Nonlinear Dynamics Modelling for Controller Evolution

Julian Togelius
jtogel@essex.ac.uk

Renzo De Nardi
rdenar@essex.ac.uk

Hugo Marques
hgmarq@essex.ac.uk

Richard Newcombe
ranewc@essex.ac.uk

Simon M. Lucas
sml@essex.ac.uk

Owen Holland
owen@essex.ac.uk

University of Essex, Department of Computer Science
Colchester, CO4 3SQ
United Kingdom

ABSTRACT

The problem of how to acquire a model of a physical robot, which is fit for evolution of controllers that can subsequently be used to control that robot, is considered in the context of racing a radio-controlled toy car around a randomised track. Several modelling techniques are compared, and the specific properties of the acquired models that influence the quality of the evolved controller are discussed. As we aim to minimise the amount of domain knowledge used, we further investigate the relation between the assumptions about the modelled system made by particular modelling techniques and the suitability of the acquired models as bases for controller evolution. We find that none of the models acquired is good enough on its own, and that a key to evolving robust behaviour is to evaluate controllers simultaneously on multiple models during evolution. Examples of successfully evolved racing control for the physical car are analysed.

Categories and Subject Descriptors

I.2 [**ARTIFICIAL INTELLIGENCE**]: Automatic Programming

General Terms

Algorithms

Keywords

Evolutionary robotics, neural networks, forward models, system identification, games

1. INTRODUCTION

When evolving or otherwise learning a controller for a physical robot, having a model of the dynamics of that robot can be immensely useful for several reasons. Firstly, both evolution and other forms of controller learning (suchastd -learning) rely on trying out a great number of partially random actions. Performing such learning on a real robot usually takes a very long time and, depending on the type of robot, might very well destroy it. Therefore it is desirable to have a dynamics model of such quality that a controller learnt solely on actions taken in that model, produces behaviour which is essentially the same when that controller is transferred to the real robot.

Secondly, when such a model is made available to the controller itself, thus giving the controller the ability to predict the results of its own actions, more deliberative control is possible. It has been observed, by us and others, that learning state-based controllers is easier and ultimately gives better results than learning direct controllers [6]. Such results go hand in hand with current thinking in embodied cognitive science, which emphasises the role of internal models of action and perception for imagination and other aspects of cognition [4].

In this paper we are investigating how best to acquire a dynamics model suitable for evolving controllers, in the absence of much prior knowledge about the system being modelled. We are explicitly looking to get away with as little domain knowledge as possible, in order to create as general and self-contained an approach as possible. For example, we want to avoid measuring anything on the robot, and we want to avoid manually inferring even qualitative models of the dynamics, to the extent that this is possible. Also note that the problem of model acquisition in general is quite different from the problem of acquiring models suitable for controller evolution. Evolutionary computation is famous for exploiting any weaknesses in a model or fitness function, and it is far from clear that those models which are "best" according to some quantitative measure are also those that are most likely to allow evolution of controllers that transfer well to the real world. Understanding evolutionary computation is at least as important as understanding physics when it comes to creating non-exploitable models.

1.1 Fast, cheap and out of control

In this paper, the system being modelled is an inexpensive toy racing car (see figure 1). The main reason for choosing this system is that we have previously investigated the evolution of controllers for simulated car racing in some depth, including incremental evolution [10], competitive coevolution [9], and evolving forward models of the car dynamics [7]. The car simulation used in those experiments was directly inspired by the toy car being modelled in this paper, but is qualitatively rather than quantitatively simi-

GECCO'07, July 7–11, 2007, London, UK.

lar. While we have had good results in simulation, we have so far not known whether these results would hold up when confronted with the unruly dynamics of the real world.

But car racing is not the only domain that would greatly benefit from an autonomous methodology to produce dynamic models and controllers; our intention is to explore techniques general enough to model any robot. Such techniques would be most useful in cases of vehicles with complicated, partially unknown dynamics, and where the brittleness of the platform precludes evolving directly on the real robot.

A prime example of the above can be seen in a parallel project of ours, which aims to create swarms of miniature robotic helicopters [3]. Evolving controllers on the real helicopters is obviously out of the question, as the random actions taken by evolution would very quickly render the vehicle permanently incapable of taking off. Furthermore, the helicopter in question is characterised by a peculiar counter-rotating rotors design enhanced by a 45° stabilising bar, a combination that makes it quite different from the helicopters normally considered in the aeronautics literature. For this reason a model based on little domain knowledge is clearly an attractive prospect.

2. NONLINEAR DYNAMICS MODELLING

The modelling techniques presented in this paper belong to the wide and diverse field of nonlinear system identification; interested readers are referred to [8] for a more complete overview, including techniques more orthodox than those discussed here. It should be noted that there are many examples of both neural networks and evolutionary computation being used in nonlinear systems identification; the examples discussed below are to our knowledge more directly related to our current endeavour.

