Int. J. Appl. Math. Comput. Sci., 2014, Vol. 24, No. 3, 579–597
DOI: 10.2478/amcs-2014-0042

amcs

IMITATION LEARNING OF CAR DRIVING SKILLS WITH DECISION TREES AND RANDOM FORESTS

PAWEŁ CICHOSZ, ŁUKASZ PAWEŁCZAK

Department of Electronics and Information Technology
Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: p.cichosz@elka.pw.edu.pl,l.pawelczak@stud.elka.pw.edu.pl

Machine learning is an appealing and useful approach to creating vehicle control algorithms, both for simulated and real vehicles. One common learning scenario that is often possible to apply is learning by imitation, in which the behavior of an exemplary driver provides training instances for a supervised learning algorithm. This article follows this approach in the domain of simulated car racing, using the TORCS simulator. In contrast to most prior work on imitation learning, a symbolic decision tree knowledge representation is adopted, which combines potentially high accuracy with human readability, an advantage that can be important in many applications. Decision trees are demonstrated to be capable of representing high quality control models, reaching the performance level of sophisticated pre-designed algorithms. This is achieved by enhancing the basic imitation learning scenario to include active retraining, automatically triggered on control failures. It is also demonstrated how better stability and generalization can be achieved by sacrificing human-readability and using decision tree model ensembles. The methodology for learning control models contributed by this article can be hopefully applied to solve real-world control tasks, as well as to develop video game bots.

Keywords: imitation learning, behavioral cloning, decision trees, model ensembles, random forest, control, autonomous driving, car racing.

1. Introduction

Autonomous vehicle control is a challenging task that has a huge scope of possible applications. It ranges from the video game industry (e.g., developing computer-operated opponents for human players), to car security and assistance systems (e.g., advanced lane guard and parking assistance systems), to exploration robots used in dangerous environments (e.g., ocean exploration, extra-terrestrial exploration). Many practically useful solutions in these areas already exist, but they still leave a considerable space for improvements that justifies further research involving new techniques.

Machine learning is a particularly appealing approach to creating vehicle control algorithms, both for simulated and real vehicles. This is because it can be used without precise and explicit domain knowledge and is capable of adapting to environment changes. While its potential utility has been demonstrated in a number of case studies (Pomerleau, 1988; Togelius *et al.*, 1996; Baluja, 1996; Kröedel and Kuhnert, 2002; Forbes, 2002; Munoz *et al.*, 2009; Cardamone *et al.*, 2009b; 2009a; 2010; Loiacono *et al.*, 2010), there is no single learning scenario and algorithm that proved universally successful and several possibilities remain unexplored.

This article addresses the task of simulated racing car control and adopts the imitation learning scenario (Chambers and Michie, 1969; Urbancic and Bratko, 1994; Atkeson and Schaal, 1997; Bratko *et al.*, 1998; D'Este *et al.*, 2003; Sammut *et al.*, 1992), in which an existing exemplary controller is used to generate training instances, needed to inductively learn a classification or regression model representing necessary control skills. This is widely and easily applicable for all control tasks that can be satisfactorily performed by humans, while their complexity prevents formulating hard-coded (pre-designed) control algorithms. Whereas a self-learning paradigm, such as reinforcement learning (Sutton and Barto, 1998; Kaelbling *et al.*, 1996), could appear more attractive and powerful, imitation learning is arguably more straightforward to apply and should be preferred whenever an appropriate source of training instances is available. It is particularly likely to be superior when seeking for a reliable and reusable control modeling procedure that can be used across a variety of

related tasks.

Unlike in most prior work on learning control through imitation, and probably all prior work on learning driving skills, a symbolic decision tree (Breiman *et al.*, 1984; Quinlan, 1986; 1993) representation of learned models is used here. In contrast to more popular subsymbolic methods, including neural networks (Hertz *et al.*, 1991), this makes the created models easy to inspect, verify, and possibly tune by human experts, and their control decisions can be explained in a meaningful way. This is an important advantage for at least some practical instantiations of the vehicle control task. Decision trees can be also learned in reasonably short time, even from large training sets. When sufficient computational power is available and loss of model comprehensibility is acceptable, decision tree ensembles are known to usually achieve better prediction quality because they are more stable (robust with respect to training data perturbations) and less prone to overfitting (Dietterich, 2000). Of those, random forests (Breiman, 2001) have proved particularly successful. They are also employed in this work to verify how much improvement over single tree models they can provide. One of primary motivations for this work is to see how algorithms that are usually employed for data mining applications (Witten and Frank, 2005; Han and Kamber, 2006) perform in a realistically complex control task.

One issue that is crucial for successful application of inductive learning in general and decision trees in particular is overfitting prevention. It becomes a particular challenge for imitation learning, where it may be quite easy but also quite useless to exactly mimic the behavior of the examplary controller for a single particular task instantiation (e.g., driving a single car on a single track), without generalizing properly to other instantiations of the same task (e.g., driving different cars on different tracks). Standard overfitting prevention techniques, such as decision tree pruning (Breiman *et al.*, 1984; Quinlan, 1993; 1999; Esposito *et al.*, 1997), reducing the level of fit to the training data with the hope to better generalize to new data, may be insufficient for successfully performing control tasks. This is because even a good model with respect to standard inductive learning quality criteria may be unable to solve a control task. By occasionally taking different actions than the exemplary controller, the model-based controller is likely to considerably depart from the state-action trajectories represented by training instances. Finding itself in state space regions not covered by the training set, it is likely to make wrong decisions ultimately leading to failure. We address this challenge by control model retraining, applied to incorporate additional training instances generated in a special active learning mode, automatically triggered on failure.

The utility of the approach proposed in this article is experimentally demonstrated using *The Open Racing Car Simulator* (TORCS)[1], known for its good level of physical realism, clear interface for connecting custom driving algorithms, and a collection of high-quality pre-programmed bot drivers. One of those, exhibiting a high skill level, is used as the exemplary driver for imitation learning. This makes the presented results objective and reproducible, which would be hardly possible with a human driver.

The article is organized as follows. Section 2 briefly summarizes the essential background information on decision trees and random forests used to represent control models as well as the TORCS simulator, with its state and control action representation. It also reviews some most closely related prior work. In Section 3 the proposed modeling procedure is described, including training set generation, model creation, and the active retraining technique. Section 4 presents the results of experimental studies conducted to verify the utility of this procedure and Section 5 summarizes the article, as well as outlines possible future research directions.

2. Background

One of basic assumptions of this article is to build a novel and useful solution upon well known and standard building blocks, so that the same approach can be easily reused on other related tasks. This is why we use decision trees and decision tree model ensembles that are much more than adequately described in the literature as well as their R language (R Development Core Team, 2010) implementations that are easily and freely available. Also the TORCS simulation environment is both easily available and well described to enable reproducing the results or extending the techniques presented here. This section can be therefore limited to providing bare minimum information on the algorithms and the simulation environment, necessary to follow the foregoing discussion of the proposed approach.

2.1. Decision trees. A decision tree (Breiman *et al.*, 1984; Quinlan 1986; 1993) is a hierarchical structure that represents a classification model, i.e., a mapping of instances from a given domain to a finite set of classes. Internal tree nodes represent splits applied to decompose the domain into regions, and terminal nodes assign class labels to regions believed to be sufficiently small or sufficiently uniform. For convenience, we will reserve the term *node* to internal nodes only and refer to terminal nodes as *leaves*.

2.1.1. Model representation. Splits are specified by some relational conditions, based on selected single attributes, that may have two or more outcomes. Formally,

[1] `http://www.torcs.org`.

a split can be represented by a test function that maps instances into split outcomes. A separate outgoing branch is associated with each possible outcome of a node's split. If the split's outcome can be unambiguously determined for any possible instance, then it does partition the domain into disjoint subsets, corresponding to the outgoing branches. It is therefore easy to see that each node or leaf of a decision tree corresponds to a region (subset) of the domain.

