the predictors had on the controllers - often doing nothing other than turning left and accelerating when asked to predict sensor data. When it comes to the evolved predictors, we are not entirely sure why they are so successful. A possible solution for the controller would be to apply brakes and thereby minimising the risk of collisions until the sensors are back online. But this *does not* seem to be the what happens. Instead, the controllers take different actions depending on the last non-blocked sensor vector, usually keeping the speed constant.

D. The Delay Task

The same 250 predictors used in the last experiment were used here. The average fitness of the predictors obtained for delay of 3 time steps is very low and there seems to exist a negative correlation between the fitness and the error (see III). In IV we can see the mean results of the evolution (delay=3time steps) for the 50 predictors evolved for prediction and for the 100 predictors evolved for track fitness (50 predictors evolved for delay of 3 time steps and the other 50 predictors for delays of 6 time steps). Here, we can see that the performance achieved by the predictor evolved for 3 time steps delay was good for a delay of 3 time steps but very poor for delays of any other length. In figure 7 we compare the performance of the best-performing, best-predicting and worst-predicting trained predictors running without prediction over a wide number of delay lengths. The corresponding plot for the evolved predictors is figure 8.

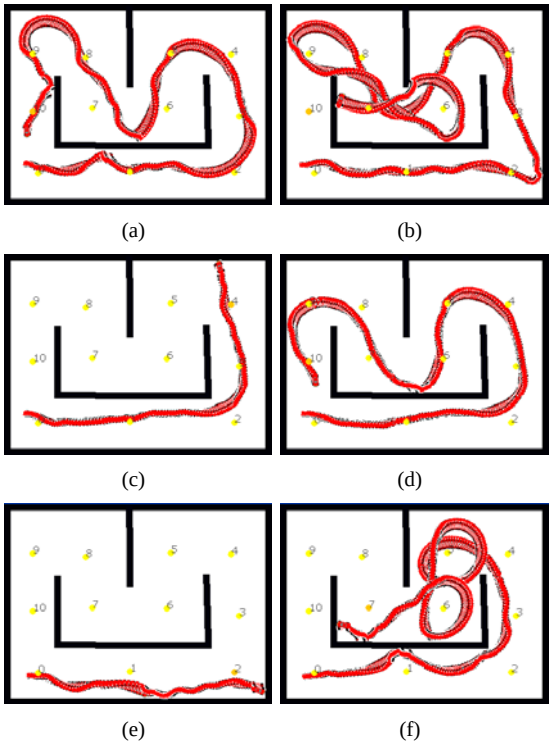

Fig. 6. Figure showing the trajectories of a range of predictors during the intermittent task with blackout length set to 10: a) The best performing predictor trained with backpropagation, b) The trace of the predictor with the lowest error when trained with backpropagation, c) The trace of the predictor with the highest error when trained with backpropagation, d) The best evolved predictor, e) The best evolved controller with the lowest error, f) Predictor with no prediction

DP	AF	sd	min	max	ME	C
7	0.063	0.099	-0.019	0.558	0.142	0.223
70	0.237	0.318	-0.045	1.800	0.028	-0.412
700	0.437	0.410	-0.003	1.869	0.006	-0.139
7000	0.482	0.454	-0.002	1.495	0.005	-0.319
70000	0.424	0.452	-0.005	1.449	0.005	-0.322

TABLE III

RESULTS AFTER TRAINING WITH BACKPROPAGATION DURING THE DELAY TASK. HEADINGS AS IN TABLE I

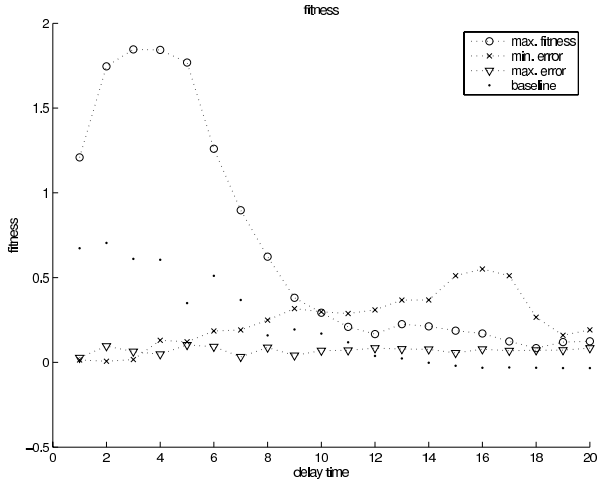

Fig. 7. Graph shows a comparison between the best-performer (max. fitness), best-predictor (min. error), worst-predictor (max. error) using no prediction and various delay lengths, trained with backpropagation. The baseline is a controller without predictor