2.1 Jakobi and the Radical Envelope of Noise Hypothesis

One of the earlist approaches to acquiring robot models specifically for evolution is Jakobi's Radical Envelope of Noise Hypothesis [5]. Jakobi advocates dividing aspects of the robot and environment simulation into a *base set* and an *implementation set*. The base set contains all the aspects that are deemed (by the experimenter) to be required for a good evolved controller; these are subject to the same variability present in their real world characteristics. The implementation set contains all other aspects, and is subject to variable amounts of noise, from none to massive, in order to discourage the evolution of controllers relying on features from that set.Jakobi does not prescribe any particular way of delineating base set aspects from implementation aspects, nor any particular way of modelling the base set.

2.2 Bongard and co-evolution of models and controllers

Among the less traditional (and more recent) approaches to modelling is Bongard's concept of self-modelling [2]. He used a legged robot that was capable of continually producing models of its own body, by using these models to retrain the controller it was able to adapt to changes in it's body structure.

The process of continuous self-modelling is best described in a cycle. The robot starts by triggering an arbitrary motor action and records its sensory consequences. This information is passed to a set of internal models that compete for the best explanation of the behaviour. In order to improve the quality of the self-model the robot tries to gather further information by triggering different actions. The action selection mechanism is model-driven, which means that it selects the next action based on the maximization of information that can flow into the model (i.e. the action that causes the greatest disagreement among the competing internal models). The action selected is then overtly executed and the cycle starts again with the new sensory input information added to the system.

2.3 Abbeel and modelling for reinforcement learning

The work of Abbeel et al. [1] is probably the most similar to the one we are describing, as in our case the final goal is to produce controllers able to handle the challenging task of controlling a toy car. In this case reinforcement learning is the preferred technique used to produce proficient controllers.

A grey-box model heavily inspired by the dynamics of the car is the model of choice and all the available domain knowledge is embedded in the form of ad-hoc equations and parameters. Various measurements including proper measurements of the turning radii and techniques such as step-input responses are also used to estimate the initial parameters to seed the model. The car is also retrofitted with a power stabilizer in order to make its performance somewhat independent from the battery level.

The use of a car with proportional steering and drive, makes the problem of control substantially different from ours, direct comparisons are therefore very hard to make.

A second difference between the approach in question and ours regards the type of controller to be achieved. While Abbeel's aim is to produce a very precise controller, in our case optimality is less important; we are more interested in using evolutionary computation to produce somewhat novel solutions to the problem (see 5.2.2 for a more detailed account of the outcomes).

2.4 How our approach differs

While interesting, Bongard's approach seems to place certain requirements on the type of robot used, requirements which our system does not meet. Importantly, our model car is dynamical and all states are transient, whereasBongard's robot can be kinematically modelled; it would also be possible to crash or otherwise lose control of our car during the experimentation phase performed by the algorithm.

Abbeel et al. succeed in modelling a car and a conventional single rotor helicopter, but not without putting in considerable domain knowledge, and even adapting their systems so as to be easier to model (such as voltage stabilisation on the car). This is all good if modelling a particular car or helicopter is the primary goal. We aim instead to address the more general problem of nonlinear modelling for controller evolution when the assumptions about the underlying system being modelled are relaxed.

As for Jakobi's approach, the reliance on a human to separate base set from implementation set clearly violates our goal of minimizing domain knowledge.

2.5 Model requirements for controller evolution

As stated above, we are interested in system identification as a means of making it possible to evolve controllers, and the requirements we have on the models we infer are thus likely to differ from those placed on models used in e.g. traditional control theory. These differences stem from evolutionary computation's well known tendency to exploit the model or fitness function at hand. As trying randomly generated strategies is essential to evolution, the candidate controllers are likely to take actions which are very different from any actions in the training data for the model, or which otherwise "break" the model so that it produces responses which are wildly different from the system it is intended to model. Such weak spots in the model can often be exploited by the evolutionary algorithm to create controllers that achieve good fitness, but are completely useless in reality. An example of such an exploit for a car racing model would be if the model moved sideways when a steering command was issued while the car was standing still. A model with such a deficiency could well be learnt if the situation (steering while not driving) was not in the training data, and the right constraints were not applied to what sort of dynamics could be learnt.

That the models behave in a way which adheres to the constraints implied by the system being modelled is paramount, however beyond learning these large scale constraints, acquisition of non exploitable models is more important than achieving high precision with a specific training set.

However, there are ways in which the model could fail to conform to the modelled system and still be usable for controller evolution. In general, we believe that deficiencies that make it harder for the controller to perform its task (e.g. driving to a way point) is admissible, whereas deficiencies that make it easier to succeed are not. For concrete examples in our current domain, we believe that it is preferable that the model overestimates the turning radius of the car to that it underestimates it, that too long lag between command and effect is preferable to too short lag, and that it is probably acceptable if the model misjudges all accelerations by some constant, but absolutely not that the model makes it possible to turn the car on the spot.

That being said, we want to avoid encoding any of the above ideas into our model representations, in the form of explicit constraints on dynamics or otherwise, in line with our principle of minimising our use of domain knowledge.