Looking from a different perspective, any instance can be "passed down" from the root node, along branches corresponding to the outcomes of consecutive splits, to a corresponding leaf. This shows that, under the assumption of each split assigning one and only one outcome to any instance, a decision tree represents a mapping of all instances from the domain to the set of its leaves. Now if we further assume that each leaf stores exactly one class label, then a decision tree can be seen as a representation of a classification model.

2.1.2. Growing. Decision tree growing (Quinlan, 1986) is a sequential process during which new nodes or leaves are added step by step in a top-down fashion, starting from a single root node. It involves the following major operations:

- class distribution calculation based on the corresponding subset of training instances,
- checking the stop criteria, which determine whether a node or a leaf will be created,
- class label assignment for leaves (usually performed for nodes as well),
- split selection for nodes (usually based on the resulting class distribution impurity),
- split application, i.e., partitioning the current subset of training instances into subsets corresponding to split outcomes.

2.1.3. Pruning. Decision tree pruning (Quinlan, 1999; Esposito *et al*., 1997) is an insurance policy against overfitting motivated by Ockham's razor (Mitchell, 1997). It can be considered an inverse of growing that results in cutting off some overgrown subtrees and replacing them by leaves with the intention to improve the tree's generalization capabilities. In many cases pruning turns out to be a good way of achieving good generalization with decision trees.

2.1.4. Prediction. Using decision trees to predict classes for arbitrary instances from the domain for which it was created is straightforward and computationally efficient. It reduces to consecutively applying splits until each instance to be classified reaches a leaf, containing a class label.

2.1.5. Implementation. The particular decision tree implementation used for this work is the `rpart` package (Therneau and Atkinson, 1997) for the R language (R Development Core Team, 2010). It can be considered a clone of the well-known CART algorithm (Breiman *et al*., 1984). Its main features include

- using binary splits, based on equality or subset membership conditions for discrete attributes and inequality conditions for continuous attributes;
- stop criteria, including
  - reaching less than a specified minimum number of instances required for a split (the `minsplit` parameter),
  - no possibility to reduce the cross-validated misclassification error by at least a specified complexity parameter value, representing the cost of adding a node to the tree or the minimum error improvement required for a split (the `cp` parameter);
- split evaluation based on the Gini index (Breiman *et al*., 1984) or the entropy (Quinlan, 1986), which are two most common impurity measures;
- cost-complexity pruning (Breiman *et al*., 1984) based on identifying a complexity parameter value that yields an acceptable balance between the size and cross-validated misclassification error of the tree;
- support for instance weights.

2.2. Random forest. When the human-readability of models is not required, better predictive performance can be obtained by combining a number of different decision trees created for the same domain. This idea of ensemble modeling (Dietterich, 2000) is in principle applicable to arbitrary classification or regression models, but has become particularly successful with decision trees due to their instability. It makes it easy to obtain diverse decision trees by using multiple perturbed copies of the original training set.

A random forest (Breiman, 2001) is an ensemble model represented by a set of unpruned decision trees, grown based on multiple bootstrap samples drawn with replacement from the training set, with randomized split selection. It can be considered an enhanced form of bagging (Breiman, 1996), which additionally stimulates the diversity of individual models in the ensemble by randomizing the decision tree growing algorithm used to create them.

2.2.1. Growing. Random forest growing consists in growing multiple decision trees, each based on a bootstrap sample from the training set (usually of the same size as the original training set), by using a standard decision tree growing algorithm. Each bootstrap sample can be easily seen to contain about 63.2% instances from the original training set (with multiple copies of some), and the remaining 36.8% instances are referred to as Out-Of-Bag (OOB) instances.

Since the expected improvement of the resulting model ensemble over a single model is contingent upon sufficient diversity of the individual models in the ensemble (Breiman, 1996; Dietterich, 2000), the following modifications are applied to stimulate the diversity of decision trees that are supposed to constitute a random forest:

- trees are grown with maximally restrictive stop criteria (until reaching a uniform class, exhausting the set of instances, or exhausting the set of possible splits),
- whenever a split has to be selected for a tree node, a small subset of available attributes is selected randomly and only those attributes are considered for candidate splits.

Individual trees built that way are likely to be overfitted, but nevertheless no pruning is applied to them, so that each of them remains maximally fitted to the training set. With the random internal attribute selection this gives them many opportunities to differ, though, and their individual overfitting is effectively canceled out if they are used as an ensemble.

2.2.2. Prediction. Random forest prediction is achieved by simple unweighted voting of individual trees from the model. With sufficiently many diversified trees (typically hundreds), this simple voting mechanism usually makes random forests extremely accurate and resistant to overfitting. As a matter of fact, in many cases they belong to the most accurate classification models that can be achieved. Unfortunately, as model ensembles, they lose much of human readability that is such an important advantage of decision trees.

2.2.3. Implementation. The particular random forest implementation used in this work is the `randomForest` package (Liaw and Wiener, 2002) for the R language (R Development Core Team, 2010), which is based on the original Fortran code developed by Breiman and Cutler.

2.3. TORCS environment. TORCS is both a realistic racing car simulator and an enjoyable racing video game. The simulation engine takes into account several factors that impact the behavior of the car on the track, such as torque, tyre adhesion, drivetrain, or aerodynamics, and makes it possible to control the car via the steering wheel, acceleration and brake pedals, as well as the clutch and gear shifter (if using manual transmission). It provides 3D visualization and a number of tracks, with more relatively easy to create, and a choice of several car models. More importantly, it also includes several pre-programmed bot drivers, and additional bots can be created essentially by defining a custom C function (Wymann, 2006). This makes it a convenient platform for research on autonomous vehicle control that has recently become quite popular (Munoz *et al.*, 2009; Cardamone *et al.*, 2009b; 2009a; 2010; Loiacono *et al.*, 2010).

2.3.1. State representation. A TORCS bot can perceive the track and its car situation through a number of attributes of two basic types:

track description: attributes describing the properties of consecutive short track segments in front of the car, including

- type (straight, left turn, or right turn),
- length,
- width,
- turn radius,

car status: attributes that represent the car status maintained by the simulation engine, including

- the distance between the center of the car and the middle of the track,
- the angle between the axis of the car and the axis of the track,
- engine RPM,
- velocity (represented by its x- and y-components, in the direction of the track and perpendicular to the track).

It is a common practice in car control studies to adopt a driver-perspective track description in the form of rangefinder sensors, providing distances between the car and track edges measured along a set of beams (Togelius *et al.*, 2006; Kohl *et al.*, 2006; Cardamone *et al.*, 2009b). This rangefinder sensor state representation can be easily calculated from the segment and car status information originally provided by TORCS. Following Loiacano *et al.* (2009), we use rangefinder sensors directed every 10° in the $[-90^\circ, 90^\circ]$ range, as illustrated in Fig. 1. We also use all car status attributes as listed above, and a limited set of additional segment-derived attributes consisting of

- the current and next segment type,

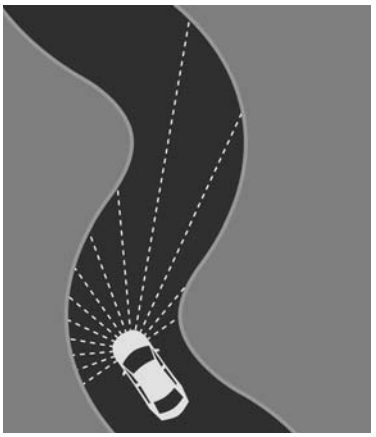

Fig. 1. Rangefinder sensors for the TORCS environment.

- the turn radius values for the the next 8 equal-length 10-meter track sections,
- the distance to the next segment of type different from the current segment type.

2.3.2. Control actions. Control actions available for a TORCS bot are specified by setting the following parameters:

steering: continuous in the $[-1, 1]$ range,

acceleration: continuous in the $[0, 1]$ range,

brake: continuous in the $[0, 1]$ range,

gear: discrete integer in the $[-1, 7]$ range.