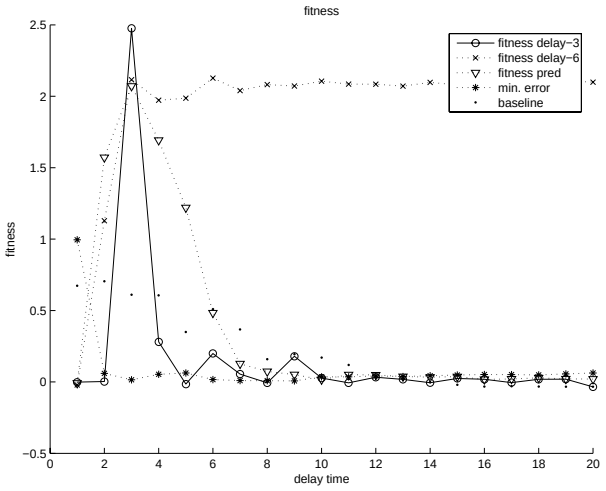

Fig. 8. Comparison between the *evolved* best-performer, best-predictor and worst-predictor using no prediction and various delay lengths. The baseline is a controller without predictor

ET	AF	sd	min	max	ME	C
delay3	2.279	0.203	1.250	2.478	0.563	-0.358
delay6	0.740	0.770	-0.070	2.271	0.592	0.060
P	0.413	0.624	-0.053	2.067	0.046	-0.200

TABLE IV

DELAY TASK: PERFORMANCE DURING THE DELAY TASK WITH DELAY LENGTHS 3 AND 6, AND FOR PREDICTION. (ET: EVOLUTION TYPE AND P: PREDICTION. OTHER HEADINGS AS IN TABLE I)

Overall, the pattern from the intermittent task was repeated: the predictors evolved for performance perform much better than those trained for prediction error on the delay task, though their prediction error was much higher. The good performance of the evolved controllers is a bit puzzling, as the controller never has access to real sensor data, and so it would seem paramount that the predictor provides as good a prediction as possible.

1) Behavioural Observations: All tested predictors and non-predictors have displayed variations on two fundamental behaviours. Most of the predictors trained for prediction simply make the controller turn to the left and crash. All other predictors (including no predictor at all) make the controller swerve from side to side, as if the driver was drunk. A plausible interpretation of this behaviour is that the controllers are constantly overcompensating for being too close to one wall or another. The fitness of a predictor seems to be inversely proportional to the magnitude of these oscillations, with the best predictors rarely if ever colliding with the wall.

E. Testing The Generality Of Evolved Predictors

Given that the evolved predictors seem to do what is asked of them in some way other than actually producing accurate predictions, it is reasonable to assume that they somehow exploit artifacts of the task, the track, simulator or the controller. We therefore decided to vary these factors.

First, we tested predictors evolved for one task on the other task. It turns out that predictors evolved for performance on

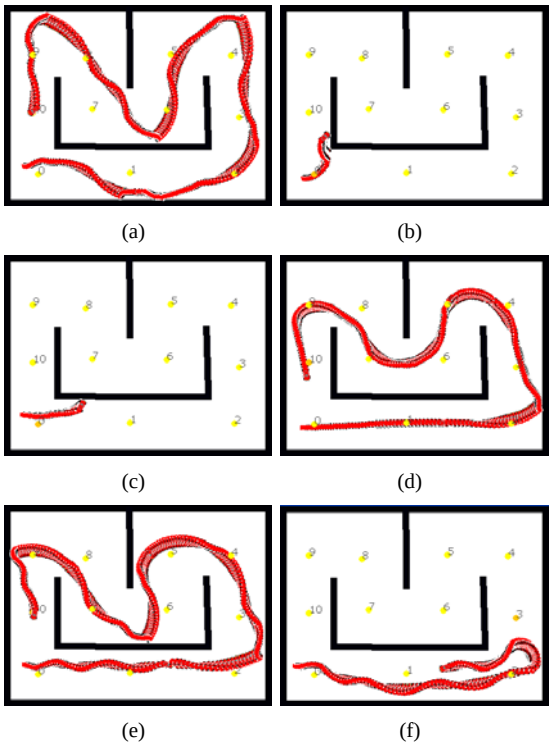

Fig. 9. Figure showing the trajectory of a range of predictors with n being the length of the delay: a) The best performing predictors trained with backpropagation for n=3, b) The trace of the predictors with the lowest error when trained with backpropagation and n=3, c) The trace of the predictors with the highest error when trained with backpropagation and n=3, d) The best evolved predictors with n=3, e) The best evolved controller with n=6 f) Predictor with not prediction and n=3