3. DATA COLLECTION

The car we used is a small and inexpensive radio controlled toy car with a mass of approximately $0.3Kg$ and a length and width of respectively $18cm$ and $10cm$ (Figure 1). The car is provided only with simple bang-bang control inputs: forward or backward at full throttle and steering right or left. The combination of the asymmetry in turning and the slack of the worn out differential drive gearbox makes controlling the car a non-trivial task. The only modification made to the car was the addition of five very light and reflectivefiducial markers necessary for tracking. No electrical modifications of any kind were made, which means that the battery level had a direct impact on the car's top speed and responsiveness. The remote control of the car was

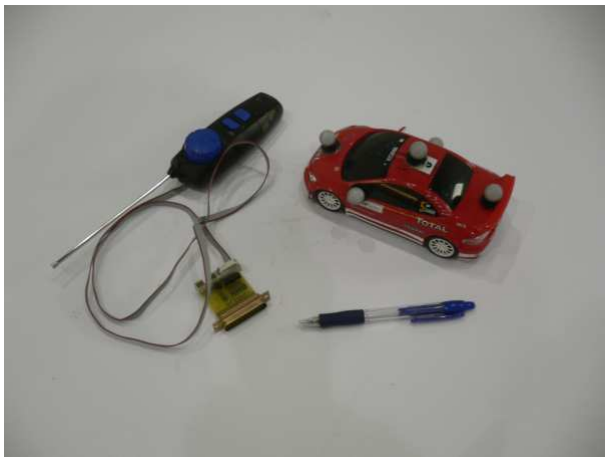


Figure 1: The toy car and its remote, note the reflecting markers.

modified in order to be controlled by the parallel port of our computer. A simple Java program outputs the steering commands given by a human driver via keyboard to the parallel port, and logs the commands together with the current car state.

The current car state was obtained from a Vicon infrared tracking system that can record the position of the markers placed on the car with a very high accuracy (in the order of millimeters) and a frame-rate of $200fps$. The real time data reconstruction delay offered by the system is also very low (in the order of milliseconds) and being considerably shorter than our control speed ($20Hz$) can be safely neglected. All the 50 square meters that constitute the test area were covered with white paper to reduce (but not eliminate) skidding of the car and infrared reflectivity of the floor.

The tracking system produces the absolute car position with reference to a coordinate frame fixed to the test area (a third person perspective), which is translated into the car centered frame of reference and numerically differentiated to produce the car state. As it can be expected the process of numerical differentiation introduces noise in the computed velocities. In order to limit the noise, we record the data with a time resolution of $200Hz$ and compute the velocities, which are then low-pass filtered and down-sampled to $20Hz$. To avoid distortion in the data a FIR (finite impulse response) filter was used and the delay introduced by the filtering process was compensated for. The control commands were logged together with the car state.

As it is common practice in vehicular dynamics we define the state of the car as the set constituted by its forward, lateral and angular velocities $[u, v, \dot{\psi}]$. The derivative of the state [1] $[\dot{u}, \dot{v}, \ddot{\psi}]$ will be used for the corresponding accelerations; u_1 and u_2 indicate the driving and steering input.

In order to minimise the use of domain knowledge, we decided not to collect data while a human driver was performing the point-to-point car racing task (see section 5.2), but

[1]Computed as $[\dot{u}, \dot{v}]_t = R(\dot{\psi}_t)^{-1} * [u, v]_{t+1} - [u, v]_t$, $\ddot{\psi}_t = \dot{\psi}_{t+1} - \dot{\psi}_t$. Where $R(\dot{\psi}_t)^{-1}$ denotes the rotation matrix between the body frame at time $t+1$ and t. Throughout the paper $[u, v, \dot{\psi}]$ and $[\dot{u}, \dot{v}, \ddot{\psi}]$ refers to the state and acceleration at time t.

instead while performing a set of various driving manoeuvres that we regarded to span the driving envelope of the car. In total, about 6 minutes of driving data was collected with different manoeuvres ranging from loops and figure-8s to simple starting and stopping.

4. MODELLING TECHNIQUES

Four different function representations, with different associated learning methods, were used to learn models of the car: a functionapproximator based on twenty-seven MLPs (multi-layer perceptrons) trained with back-propagation (*backprop27*), another based on three MLPs aided by an integration routine and trained with artificial evolution (*evolved3*), a simple nearest neighbour classifier, also using an integration routine, and a parametrized model of the car based on physical insight, with its parameters set by evolution.

In the nonlinear system identification literature, colour-coding is traditionally used to distinguish the level of prior knowledge that is available:

- *White Box models*: the model is perfectly known,
- *Grey Box models*: the model structure is known based on some physical insight, but its parameters remain to be estimated,
- *Black Box models*: prior knowledge is not used.

The *backprop27* is the darkest model we produced as it produces the next state of the system based on its current state and on the control inputs. The only handcrafted information pushed through the system was the choice of the neural network used depending on the active control signal (see below). The *evolved3*, the nearest neighbour and the parametrised models are grey-box models as they all assume a physical system. Given a state and a control action, they only produce the derivative of the state vector, a sound assumption for any physical system actuated by forces and torques. The *evolved3* and the nearest neighbour models are very dark grey boxes; they simply predict accelerations, leaving the computation of the new state to an external routine that takes care of the integration. In contrast, the parametrized model incorporates substantially more domain knowledge, as it explicitly tries to model coupling effects and friction that are known to be present in the real car.