These represent the steering wheel, the acceleration and brake pedals, and the gear shifter of the car, respectively. For simplicity, they all will be referred to as *actions* thereafter. The car driving task in the TORCS environment (just like the real car driving task) is therefore a *multidimensional* control task, where the control model is required to generate a vector of multiple control actions. While TORCS makes it possible to also control the clutch, this capability is not used in our experiments (the clutch is operated automatically).

2.4. Related work. This article adds its contribution to the on-going effort in the area of learning control behavior, dated since at least the 1960s. A variety of learning tasks and algorithms have been considered over the last five decades. Of those, particularly relevant to this article are all studies on learning by imitation (Chambers and Michie, 1969; Urbancic and Bratko, 1994; Atkeson and Schaal, 1997; Bratko *et al.*, 1998; D'Este *et al.*, 2003; Sammut *et al.*, 1992), especially those addressing the vehicle control task, either in the TORCS environment (Munoz *et al.*, 2009; Cardamone *et al.* 2009b; 2009a; 2010) or another simulated or real environment (Pomerleau, 1988; Togelius *et al.*, 1996; Baluja, 1996). Other approaches to this task that do not follow the imitation learning scenario, including those based on reinforcement learning (Krödel and Kuhnert, 2002; Forbes, 2002; Loiacono *et al.*, 2010), even if adopt substantially different assumptions about the available training information and use different learning algorithms, need to face the same crucial issues of state information and control action representation. In these respects, this work borrows substantially from many of those prior solutions.

Interestingly, relatively few previous imitation learning demonstrations used symbolic knowledge representation and learning algorithms. Subsymbolic techniques, such as neural networks (Hertz *et al.*, 1991), have received much more attention in this context (Pomerleau, 1988; Togelius *et al.*, 1996; Baluja, 1996; Cardamone *et al.*, 2009a; 2010; Anderson *et al.*, 2000). While there are some noteworthy exceptions to this general trend (Urbancic and Bratko, 1994; Bratko *et al.*, 1998; Sammut *et al.*, 1992; Sammut, 1996; D'Este *et al.*, 2003), none of them addresses the vehicle control task.

This article follows the less popular symbolic imitation learning path using decision trees (Breiman *et al.*, 1984; Quinlan, 1986; 1993), the most widely used symbolic learning algorithm family capable of producing models that are both accurate and human-readable, and decision tree ensembles (Dietterich, 2000; Breiman, 1996; 2001) that sacrifice the comprehensibility for greater stability and overfitting resistance. It successfully addresses all major challenges encountered when attempting their application to cloning car driving skills, which differs from popular data mining applications (Witten and Frank, 2005; Han and Kamber, 2006) in several important ways.

The relations of this research to prior work using both the learning by imitation approach and the TORCS environment are particularly close and deserve more detailed comments. While this work has much in common with them, in has some unique qualities that hopefully make it interesting and useful:

- employing a symbolic decision tree model representation for best comprehensibility and random forest for improved stability and generalization,
- applying a retraining technique to compensate for insufficient representativeness of training instances and adapt to different state trajectories,
- using several diverse test tracks of various

complexity levels, unseen during model training, for performance evaluation,

- using relatively low-level unprocessed state and action representation almost directly corresponding to the task's natural state and action space, without any smart intermediate processing layer typically incorporated to make learning easier.

The latter deliberately departs from the path that is suggested by previous findings (Cardamone *et al*., 2009b), indicating that high-level state representation may lead to better learning results. It is intended to keep our approach generic and easily reusable, with the option to incorporate refinements in the future work if it turns out promising.

Successful demonstrations of autonomous real car driving, including the DARPA Grand Challenge (Buehler *et al.*, 2007; 2009) and Google's self-driving cars (Thrun, 2010; Guizzo, 2011), show that learning driving skills is just one of numerous algorithmic approaches that may be regarded as building blocks of future driverless vehicles. The issue of modeling and control addressed by imitation learning may appear to be essential, but several other issues may be similarly or more challenging. These include, in particular, sensor design and calibration, localization, object recognition, and trajectory planning (Levinson *et al*., 2011). What may turn out to be crucial for practical real-world applications is the perception of terrain conditions and road smoothness using sensors as well as visual features extracted from camera images (Stavens, 2011). These issues are entirely beyond the scope of this article.

3. Control models

This section presents the details of the modeling procedure used to achieve control models represented by decision trees and decision tree ensembles. While developed by trial-and-error to some extent, based on experimenting with the TORCS environment, it is presented here in a systematic and mostly task-independent way, with some specific issues directly related to TORCS postponed to the next section.

3.1. Model design. Our approach to learning control models is based on the following principles:

1. *A control task with multidimensional control actions is decomposed into multiple control tasks with one-dimensional actions, solved in parallel by separate control models operating without interactions.* For the TORCS environment, one will therefore have separate control models for the steering, acceleration, brake, and gear parameters, which jointly represent the complete control model for a simulated car.
2. *A control task with one-dimensional continuous actions is simplified to a control task with one-dimensional discrete actions using one of the following two approaches:*

   plain discretization: continuous control actions are directly discretized into a number of intervals, with single representatives of each interval used as discrete control actions,

   differential discretization: continuous control action changes (i.e., action differences for two consecutive time steps) are discretized, with single representatives of each interval used as discrete control actions.

   In the experiments reported in this article the steering, acceleration, and brake continuous actions will be discretized using the plain discretization approach, with the investigation of possible advantages of differential discretization postponed to future work.
3. *A control task with one-dimensional discrete actions is considered a classification task, with class labels corresponding to its possible discrete actions.* For the TORCS environment, discrete actions corresponding to the steering, acceleration, brake, and gear parameters will be treated as class labels, to which state vectors are to be mapped by classification models.
4. *All classification models dedicated to individual one-dimensional control tasks are trained from the same set of training instances, with different class labels (appropriate for particular tasks).* For the TORCS environment, a single set of training instances, based on the behavior exhibited by the exemplary driver, is used to train multiple separate classification models for the steering, acceleration, brake, and gear parameters, with just class labels assigned differently.
5. *When applying classification model predictions to the control task, class labels for discretized actions are converted back to the original continuous representation using discretization interval representatives.* For the TORCS environment, the steering, acceleration, and brake actions will be obtained as the representatives of the discretization intervals corresponding to the predicted class labels.

These principles, along with the one about using decision trees and random forests for model representation (as already stated several times before) constitute the foundations of this article's approach to learning control models. They can be hardly considered surprising or unobvious, being mostly rather

straightforward and common in prior work on imitation learning. Two of them deserve more justification, as apparently oversimplified: multidimensional action decomposition and continuous action discretization. It is not unlikely that they both indeed limit the quality of control policies that can be represented and acquired. They are both currently useful, though, since they make it possible to apply any standard general-purpose classification algorithms to create control models.

Dealing with the original multidimensional action space would require some way of detecting, representing, and incorporating dependencies between different action parameters to the modeling process. Some of them may be quite simple and suggested by background knowledge (e.g., for the acceleration and brake actions) and some other much more complex and unobvious. Dealing with the original continuous actions without discretization would require using regression, rather than classification algorithms. Both these enhancements are certainly worth addressing by future work, but sticking with the simplified approach makes it possible to focus on investigating the capabilities of classification algorithms and refining the learning scenario before looking for the best possible quality.

The way from such basic principles to a truly useful solution that delivers satisfactorily working control models may be long and quite tricky, though, requiring in particular appropriate modeling techniques to be employed. The remainder of this section gives an account of the latter.

3.2. Training sets. Training sets generated from the exemplary control behavior to be imitated consist of input state vectors accompanied by several class labels, corresponding to each classification task to which the original control task is decomposed. State vectors contain task-specific attribute values that are left in their original form. Each state vector is accompanied by the corresponding control action vector, with actions discretized as described above.