the intermittent task do extremely poorly when tested on the delay task. Typically, they back into the wall behind them, and so receive a slightly negative fitness. The performance of predictors evolved for the delay task on the intermittent task is a completely different matter. Here we saw a large variation between the predictors we tested, with some having a fitness of around 0.2, and others reaching as high as 1.9. It is worth noting that the best predictor on the delay task (with delay 3) was one of the worst on the intermittent task, while we found others that performed well on both tasks, with no distinctive pattern. We then tested the same predictor, on the task for which it was evolved, with several different tracks. We used the same six tracks of similar difficulty as was used in [13] for testing generalization and specialization among controllers. Three of the tracks run clockwise and three run counter-clockwise. The predictors on the delay task show some degree of generalization to other tracks, typically performing well on two or sometimes three out of the six tracks. Interestingly, almost all of them perform well on the mirrored, clockwise version of the original (counterclockwise) track. When testing generalization ability of simple controllers, they typically perform better on different tracks going on the same direction than on the same track in other direction. On the intermittent task, the predictors generalize much better. Many of them reach a fitness greater than 1 on all the six tracks in the testing set. Finally, we tested using the evolved predictors together with different evolved controllers. Three new robust controllers were evolved, using the same incremental method as used to develop the original controller, and their performance tested. On the basic task with uninterrupted sensors they generally had slightly lower fitness than the original controller, in the range of 2.3 to 2.5. Tested on the intermittent task without prediction, these three controllers did slightly better than the original controller, and on the delay task they did significantly better, in the range of 1.5 to 1.8 with a delay of 3. This is probably due to these controllers generally driving slower. However, when combining the predictors evolved to cooperate with the original controller with the new controllers, results look different. On the delay task performance is abysmal, all tested controller/predictor combinations had fitness between 0 and 0.7. On the intermittent task there is more variation, with performance that is usually in the region of that the tested predictor would have in combination with the original controller, but with some combinations of controllers and predictors failing completely. Again, there seems to be no obvious way to predict which controllers will work with which predictors. To sum up, the task-specificity of the evolved predictors is largely one way: predictors for the delay task often work well on the intermittent task, but predictors for the intermittent task fail completely on the delay task. Contrary to our expectations, the predictors seem not to be confined to working on one track only, but instead generalize reasonably well. On the other hand, evolved predictors seem completely confined to working only with the controller they were evolved with, at least on the delay task.

1) The "remarkable" predictor: The best evolved predictor is shown in the graph of figure 8 with a delay length of 6. This predictor is a bit of an anomaly in the grand scheme of things, however it is worth noting as it performs so well. The predictor continues to have a high fitness during cycles of 20 timestep delays, furthermore it is able to perform well (i.e successfully complete tracks) on 4 out of 6 tracks (see overview of possible tracks in [13]). Curiously though, it does not do very well on delay lengths less than 3. The behaviour is the usual wiggle and exploitation of the wall bumping tactic - hitting walls with the side and back of the car during turns and corner making. The likely reason for predictors to have a low fitness of less than 1.1 during prediction delays less than 3 to, is due to ineffective prediction abilities. Longer delay lengths allow for exploitation of the fitness function by ignoring almost anything to do with prediction and instead evolving a reactive behaviour. This type of behaviour can emerge when there is tight coupling between the agent and the environment, examples of this can be found in [21]. The car does not need any sensors in order to navigate successfully, wiggling along at high speed enables the car to drive into walls and more often than not avoid getting stuck. Another factor is the majority of the environments in which the predictor was tested had a similar topology. The predictor can exploit that

in most cases if it hits a wall it will more or less bounce of it without getting stuck, because the 'wiggle' ensures an angled impact trajectory. This strategy is more likely to fail in tracks with angled walls, narrower paths and fewer waypoints (see [13] for examples of such environments).

F. Evolving without predictiors

Finally, we investigated how well a controller without a predictor could perform, given that it was evolved specifically to perform one of the sensor-impaired tasks. We evolved several controllers from scratch for the intermittent task with maximum blackout length 10, and for the delay task with delay lengths 3, 6 and 20. For all cases we were able to find controllers that performed satisfactorily, except for the delay task with length 20. However, none of them were as good as the combination of the original controller with the best predictor. For the intermittent task the best fitness was 1.65; the strategy of this controller took the simple form of a very slow driving behaviour. For the delay task with delay 3 the best fitness was 2.1, for delay 6 the best fitness found was 1.9, and for delay 20 it was 0.4.