4.1 Back-propagation MLPs

Our first model architecture does not assume anything about the system to be modelled, but rather about the space of inputs. Since the car has a relatively small set of action commands, and since those actions are discrete, it is possible to to take advantage of this by using a separate neural network for each possible command. This would possibly give each network a simpler function to learn, not only by reducing the output space, but also by eliminating the motor commands from the input space (as the appropriate network outputs can be selected solely on the base of the action input). In total we used 27 networks (9 possible actions $\times$ 3DoF) receiving as inputs the full state ($[u,v,\dot{\psi}]$) and the usual bias. The networks were then trained to predict the state of the system at the next time-step. We used the standard back-propagation algorithm based on the square error between the model output and the logged car data for the training. All the almost 7000 data point logged were repeatedly used in the 1000 training epochs.

4.2 Evolved MLPs

In this second attempt we raise the level of domain knowledge in the model by assuming that the state of the system is constituted by physical quantities that can be computed by integrating acceleration, direct results of states and input commands.

Three separated MLPs are in this case evolved to produce the difference between the current and the future state of the system. Each network uses as input the current state of the system and the control command and produces the change in one of the states. The weights of the neural network were then evolved using a standard 50+50 evolution strategy, encoding the weights of the networks as real numbers, and mutating themwith Gaussian mutation with variance 0.01.

The fitness function was the mean square error between the predicted state and the ground truth state logged for the car, calculated as follows. A starting point in the data was picked at random, and the state of the simulated car was initialised to be the same as the state of the real car. The same commands were then fed to the simulated car as to the real, and the state of the simulated car updated using the model under evaluation, for 10 time steps, after which the square difference between the real and the simulated state was calculated. The fitness of a model was defined as the negative mean of these errors over 100 repetitions of this process.

The *evolved3* models were generally evolved for 1000 generations, though they consistently achieved peak fitness after a few hundred generations.

4.3 Nearest neighbour

Like the *evolved3* models, but unlike the *backprop27* models, the nearest neighbour-based model maps current state and command to accelerations. The model is very simple and consists in having the real car data organized in nine tables, one for each control command, each entry mapping a recorded state $[u, v, \dot{\psi}]$ to a current acceleration $[\dot{u}, \dot{v}, \dot{\psi}]$. When predicting, given the current control command and state, the algorithm finds the state entry in the associated table with the smallest Euclidean distance and adds the corresponding acceleration entry to the current state.

4.4 Parametrised model

The parametrised model we adopted is inspired by the model presented in [1], and is mainly fruit of the insight on the underlying physics we gained by driving the car and studying the logged data. It consists in a set of algebraic equations, the parameters of which will be fitted to best approximate the car data. Since the lateral translations are generally very small, we deliberately neglected the lateral motion. There were three principal effects we wanted the model to be able to reproduce: 1) there can be asymmetry between forward and backward motion and right and left turning; 2) it is not possible to turn the car on the spot; 3) the motion of the car is characterised by static and dynamic friction. To achieve the first requirement we simply allowed for four different proportional constants between the control input and the respective accelerations (forward acceleration $\dot{u}_{\mathrm{f}}$, backward acceleration $\dot{u}_{\mathrm{b}}$, right angular acceleration $\ddot{\Psi}_{\mathrm{r}}$, left angular acceleration $\ddot{\Psi}_{\mathrm{r}}$). As suggested in

the model by Abbeel, to stop the model from spinning on the spot we simply defined the rotational speed as a proportion of the forward speed (see equation 1). And finally to account for static and dynamic friction we defined minimum velocity factors (minimum forward velocity $\mathrm{u_m}$, minimum angular velocity $\dot{\Psi}_\mathrm{m}$) and linear and viscous drag (linear drag $\mathrm{D_{ul}}$, linear viscous drag $\mathrm{D_{uv}}$, angular drag $\mathrm{D_{\Psi l}}$, angular viscous drag $\mathrm{D_{\Psi v}}$). Additional safety factors were also defined to avoid unrealistic velocities (maximum linear velocity $\mathrm{u_M}$, maximim angular velocity $\dot{\Psi}_\mathrm{M}$).

$$
\begin{aligned}
\ddot{\psi}_d &= \dot{\psi}\mathrm{D}_{\psi l} + sgn(\psi)\dot{\psi}^2\mathrm{D}_{\psi v} \\
\ddot{\psi} &= \mathbf{1}(|\ddot{\psi}| > \ddot{\Psi}_\mathrm{M})(\mathbf{1}(u_2 = 1)u\ddot{\Psi}_\mathrm{r} - \mathbf{1}(u_2 = -1)\ddot{\psi}\ddot{\Psi}_\mathrm{l}) - \ddot{\psi}_d \\
\dot{\psi} &= \mathbf{1}(|\dot{\psi}| < \dot{\Psi}_\mathrm{m})u + \mathbf{1}(|\dot{\psi}| \geq \dot{\Psi}_\mathrm{m})sgn(\dot{\psi})\dot{\Psi}_\mathrm{M} \qquad (1) \\
\dot{u}_d &= u\mathrm{D}_{ul} + sgn(u)u^2\mathrm{D}_{uv} \\
\dot{u} &= \mathbf{1}(|\dot{u}| > \dot{\mathrm{u}}_\mathrm{M})(\mathbf{1}(u_1 = 1)\dot{\mathrm{u}}_\mathrm{f} - \mathbf{1}(u_1 = -1)\dot{\mathrm{u}}_\mathrm{b}) - \dot{u}_d \\
u &= \mathbf{1}(|u| < \mathrm{u_m})u + \mathbf{1}(|u| \geq \mathrm{u_m})sgn(u)\mathrm{u_M}
\end{aligned}
$$