3.3. Retraining. One issue with imitation learning that appears not to have received sufficient attention in previous studies is the possible harmful effect of the excellence of the exemplary behavior. Paradoxical as it may seem, if the exemplary behavior is perfect or near-perfect, it is likely not to provide sufficiently good training instances, given the inherent imperfection of inductive learning. Small error levels that might be perfectly acceptable for ordinary classification or regression tasks may prevent the models from performing the corresponding control task entirely, if they diverge from the state-action trajectories represented by training instances. Clearly, even very sparingly performed wrong actions may ultimately lead to control failures, if the control system is unable to recover from unexpected and undesirable situations encountered. Training instances generated from perfect exemplary behavior are unlikely to demonstrate such recovery procedures, though, as they are normally not necessary during perfect or near-perfect operation. Even if incorrect actions are taken in non-critical, safe states, their cumulative effect may expose the system to states in which it is unable to behave correctly. This is the challenge addressed by retraining.

3.3.1. Failure avoidance. The primary objective of retraining is to learn how to avoid control failures by successfully recovering from potentially dangerous situations. The idea is to monitor the performance of the obtained control model and detect control failures, which then automatically trigger an active learning mode, in which additional training instances are generated. They demonstrate the actions that would be performed by the exemplary controller in a number of recent states, specified as the retraining window, which preceded the failure. These additional training instances can be then used to improve the learned control models.

Ideally, retraining should be handled with an incremental learning algorithm (Mitchell, 1997; Cichosz, 2007), which is able to modify an existing model based on new data, such as the incremental decision tree induction algorithm (Utgoff, 1989). Since the decision tree implementation used for this work is not incremental and the storage and computation cost is not an issue anyway, currently we simply build new models from training sets containing both the original training instances and the newly generated ones. In a sense, this may be viewed as an opposite of the iterative refiltering procedure proposed by John (1996), augmenting decision tree pruning by removing noisy instances from the training set and tree re-growing.

It may be reasonable to put different weights to the original and newly generated training instances (if using a modeling algorithm capable of handling instance weights) or resample the training set appropriately (otherwise) to balance the sensitivity of the retrained model to the old and new training data. For the `rpart` implementation of decision trees used in this work, instance weights can be specified via the `weights` parameter.

Some care is needed when choosing the number of states preceding the failure used for generating new training instances. Too little may be insufficient to demonstrate how the exemplary driver avoids the failure in a dangerous situation, particularly if it requires an early reaction. Too many, on the other hand, may permit the exemplary driver to avoid entering the dangerous situation at all (just like when generating the original training set), making retraining ineffective.

3.3.2. Overfitting prevention. Retraining can be also regarded as an insurance policy against overfitting, supplementing or replacing standard overfitting prevention techniques. It is triggered whenever the control model faces a situation it cannot correctly react to. As discussed above, this may be the cumulative effect of occasionally taken incorrect actions. An overfitted model is more than likely to fail in state space regions not represented in the training set. If retraining prevents such failures, it effectively prevents overfitting.

It is worth stressing that this approach may be expected to work even if the particular modeling algorithm and parameter setup tend to overfit. This is the case, in particular, with maximally grown unpruned decision trees. While the initial models created on the original training set are indeed more than likely to be overfitted, subsequent retraining cycles reduce the overfitting by incorporating new instances, corresponding to states for which the previous models were unable to perform as expected. This may ultimately lead to models that are no longer overfitted to the original training set. They may still be overfitted to the extended training set (with retraining instances added), but this is not necessarily harmful since the latter much better represents the diversity of situations the control model may encounter during its application. While such models may be still unable to drive on tracks different from the one used for training, they can be expected to perform robustly with respect to their own mistakes.

Fighting overfitting by adding more training instances is in striking contrast with the above-mentioned iterative refiltering procedure (John, 1996) which attempts to achieve the same objective by removing training instances considered harmful. Paradoxical as it may seem, these two opposing techniques both make sense depending on what makes the modeling algorithm more likely to overfit: data noise, as in many data mining applications, or insufficient data representativeness, as in our case.

4. Experiments

The modeling procedure described above was applied to create bot drivers for the TORCS environment and experimentally evaluate their performance. This section describes the experimental procedure and presents the obtained results.

4.1. Experimental design. The goal of the experiments is to provide an initial unbiased assessment of the capabilities of the imitation learning technique proposed in this article rather than to optimize the driving performance by any means. They are therefore purposely designed to be simple and easily reproducible, with no unnecessary tweaks.

Fig. 2. Training track (*CG Speedway number 1*, left) and test track (*CG Speedway 2*, right).

4.1.1. Tracks. Experiments reported in this article use both existing and custom TORCS tracks for creating and evaluating control models. They are presented below.

Simple tracks. Our initial experiments use two standard TORCS tracks, referred to as *simple tracks* thereafter:

training track: *CG Speedway number 1*—used to generate training instances,

test track: *CG Track 2*—used to evaluate the driving performance.

The two tracks are illustrated in Fig. 2. The overall complexity of these tracks is similar, although the latter has some sharper bends. Models created based on training sets from the training track are evaluated both on the training track, to see how well they manage to imitate the exemplary behavior, and on the test track, to see whether the acquired skill can be transferred to a modified environment.

Complex tracks. Further experiments use more challenging tracks, referred to as *complex tracks* thereafter:

training track: a custom track of relatively large length and complexity, designed to adequately represent typical bend patterns occurring in standard TORCS tracks—used to generate training instances,

test tracks: all standard TORCS road tracks—used to evaluate the driving performance.

The training track is presented in Fig. 3 and the test tracks in Fig. 4. It is worth stressing that, unlike in some other studies, we keep the training and test tracks strictly separate. This makes it possible to observe the performance of the learned control models on several different *previously unseen* tracks.

4.1.2. Training data. Training instances for imitation learning are generated using the *Inferno* bot distributed with TORCS as the exemplary driver. It can be considered one of TORCS masters, with fast and aggressive driving style that is really hard to beat. The training sets for each

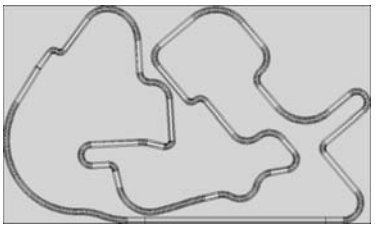

Fig. 3. Custom complex training track.

Table 1. Continuous action discretization intervals and their representative values.

action	interval	representative
steering	$[-1, -0.25]$	-0.25
	$(-0.25, -0.125]$	-0.125
	$(-0.125, -0.05]$	-0.05
	$(-0.05, 0.05)$	0.0
	$[0.05, 0.125)$	0.05
	$[0.125, 0.25)$	0.125
	$[0.25, 1]$	0.25
acceleration	$[0, 0.25)$	0.0
	$[0.25, 0.5)$	0.25
	$[0.5, 1]$	0.5
brake	$[0, 0.25)$	0.0
	$[0.25, 0.5)$	0.25
	$[0.5, 1]$	0.5

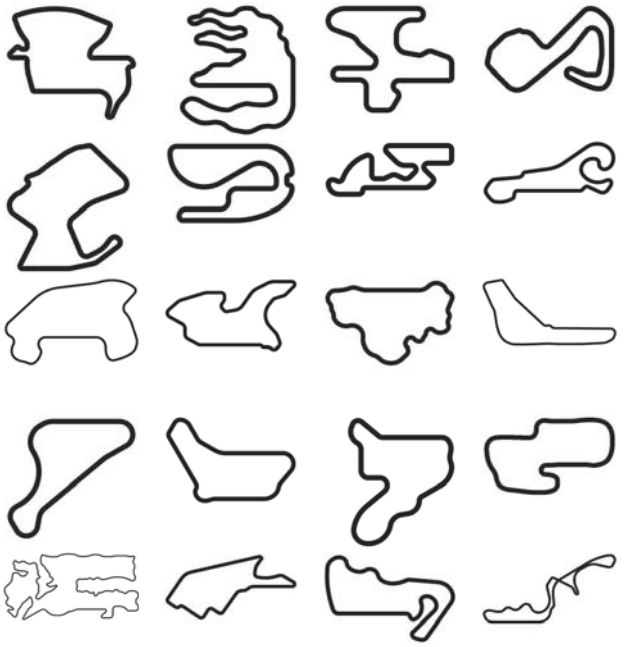

Fig. 4. Set of TORCS road tracks (left to right, top to bottom: *Aalborg*, *Alpine-1*, *Alpine-2*, *Brondehach*, *Corkscrew*, *E-track-1*, *E-track-2*, *E-track-3*, *E-track-4*, *E-track-6*, *Eroad*, *Forza*, *G-track-1*, *G-track-2*, *G-track-3*, *Ruud-skogen*, *Spring*, *Street-1*, *Wheel-1*, *Wheel-2*).

model consist of state–action pairs encountered during the first two laps when driving the training track from a steady start. Whereas using a bot rather than human drivers departs from the traditional form of behavioral cloning, it makes the experimental evaluation process more objective and reproducible.