V. Conclusions

This paper describes mainly the learning of predictors used to drive a simulated car around a track in two different experimental setups: the intermittent task and the delay task. In relation to the questions presented in section II-A we can conclude that

1) It is possible to create predictors which are good enough to drive our simulated car along the track. However, these predictors are only predictors in the sense that they help the controller work without current sensor information - they do not actually predict very well in the sense of error minimization.
2) The sensory data can be predicted rather well in terms of the error but that does not result in a good performance
3) We found that predictors with the best fitness do not work by minimizing the prediction error. Predictors with small prediction errors did not generally perform well in any of the tasks
4) Predictors trained with back propagation were generally surpassed by predictors evolved for performance. It is plausible that they exploited the characteristics of the controller, the track, simulator or a combination.

VI. Future Work

In the future we aim to study whether different sensor configurations or the use of recurrent networks would improve our predictive capabilities. We also want to try to co-evolve accurate prediction together with performance to see if the results differ from the ones presented here. Other ideas include the use of abstractions rather than raw sensory data to drive the car. We have already some preliminary results on the use of abstractions where we managed to evolve a controller driving fairly well (fitness above 2.1) using clusters of the sensory data. Additionally, we would also like to try co-evolving the clusters with performance and check whether this changes our predictive abilities. We would also like to improve our simulator by embedding it in a real physics-simulator.

VII. Acknowledgements

We would like to thank Richard Newcombe and Renzo De Nardi from the Machine Consciousness lab for all their helpful comments and inspiring ideas. Hugo Marques is financially supported by the Portuguese FCT (Funcação da Ciência e Tecnologia)

References

[1] A. Clark and R. Grush, "Towards a cognitive robotics," *Adaptive Behavior*, vol. 7, no. 1, pp. 5–16, 1999.
[2] R. R. Llins, *I of the Vortex - From Neurons to Self*. The MIT Press, 2002.
[3] B. Webb, "Neural mechanisms for prediction: do insects have forward models," *Trends in Neurosciences*, vol. 27, pp. 278–282, 2004.
[4] D. P. Harland and R. R. Jackson, "Portia perceptions: the umwelt of an araneophagic jumping spider," in *Complex Worlds from Simpler Nervous Systems*, F. Prete, Ed., Cambridge, MA, USA, 2004, pp. 5–40.
[5] K. Lorenz, *The foundations of ethology*. Springer-Verlag, 1981.
[6] V. Braitenberg, *Vehicles, experiments in synthetic psychology*. MIT Press, 1984.
[7] R. Brooks, "Intelligence without representation," *Artificial Intelligence*, vol. 47, pp. 139–159, 1991.
[8] R. D. Beer, "The dynamics of adaptive behavior: A research program," *Robotics and Autonomous Systems*, vol. 20, pp. 257–289, 1997.
[9] S. Nolfi and D. Floreano, *Evolutionary robotics:The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA: MIT Press, 2000.
[10] T. Ziemke and M. Thieme, "Neuromodulation of Reactive Sensorimotor Mappings as a Short-Term Memory Mechanism in Delayed Response Tasks," *Adaptive Behavior*, vol. 10, no. 3-4, pp. 185–199, 2002.
[11] T. Ziemke, D.-A. Jirenhed, and G. Hesslow, "Internal simulation of perception: a minimal neuro-robotic model." *Neurocomputing*, vol. 68, pp. 85–104, 2005.
[12] R. Grush, "The emulation theory of representation: motor control, imagery, and perception (with commentary)," *Behavioral and Brain Sciences*, vol. 27, no. 3, 2004.
[13] J. Togelius and S. M. Lucas, "Evolving robust and specialized car racing skills," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2006.
[14] ——, "Evolving controllers for simulated car racing," in *Proceedings of the Congress on Evolutionary Computation*, 2005.
[15] ——, "Arms races and car races," in *Proceeding of Parallel Problem Solving from Nature*. Springer, 2006.
[16] I. Harvey, E. D. Paolo, R. Wood, M. Quinn, E. Tuci, and E. T. Iridia, "Evolutionary robotics: A new scientific tool for studying cognition," *Artificial Life*, vol. 11, no. 1, pp. 79–98, 2005.
[17] O. Miglino, H. H. Lund, and S. Nolfi, "Evolving mobile robots in simulated and real environments," *Artificial Life*, vol. 2, no. 4, pp. 417–434, 1995.
[18] D. M. Bourg, *Physics for Game Developers*. O'Reilly, 2002.
[19] M. Monster, "Car physics for games," http://home.planet.nl/ monstrous/tutcar.html, 2003.
[20] H. Marques and O. Holland, "Minimal architectures for embodied imagination," in *Proceedings of Brain Inspired Cognitive Systems 2006*, Lesvos, Greece, 2006.
[21] O. Holland and C. Melhuish, "Stimergy, self-organization, and sorting in collective robotics." *Artificial Life*, vol. 5, no. 2, pp. 173–202, 1999.