The 12 parameters that fully define the model were then evolved using the same evolution strategy and fitness function as described in section 4.2. In this case the parameters were encoded as arrays of real numbers and evolved with the same 50 + 50 elitist scheme adopted for the 3MLPs model. As with the MLPs, the parametrised models were evolved for 1000 generations but peaked after a few hundred.

4.5 Single- and multi-model controller evolution

In order to compensate for this we introduced multi-model evolution: we evolved two extra controllers using more than one model, at each controller evaluation the controller was tested using two or three of the best evolved models, acquired using different techniques and representations; the fitness used was the lowest fitness of those achieved. In this way, we reasoned, any evolved strategy that relied on exploiting a weakness of a particular model would score badly, as this particular weakness would not be present in the other models (but rather other weaknesses). According to this hypothesis, the extent to which we can avoid any systematic weaknesses that plague *all* our models depends both on the quality of the training data and the diversity of function representations and learning algorithms used to acquire the different models. A certain kinship with Jakobi's radical envelope of noise hypothesis can be seen in that the use of multiple models can be said to implicitly separate a base set (properties that can be modelled without exploitable weaknesses) from an implementation set (those that can not).

The performance of each controller was then tested with each one of the models and finally with the best model of all: the physical car.

5. EXPERIMENTAL RESULTS

5.1 Model acquisition

Using the various architectures discussed, several models were obtained and selected for controller evolution. In the case of the *n. neighbour* and *backprop27* only one model was produced for each technique; with both *parametrised* and *evolved3* the best models produced after the first evolutionary run had complete were chosen. The accuracy of the selected models were then verified using a validation dataset held back during training. In testing the models are initially given the state from the values of the real car. Each model is given the control signals (u_1 and u_2) recorded from the real car and at each time step the predicted state is propagated as the next state. In this way, we are able to see the deviation of each model from the baseline given by the testing data. In Table 1 are shown the root mean squared error (RMSE) of the predictions of the velocities (u, v and $\dot{\psi}$) when compared with the testing data.

models	u	v	$\dot{\psi}$
backprop27	0.6528 (0.6526) [58.90]	0.0832 (0.0832) [106.58]	1.3036 (1.3033) [80.35]
parametrised	0.3213 (0.3211) [28.99]	0.0669 (0.0668) [85.63]	0.5294 (0.5291) [32.63]
evolved3	0.6668 (0.6666) [60.17]	0.1498 (0.1497) [191.80]	1.0061 (1.0055) [62.01]
n. neighbour	0.3164 (0.3157) [28.55]	0.0223 (0.0223) [28.58]	0.4122 (0.4114) [25.41]

Table 1: The root mean squared error $[m/s]$, (standard deviation $[m/s]$) and [root relative error %] of each model in the testing data.

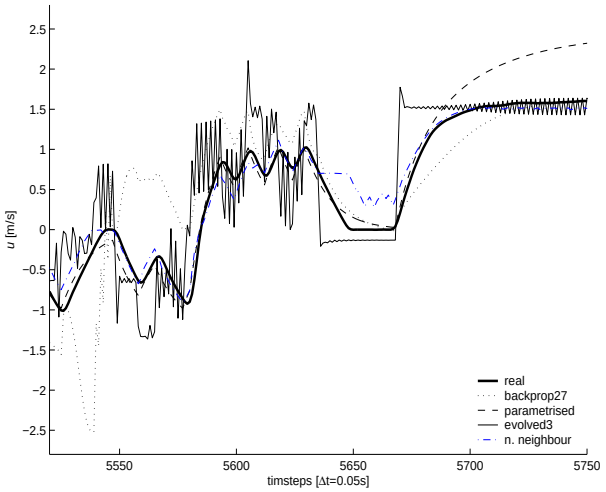


Figure 2: Predicted forward speed u for each of the models acquired

A plot of the predictions made by each model is shown in Figs 2 and 3 using a subset of the testing data. Here we show only the results of the variables u and $\dot{\psi}$, since these are the ones that have greater impact on the quality of the model.

The nearest neighbour model sometimes follows the baseline accurately and other times misses it completely. Without any form of interpolation, the nearest neighbour algorithm understandably fails to generalise in novel situations which are outside the envelope of the training data. In addition, the parametrised model often produces overestimates and sometimes is slow to react to the control commands.

The set of 27 networks trained with back-propagation can follow the baseline relatively closely, however, as seen in the example plots large errors in the predictions can occur over short periods of time (spikes). A quite different behaviour is seen in the predictions produced by the *evolved*3 model which has high frequency oscillations of varying amplitude.