Continuous state attributes are used directly as provided by the TORCS environment or calculated, without any transformations. The discrete *gear* action is left unmodified. Continuous actions are discretized using the plain discretization approach, with the discretization intervals and their representatives given in Table 1. These are manually chosen based on preliminary experiments to provide sufficient but not excessive resolution.

Such a moderate discretization resolution is definitely sufficient for good driving performance, given the fact that cars in TORCS (and many other racing games) can be controlled successfully by human players using binary actions communicated through a computer keyboard. Notice that the less extreme interval boundaries are chosen as interval representatives rather than the middle values (except for near-0 steering). This introduces a kind of conservative bias to control action selection. In particular, the range of steering values used by the control model is restricted to 25% and the range of acceleration and brake power to 50% of the corresponding maximum values permitted by the simulator. While this may prevent the control model from exhibiting a very aggressive driving style, it is perfectly acceptable for the initial studies reported here.

4.1.3. Model application. Decision tree models produced by `rpart` and converted to C code (series of **if-else** statements) are used to implement a TORCS bot. This straightforward conversion is performed by a custom utility developed to enable performing multiple experiments automatically. For random forest models their individual components are extracted, similarly converted to C code, and combined by simple voting.

If the models are not of sufficiently good quality, the following control failure situations may occur:

- leaving the track (detected if the normalized distance between the center of the car and the middle of the track exceeds 1.1),
- heading the wrong direction.

Of course, the control models cannot be expected to recover from such failures autonomously, since they are not trained to do so. To make it possible even for poor models to continue driving, a recovery process is activated

to put them back in the middle of the track, facing the right direction. This is achieved by temporarily passing control to the exemplary driver, for the number of steps sufficient for recovery. Of the potentially possible failure situations listed above, only the first actually occurred in our experiments.

4.1.4. Retraining. Retraining is performed by applying the control model on a track for two laps (in the case of the experiments reported here, it is the same training track used to generate the initial training set) and generating new training instances whenever a control failure situation arises. When this happens, a set of additional training instances is generated, containing the last 100 states before the failure and the corresponding exemplary driver's actions. This number of preceding states has been found to be sufficient to demonstrate how the exemplary driver avoids the failure. If the control model is found to be unable to continue driving (i.e., it encounters two more failures within a 10 m distance), the run is terminated.

In preliminary experiments the decision tree control models sometimes exhibited the overly conservative behavior of slowing down too much on straights. While this is not, strictly speaking, a failure, it effectively prevents the control model from completing a lap in reasonable time. Postponing the issue of stimulating a more aggressive driving style for future work, this has been also solved using retraining. Whenever the decision tree bot running at 10 m/sec or slower on a straight starts to brake, retraining instance generation from the previous 100 steps is triggered, as on failure. Moreover, the exemplary driver takes control over the car for the next 100 steps to get it back to a normal speed. If the slowing down situation repeats two more times, the run is terminated.

After each retraining run, the combined dataset, consisting of the initial training set and all the retraining instances generated so far, is used to create a new control model. This completes a single retraining cycle. The retraining process consists of multiple cycles and is terminated when no new retraining instances are generated.

4.1.5. Performance measures. The following model performance measures are reported:

distratio: the distance traversed within the evaluation period, normalized by dividing by the corresponding distance traversed by the exemplary driver,

failcount: the number of control failures within the evaluation period.

4.1.6. Experimental studies. The experiments reported in this article are organized into the following two studies:

single tree models, simple tracks: using single fully grown decision trees as control models for simple tracks,

single tree models, complex tracks: using single fully grown decision trees as control models for complex tracks,

random forest models, complex tracks: using random forest ensembles as control models.

Results of experiments using pruned decision trees are not reported and were generally disappointing due to extreme instability. Cost-complexity pruning (Breiman *et al.*, 1984) available in `rpart` was found to yield unstable results due to the internal cross-validation used to estimate the misclassification error for different complexity parameter ranges, with considerably different pruned trees obtained on multiple runs. Only some of them improved the performance over the maximally fitted unpruned trees.

Using ensembles of pruned trees, with each tree obtained by cost-complexity pruning the same fully grown tree independently a number of times, can be considered a stabilization technique for cost-complexity pruning. Unfortunately, the increased stability leads to some loss of human-readability if control actions are determined by the voting of multiple trees. When the comprehensibility of control models is not required, much better performance can be achieved by random forest models.

For single-tree studies the only learning algorithm parameters that are not kept at fixed default values are

`minsplit`: the minimum number of instances required for a split,

`cp`: the minimum error reduction required for a split.

They are set to 2 and 0.0, respectively, which produces fully grown, maximally fitted trees. The `weight` parameter was used to specify different weights to the original training instances and the new retraining instances in a number of preliminary runs, but it gave no improvement over uniform weights.

For the random forests study there are two parameters that were adjusted:

`ntree`: the number of trees,

`mtry`: the number of attributes randomly picked at each node for split selection.

They are set to 300 and 12, respectively—values roughly optimized in preliminary runs.

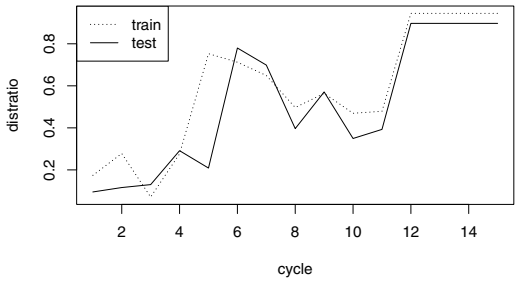

Fig. 5. Normalized distance traversed for Study 1.

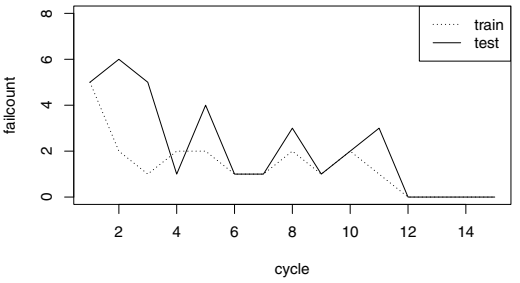

Fig. 6. Number of control failures for Study 1.

The evaluation period is set to 5000 simulation steps for Study 1 and to 15000 simulation steps for Study 2. This is fully adequate for the evaluation purpose, as it is sufficient for the *Inferno* bot to make nearly two full laps from a steady start for the training tracks.

4.2. Results. The obtained results for each of the experimental studies are presented and discussed below.

4.2.1. Study 1: Single tree models, simple tracks. This is the most basic, but the essential experimental study for the article. Its objective is to evaluate the utility of the modeling procedure proposed in this article and the capabilities of decision trees used for control model representation using two relatively simple tracks for training and testing. Although the test track has a similar level of complexity and overall shape as the training track, it is noteworthy that we use imitation learning to acquire control skills on one task instantiation and then apply them on another task instantiation.