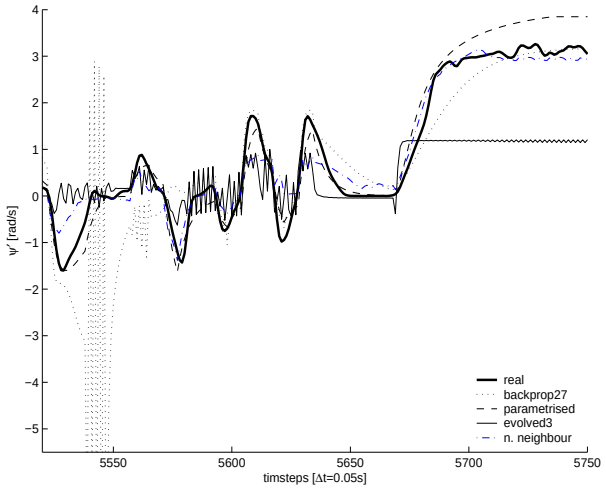


Figure 3: Predicted angular speed $\dot{\psi}$ for each of the models acquired

5.2 Controller learning

Once a set of models had been derived we proceeded to investigate their usefulness as simulators with which to evolve controllers. The task we chose to evolve controllers for was point-to-point car racing, a task for which we have previously compared various approaches to learning controllers in a non-physically-grounded simulation [6]. The reasons for choosing this task over the more complex walled car racing task we have studied in some other papers is that it does not require modelling of collisions with walls.

The fitness function of the task is defined as follows: the controller is allowed to control the car for 500 time steps, equivalent to 25 seconds of simulated time. During this time, the car has to pass as many way points as possible, and the fitness is equal to the number of passed way points at the end of the 500 time steps. A way point is considered passed when the center of the car is within 30 centimeters of the centre of the way point; when a way point is passed, a new way point immediately pops up at a random position within a radius of 1.5 meters from the centre of the arena. Only one next way point is available to pass at any one time.

The controllers we evolve are based on recurrent neural networks with 5 inputs, 6 hidden neurons, and 2 outputs. The inputs are as follows: a constant bias input of 1, the forward speed of the car (u), the rotational velocity of the car ($\dot{\psi}$), the angle to the next way point (relative angle between the forward direction of the car and the line that connects the center of the car and the waypoint), and the Euclidean distance to the next way point. All inputs are in SI units. The two outputs are interpreted as follows: the car is sent the command to steer left if the first output is below -0.3, to steer right if above 0.3 and straight forward otherwise. Similarly, the value of the second output means drive backward if below -0.3, forward if above 0.3 and neutral otherwise.

contr.	bp27	param	evo3	nn	real
bp27c	18.6 (0.76)	9 (1.19)	1.3 (0.61)	14 (1.04)	9.33 -
paramc	18.1 (0.79)	15.1 (0.66)	1.3 (0.43)	15.5 (0.62)	14 -
evo3c	0.7 (0.52)	0.2 (0.19)	13.4 (0.87)	0.3 (0.29)	0.33 -
nnc	15 (1.22)	5 (1.45)	0.7 (0.43)	17.0 (0.83)	5.33 -
mm1c	11 (1.57)	4 (1.3)	15.0 (0.95)	12 (1.40)	8.33 -
mm2c	13.6 (0.72)	10 (1.04)	10.2 (0.75)	11.7 (0.89)	9.33 -
humFc	- -	- -	- -	- -	8.33 -
humBc	- -	- -	- -	- -	6.0 -

Table 2: Fitness and standard deviation (below) of each controller on each model: the columns refer to the models and the rows to the controllers.

As for the evolutionary algorithm, the very same evolution strategy is used as is used for the evolutionary model acquisition above. All weights of the neural network are mutated in parallel by adding numbers drawn from a Gaussian distribution. Each fitness evaluation is the mean of ten trials of 500 time steps each.

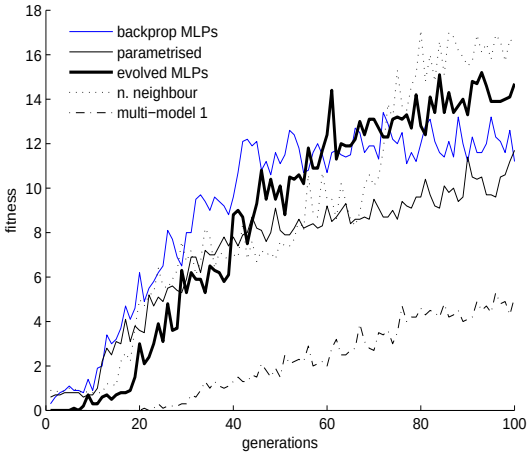


Figure 4: Evolution of controllers.

5.2.1 *Results*

Controllers that successfully completed the point-to-point racing task within a given model could reliably be evolved with that model within a few hundred generations (see figure 4). However, this is more or less a tautology. Things start to get interesting when controllers evolved for one model are tested on another, and really interesting when they are tested on the real car. In table 5.2 we can see the results of a number of such tests. The table contains four controllers derived using the modelling techniques described: nearest neighbour, backpropagation, neuroevo-

lution, evolutionary parameter optimisation (*nnc*, *bp27c*, *evo3c*, *paramc*), two controllers derived using versions of the multi-model controller evolution (*mm1c,mm2c*), and the best efforts of one of the authors driving either forward or backward (*humFc,humBc*). The controllers tested were the best of two evolutionary runs on the same model; no attempt at measuring the fitness variance between evolutionary runs was made, but we believe it to be low.

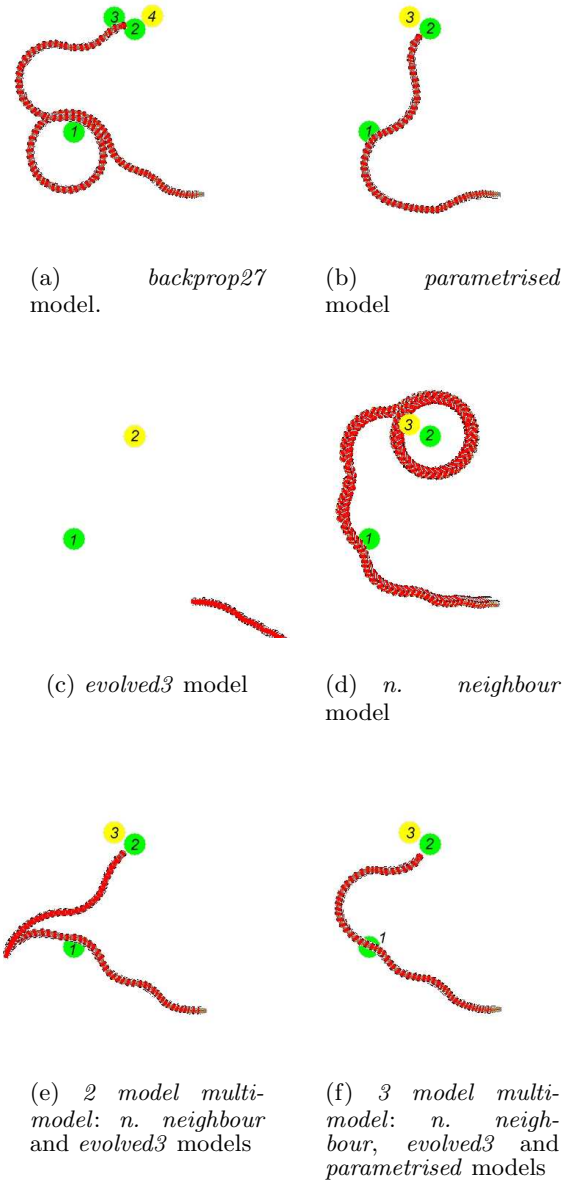
As is clear from the table, a controller evolved using a particular model works better on that model than on any other. This effect is most pronounced for the controller produced on the evolved neural network model, but is present for the other controllers as well. Qualitatively, the evolved neural network model behaves quite differently to the other models, with abrupt accelerations and huge turning radii. The *backprop27* model is generally the one that "feels most natural", while the nearest neighbour model behaves just like the real car in many situations only to behave rather inappropriately in situations for which it has no data.

The only controller that consistently performs well on all models is one of the multi-model-evolved models, which was tested simultaneously on the *evolved3* model and the nearest neighbour model. This controller also produced the most robust behaviour on the real car, even if the highest mean fitness on the real car was achieved by the controller evolved on the *backprop27* model. However, as we shall see below, these controllers go about their task in rather different ways. This is illustrated by figure 5, that traces a few seconds of several controllers trying to reach the same sequence of way points with the real car.

5.2.2 Analysis of evolved control strategies

When transferring the evolved controllers to the real car it was observed that of them drive the car backwards; all except the one which was evolved based on the 3-model multi-model trick (*mm2c*). The advantage of driving backwards seems to be that the car is more maneuverable; the car does go faster when driving forward, but this is apparently not very important given the limited size of the arena and the time taken to accelerate and decelerate.

Apart from mostly driving backwards, the behaviour of the controllers vary wildly. The controller evolved on the parametrised model (*paramc*) is perhaps the most straightforward (or straight-backward) as it drives directly towards the way point at full speed at all times. This works very well when the angle between the car and the way point is small or the distance to the way point is high, put if the next way point pops up right next to the car, the controller gets stuck "orbiting" around the way point, driving in endless circles without being able to reach it as its turning radius is too large. A variation on this behaviour is exhibited by the on average best-performing controller (*bp27c*) evolved on the *backprop27* model. This controller almost always turns right; approaching a way point it aims slightly left of the point and then does an inexplicably sharp right turn just before passing the way point. (We would have suspected an evolutionary exploit of a weakness in the model if the model was not actual physical reality.) The strategy works fabulously for most way points, but even this controller sometimes gets stuck in orbit, and some trials get very bad fitness. The reason for its high average fitness is that the trials are short, and switching between two trials usually gets the car out of orbiting.



(a) *backprop27* model.

(b) *parametrised* model

(c) *evolved3* model

(d) *n. neighbour* model

(e) *2 model multi-model*: *n. neighbour* and *evolved3* models

(f) *3 model multi-model*: *n. neighbour*, *evolved3* and *parametrised* models

Figure 5: Traces with each controller's performance using single models (a,b,c,d) or combinations of those models (e,f).