The values of the two performance measures for the initial control model and the models obtained after each retraining cycle are plotted (versus the cycle number, with cycle 1 corresponding to the initial training) in Figs. 5 and 6. Figure 7 shows the number of all training instances used to create the initial model (cycle 1) and subsequent retrained models.

The observations may be summarized as follows:

- the initial control model performs poorly on both the training and test track:
  - it makes about a 6 times shorter training track distance and a 10 times shorter test track distance than the exemplary driver during the evaluation period,
  - it falls out of track 5 times;
- it takes 11 retraining cycles and about 3200 newly generated training instances (added to the initial training set of about 5000 instances) to produce a model that exhibits near-perfect performance on the training track and reasonably good performance on the test track:
  - it makes nearly 95% of the distance traversed by the exemplary driver on the training track and nearly 90% on the test track,
  - it never falls out of track;
- the improvement in subsequent retraining cycles is not perfectly monotonic (with some cycles yielding performance degradation).

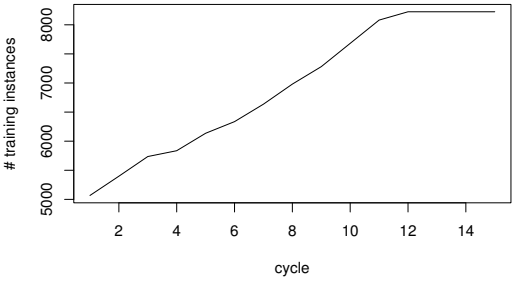

Fig. 7. Number of training instances used for Study 1.

It is particularly striking that the initial control model is unable to successfully imitate the exemplary driver's behavior even on the training track and retraining is essential for delivering satisfactory results. This cannot be attributed to the overfitting of maximally grown decision trees, since none of our preliminary experiments with pruned decision trees delivered successful results for unretrained models. It is therefore indeed the departure from the state-action trajectories demonstrated by the exemplary driver and represented in the original training set that prevents the control model from exhibiting

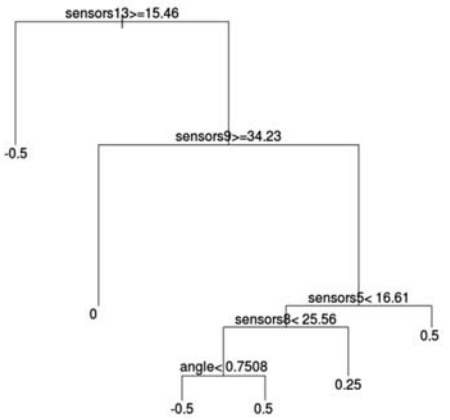

Fig. 8. Top three levels of the steering decision tree for Study 1.

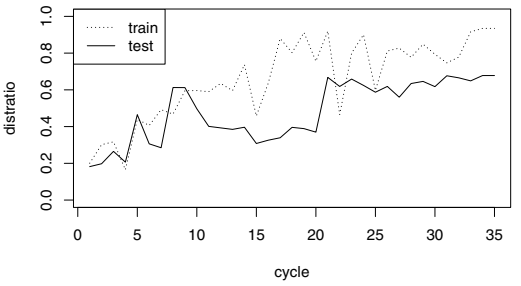

Fig. 9. Normalized distance traversed for Study 2.

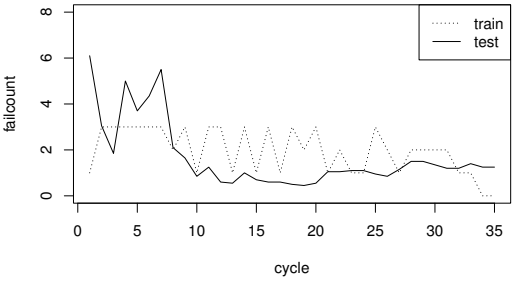

Fig. 10. Number of control failures steps for Study 2.

good performance. Retraining meets the expectation of overcoming this problem.

The size of decision trees obtained in this study reaches several hundred nodes. This should not be surprising given the maximum-fit setup of stop criteria and lack of pruning. Such a large tree size makes the comprehensibility of control models questionable. Indeed, large decision trees can be hardly fully analyzed and understood by a human expert. Nevertheless, their interpretable structure is an important advantage over subsymbolic model representations. Even if the size of the complete tree prevents in-depth inspection, selected smaller subtrees corresponding to particular conditions can be extracted and analyzed. If necessary (e.g., due to a failure situation), control actions taken by the model can be explained by providing a set of conditions that caused the particular action choice. Moreover, a small number of top tree levels may be used to get an approximate understanding of the represented control policy. As an illustration, Fig. 8 presents the top three levels of the decision tree model for the steering action obtained in this study.

4.2.2. Study 2: Single tree models, complex tracks. This study explores the possibility of learning control models applicable to a variety of task instantiations, i.e., capable of driving on several different tracks. It applies the same modeling algorithm and experimental procedure to the custom complex training track and the set of test tracks consisting of all standard TORCS road tracks. The results are presented in Figs. 9 (the normalized distance), 10 (the number of failures), and 11 (the number of training instances). The test track performance measures are averaged over all test tracks used.

The observations may be summarized as follows:

- it takes 34 retraining cycles and nearly 11000 additionally generated retraining instances (in addition to the initial training set of 14500 instances) to reach good performance on the complex training track:
  - more than 93% of the distance traversed by the exemplary driver,
  - no out-of-track failures;
- the discrepancy between the training and test track performance is much greater than observed in Study 1:
  - the control model achieves 68% of the exemplary driver's distance on the average,
  - it falls out of track 1.25 times on the average;
- the retraining process is non-monotonic, with some cycles degrading rather than improving performance.

Given the considerably increased complexity of the training track in comparison to Study 1, the retraining

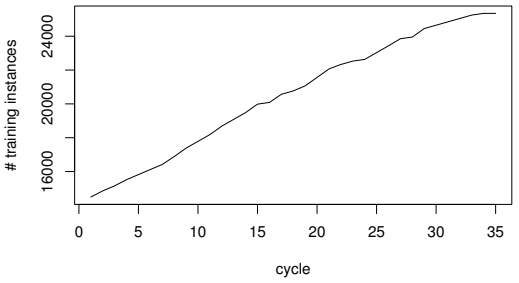

Fig. 11. Number of training instances used for Study 2.

Table 2. Individual test track performance for Study 2.

track	distratio	faicount
Aalborg	0.442	7
Alpine-1	0.823	0
Alpine-2	0.917	0
Brondehach	0.742	0
Corkscrew	0.703	2
E-track-1	0.595	1
E-track-2	0.678	2
E-track-3	0.638	0
E-track-4	0.798	0
E-track-6	0.697	0
Eroad	0.647	0
Forza	0.569	3
G-track-1	0.731	1
G-track-2	0.739	0
G-track-3	0.462	3
Ruudskogen	0.534	3
Spring	0.564	2
Street-1	0.827	0
Wheel-1	0.666	0
Wheel-2	0.786	1

process can be still considered successful, as it yields only slightly worse training track performance than before. Not surprisingly, it takes more than a double number of retraining cycles to complete. Unfortunately, the observed test track performance reveals the limitations of unpruned single tree models. It looks that the test tracks are sufficiently different from the training track and from one another to make it a challenge to successfully drive on all of them. Even small differences in the bend radius and length, the length of straights between bends and the particular sequence of bends and straights may require substantially different control actions.

Table 2 shows the performance of the finally obtained control models on individual test tracks and Fig. 12 presents the histograms of the normalized distance traversed and the number of failures (measured after the

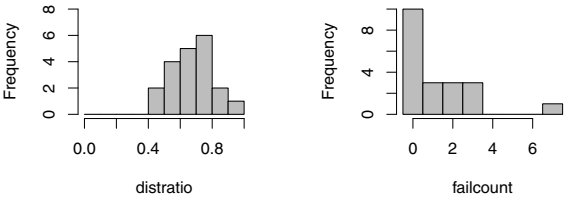

Fig. 12. Histograms of test track performance for Study 2.

final retraining cycle) over all test tracks. This makes it possible to examine the obtained test track performance in more detail and make the following observations:

- the normalized distance traversed exceeds 0.8 on three test tracks and exceeds 0.7 on additional six tracks,
- the normalized distance traversed falls below 0.6 for six tracks,
- for exactly a half of the test track pool the control model encounters no out-of-track failures and it encounters a single failure only on additional three tracks,
- on all but a single test track (*Aalborg*) the number of failures does not exceed 3.

While the test track distance ratio of less than 0.7 may be considered disappointing, the number of failures is actually quite small. Being able to drive without falling out-of-track on ten previously unseen test tracks is a noteworthy achievement. Interestingly, even for no-failure test tracks the distance ratio often falls below 0.8. This shows the learned model's inability to match the exemplary driver's speed.

4.2.3. Study 3: Random forest models, complex tracks. This study repeats Study 2 using the random forest modeling algorithm. Due to its inherent randomness the results are not deterministic and hence are averaged over 10 independent runs, differing only in the initial seed of the random number generator. Standard deviation bars are presented additionally to the mean lines on the plots displayed in Figs. 13 (the normalized distance), 14 (the number of failures), and 15 (the number of training instances).

The following observations can be made:

- near-perfect training track performance is achieved without an excessive number of retraining cycles and instances:

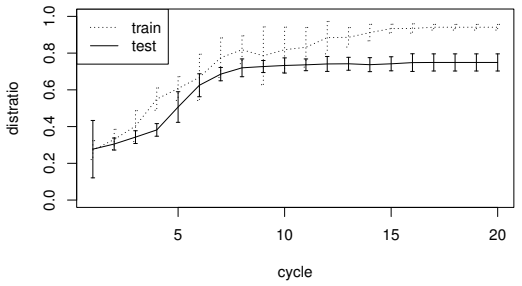

Fig. 13. Normalized distance traversed for Study 3.

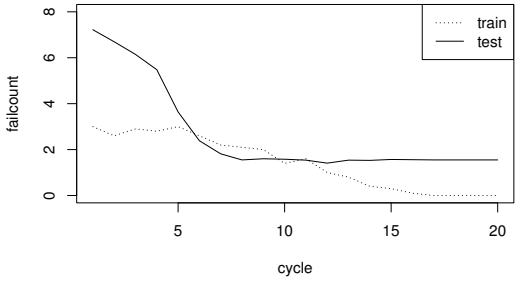

Fig. 14. Number of control failures for Study 3.

  - nearly 95% of the distance traversed by the exemplary driver,
  - while it takes 17 retraining cycles to complete all runs, most of them last no more than 12–13 cycles,
  - the number of retraining instances used is only about 3500, considerably less than in the previous study;
- there is still discrepancy between the test track and training track performance:
  - the average test track normalized distance of about 0.75,
  - about 1.5 out-of-track failures cannot be avoided on average;
- individual experimental runs differ, but the final training track distance ration is extremely repeatable with a small standard deviation of about 0.015, and the final test track distance ratio exhibits moderate variability with a standard deviation of less than 0.05;
- the retraining process is smoother and much more monotonic than observed in the previous studies, with respect both to the training and test track performance.

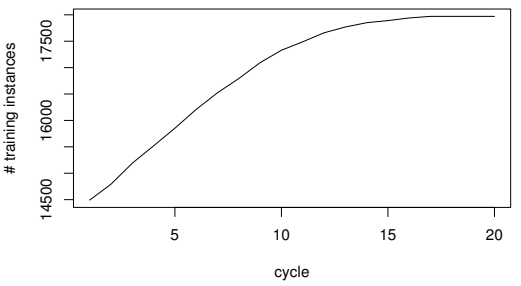

Fig. 15. Number of training instances used for Study 3.

Table 3. Individual test track performance for Study 3.

track	distratio	faicount
Aalborg	0.508	7.2
Alpine-1	0.903	0.0
Alpine-2	0.978	0.0
Brondehach	0.641	1.1
Corkscrew	0.871	0.9
E-track-1	0.696	1.9
E-track-2	0.679	2.1
E-track-3	0.746	1.2
E-track-4	0.767	0.8
E-track-6	0.663	1.8
Eroad	0.715	0.6
Forza	0.744	1.6
G-track-1	0.855	0.5
G-track-2	0.823	0.3
G-track-3	0.803	2.0
Ruudskogen	0.711	1.9
Spring	0.668	2.1
Street-1	0.747	1.5
Wheel-1	0.750	1.5
Wheel-2	0.720	2.0

While the average test track distance ratio still leaves much to be desired, it is clearly superior to that observed in the previous study. However, the number of encountered out-of-track failures is greater, which is definitely disappointing.

Individual test track performance is presented in Table 3, and the histograms in Fig. 16 illustrate the performance distribution over all test tracks. For each track the normalized distance and the number of failures are still averaged over multiple experiment runs. The following observations can be made:

- the normalized distance traversed reaches or exceeds 0.9 on two tracks, exceeds 0.8 on additional four

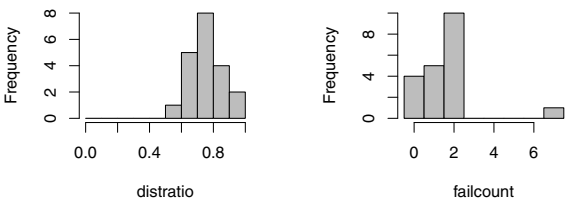

Fig. 16. Histograms of test track performance for Study 3.

tracks, and exceeds 0.7 on further eight tracks,

- the normalized distance traversed falls below 0.6 for a single track only (*Aalborg*),
- there are four test tracks on which the learned control model makes 0 or nearly 0 failures on the average and five tracks with the average number of failures about 1,
- on all but a single test track (*Aalborg*) the number of failures does not exceed 3.

The major observed advantage of random forest models over single tree models is the greatly reduced number of retraining instances used as well as the improved stability and monotonicity of the learning process. Unfortunately, this does not translate into considerably improved test track performance. While the normalized distance traversed is greater than for single trees, out-of-track failures are more frequent.

4.3. Discussion. It is noteworthy that the *Inferno* bot used as the exemplary driver in our experiments belongs to the most refined TORCS bots with an aggressive driving style and supreme driving performance. Approaching the same performance level by control models learned by imitation is therefore a significant achievement. In several preliminary experiments we found it much easier to imitate simpler TORCS bots, exhibiting more conservative driving style and worse performance. For such bots it was possible to create imitation models driving simple tracks without retraining, but their performance is far below the level reported here.

The control model's inability to outperform the exemplary driver may appear disappointing, but is actually to be expected given the learning scenario. In particular, if some track segments can be traversed considerably faster than demonstrated by the exemplary driver (which is actually highly unlikely given the quality of the *Inferno* bot), this possibly remains unnoticed by the control model trained based solely on the examples of its performance. No exploration mechanisms exist in our current approach that would make it possible to detect such improvement possibilities. The state representation used by the control model, including a subset of state attributes available in TORCS, and the discretized control actions, preventing it from making some fine-tuned direction or speed corrections, may also limit its potential performance. Still, delivering performance comparable to that of the exemplary behavior on a variety of task instantiations (different tracks it this case) can be considered a success of imitation learning that should be by no means confused with simple memorization of observations and actions.

The non-monotonic changes of performance in consecutive retraining cycles, with improvements sometimes alternating with degradations, may be caused by the inherent instability of decision tree growing. Even relatively minor dataset variations (such as adding a small number of additional training instances) are likely to yield substantially different models, which may avoid the failures encountered for the previous models, but lead to different (and possibly more frequent) failures. Instability is likely to be the major problem with applying decision tree models to control tasks. As mentioned before, it is excessive instability that prevented successful application of decision tree pruning. It also makes the whole modeling procedure overly sensitive to some minor experimental settings, such as the size of the initial training set or the number of retraining instances generated on control failures.