The only controller that seems never to get stuck is the one evolved on the 2-model multi-model controller (*mm1c*) evolution. While this controller quite often misses a way point, it usually immediately brakes to a full stop just afterwards. When accelerating again, the car has a narrower turning radius than it would have at full speed, and is thus able to turn back and reach the way point. This strategy surprised us at first, as we had not thought of it ourselves, but it obviously works.

Another strategy, as displayed by the 3-model multi-model evolved controller, is to drive forward most of the time, trying to aim for the way point. When missing a way

point the car backs off some distance and fully repositions so as to be able to reach the way point on a second try. The controller evolved on the 3-MLP model (*evo3c*) seems to perform some sort of peculiar dance around the arena, with little relation to the position of the way point or to the behaviour exhibited by the same controller in the model it was evolved for.

5.2.3 Analysis of strengths and (exploitable) weaknesses of the models

As the controller representation and evolutionary method are kept constant for all attempts at controller evolution, the characteristics of the models are bound to be the sole determinants of the differingfitnesses and behaviours of the evolved controllers. In this section, we attempt to analyse the strengths and weaknesses of these models based on observations when driving them manually, and observations on the evolved controllers driving in their native models.

The 3-MLP model is the most obviously exploitable model, as its very high rates of acceleration from standstill makes a kind of zig-zagging strategy possible, which certainly would not work on the real car. As the top speed of the model is not very high, but the turning radius is, it is all too tempting to use this exploit to quickly get to a way point. However, we note that the 3-MLP model works well in combination with another model for multi-model controller evolution, as its high turning radius precludes a turning radius exploit and the zig-zagging won't work in any other model.

The parametrised model, which is the model incorporating the most domain knowledge, is generally admirably well-behaved. But it is still vulnerable to turning radius exploitation, as its turning becomes unrealistically sharp at very low speeds. We have seen controllers evolved for this model driving fast and straight, and suddenly slowing down and creeping around the turns. A similar exploit seems to exist for the nearest-neighbour model. A more serious exploit for that model, however, is the spinning on the spot phenomenon, whereby the car can turn around without moving forward after certain decelerations.

For the *backprop27* model, things look different: when driving fast, and suddenly braking and changing the steering at the same time, the car can suddenly accelerate in unexpected directions. This little oddity seems not to be exploitable by the controller, but we have seen its slight underestimation of the car's turning radius being exploited.

6. DISCUSSION

The approach to dynamics modelling and controller evolution presented in this paper apparently works well enough to produce proficient (and interesting) controllers for toy car racing, using very little domain knowledge and an ill-behaved toy car. While others have been able to learn competent control of radio-controlled toy cars, we believe we are by far assuming the least about the system we are modelling. Minimising use of domain knowledge might be seen as an academic concern in the current context, but becomes all the more important in the domains we intend to investigate next, where domain knowledge is often lacking. Additionally, our concern ties in very well with the general spirit of evolutionary robotics, which is to minimise human interference in the process of controller design. We expect lessons learned and techniques developed here, in particular the use of multi-model controller evolution (which to some extent was the "trick" that finally made our experiments work) to be transferable to our other projects.

6.1 Future work

This paper was a first step towards a stable framework for the modelling of physical machines. In the near future we want to refine the methodology used here in order to produce a more flexible and general approach, which can be used in our various other projects. We do not mind using systems with greater domain knowledge, but we do mind handcrafted modelling. In addition we are planning to use a physics simulator as an aid to the modeling process, the idea being to inject non-specific domain knowledge about the laws of physics (gravity, momentum etc.).

7. REFERENCES

[1] P. Abbeel, M. Quigley, and A. Y. NG. Using inaccurate models in reinforcement learning. In *International Conference on Machine Learning (ICML) Pittsburgh PA, USA*, 2006.

[2] J. C. Bongard, V. Zykov, and H. Lipson. Automated synthesis of body schema using multiple sensor modalities. In *Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems (ALIFEX)*, pages 220–226, 2006.

[3] R. De Nardi and O. Holland. Ultraswarm: A further step towards a flock of miniature helicopters. In *Proceedings of the SAB Workshop on Swarm Robotics*, 2006.

[4] O. Holland, editor. *Machine consciousness*. Imprint Academic, 2003.

[5] N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, 6(2):325–368, 1997.

[6] S. M. Lucas and J. Togelius. Point-to-point car racing: an initial study of evolution versus temporal difference learning. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.

[7] H. Marques, J. Togelius, M. Kogutowska, O. E. Holland, and S. M. Lucas. Sensorless but not senseless: Prediction in evolutionary car racing. In *Proceedings of the IEEE Symposium on Artificial Life*, 2007.

[8] J. Sjöberg, Q. Zhang, L. Lijung, A. Benveniste, B. Deylon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky. nonlinear black-box modeling in system identification: a unified overview. *Automatica*, 31, 1995.

[9] J. Togelius and S. M. Lucas. Arms races and car races. In *Proceedings of Parallel Problem Solving from Nature*. Springer, 2006.

[10] J. Togelius and S. M. Lucas. Evolving robust and specialized car racing skills. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2006.