The random forest algorithm generally stood up to expectations with better generalization and greater stability. It made the retraining process smooth, with a monotonic performance improvement, finally yielding near-perfect training track performance despite the high complexity of the training track. This was possible using several times less retraining instances than for single tree models. The test track performance was also improved, according to the normalized distance measure, but out-of-track failures occurred more frequently. It looks that random forest models maintain higher driving speed and therefore encounter more difficult situations, some of which ultimately led to falling out of track. To understand how this is possible, recall that retraining is triggered not only on out-of-track situations, but also on excessive slow-downs. Single tree models and random forest models appear to have reached different levels of tradeoff between avoiding falling out of track and excessive speed loss. Controlling this tradeoff explicitly and dynamically during model application may be necessary to successfully combine high driving speed where possible with slowing down for security where necessary.

The higher than expected out-of-track failure frequency observed for random forests may be also explained by the smaller number of retraining cycles and,

more importantly, the much smaller number of retraining instances used than for single tree models. This shows that they were much more successful on the training track, encountering failures and triggering retraining instance generation sparingly. This exceedingly good training performance prevented random forest models from being demonstrated with more failure avoidance examples and may have had negative impact on their test track performance. One potential remedy could be deliberately disturbing control actions in the training phase, to engender more failures and more frequently trigger retraining instance generation. This would make the retraining process more exploratory.

The performance observed when using multiple test tracks, even with random forest models, may at first appear disappointing. Despite using a relatively complex training track that seems to represent most bend patterns occurring in the test tracks, the learned control model fails to successfully drive on a substantial number of them. Performance failures on previously unseen tracks may be hardly avoidable, though, without using some considerably enriched state representation or a built-in track hardness exploration mechanism. It is also not unlikely that a more refined training track, providing a better representation of the complexity of tracks which the control model is expected to handle, would yield better performance. Another self-suggesting method of getting a control model capable of driving on a variety of tracks would be to use a greater number of diverse tracks in the training or retraining process.

5. Conclusion

What has been successfully attempted by this work and is its unique novel contribution is the application of decision trees and random forests to the representation of control models learned by imitation. Being much more common in data mining applications, decision trees are usually not considered for control tasks, although they have some quite important advantages over more common subsymbolic methods, such as neural networks. They are fast to learn and apply, and they are human readable. The former makes it possible to learn from large training sets, create multiple models, and retrain if necessary. The latter makes it possible for a human expert to inspect and verify the models, explain their decisions, and even combine them with some pre-existing domain knowledge for even better performance. Random forests sacrifice these advantages, but offer better overall predictive power, and in particular greater stability and overfitting resistance. Therefore the demonstration of the suitability of decision trees and random forests for control applications, using the realistically complex simulated car control task, can be argued to be a noteworthy contribution.

Our computational experiments include both a basic two-track setup, with a single training track and a single test track of similar complexity, and a multiple-track setup, with a single more complex training track and multiple diverse test tracks. In both cases we evaluate the performance of the learned control models *not only* on tracks on which the training data was generated, *but also* on previously unseen tracks. This is why the experimental results reveal both the strengths and limitations of the proposed approach.

On less demanding tracks the created decision tree control models achieve nearly the same performance levels as the exemplary driver, while using a relatively small subset of available input attributes and human-readable tree structures. These results provide a strong encouragement to consider a similar procedure for developing control algorithms for real-world tasks, as well as video game bots. With increased track complexity and diversity, it becomes clearly more difficult to properly generalize and transfer the driving skill acquired on one task to other tasks. This results in poor performance on several previously unseen tracks even if trained on an adequately complex training track. Our attempts to reduce overfitting with cost-complexity pruning failed due to the increased instability, resulting from its internal cross-validation used to estimate the expected error of pruned trees.

We have demonstrated that random forest models overcome the limitations of single decision tree models to a substantial extent. They achieve better generalization capabilities and improve stability as well as reduce the requirements for the amount of training data, although they encounter out-of-track failures somewhat more frequently. The latter may be a consequence of their much better training performance, which results in much less retraining instances being generated and used, and could be possibly alleviated by introducing some exploration to the retraining process. Another disadvantage associated with random forest models is the loss of human-readability that constitutes an important part of the motivation behind using decision trees for imitation learning in the first place. Getting more stability without sacrificing comprehensibility is probably the major challenge in using symbolic learning algorithms for control tasks. It may be worthwhile to investigate the utility of existing approaches to getting comprehensible models out of model ensembles (Park and Kargupta, 2002; Van Assche and Blockeel, 2007; Triviño Rodriguez *et al.*, 2008).

The presented results demonstrated the capabilities not only of the employed classification algorithms, but also, and more importantly, of the modeling procedure contributed by this article, including the retraining technique, automatically triggered on control failures. It was found to be absolutely essential for the performance

level possible to obtain. While it remains to be verified by future work, we believe the procedure is reusable and likely to yield good control models across a variety of tasks where imitation learning can be applied, including other instances of vehicle control, involving both simulated and real vehicles.

The actual quality of the learned driving skill in the TORCS environment is reasonably good, but definitely leaves some considerable space for improvement. What is missing in the current approach, in particular, is the ability to explore the hardness of the track and the capabilities of the car, making it possible to adapt the control policy accordingly. With a larger range of permitted control actions (e.g., a greater choice of turn radiuses and acceleration levels) and possibly differential rather than plain action discretization, this might enable the control model learned by imitation to perform more aggressively and eventually outperform the exemplary driver.

One possible algorithmic framework that can be used to study the idea of track exploration and control policy adjustment that is particularly appealing is provided by reinforcement learning (Sutton and Barto, 1998; Kaelbling *et al.*, 1996). Combining imitation learning to obtain a reasonably good initial control policy and reinforcement learning to refine it or adapt to new conditions is probably the most promising, but also most challenging future work direction. When pursuing this path, it may be necessary to employ reinforcement learning speedup techniques (e.g., Cichosz, 1995; Zajdel, 2013). Other prospective future research topics include eliminating some simplifications adopted for our modeling procedure to get us started. It may be worthwhile to reconsider directly handling multidimensional control actions rather than decomposing them into independent one-dimensional actions, and using the original numerical action representation on input and output rather than discretization interval representatives. It would be particularly interesting to investigate some possible refinements of the retraining procedure, e.g., triggering retraining instance generation not only on failures, but also on other situations believed to provide useful information (e.g., divergence from a pre-calculated target trajectory).

The TORCS simulator used for the presented experiments, while considered quite realistic physically, is not representative for many potential vehicle control applications due to its idealized racing track environment, which has little in common with, e.g., city traffic or off-road terrain exploration. Addressing the challenges related to such tasks is definitely another important avenue for further research. On the other hand, since TORCS is also an enjoyable video game, this article can be viewed as a proposal of a possibly promising approach to creating game bots, serving as interesting and demanding opponents for human players.

References

Anderson, C.W., Draper, B.A. and Peterson, D.A. (2000). Behavioral cloning of student pilots with modular neural networks, *Proceedings of the 17th International Conference on Machine Learning (ML-2000), Stanford, CA, USA,* pp. 25–32.

Atkeson, C.G. and Schaal, S. (1997). Robot learning from demonstration, *Proceedings of the 14th International Conference on Machine Learning (ML-97), Nashville, TN, USA,* pp. 12–20.

Baluja, S. (1996). Evolution of an artificial neural network based autonomous land vehicle controller, *IEEE Transactions on Systems, Man and Cybernetics* **26**(3): 450–463.

Bratko, I., Urbancic, T. and Sammut, C. (1998). Behavioural cloning of control skill, *in* R.S. Michalski, I. Bratko and M. Kubat (Eds.), *Machine Learning and Data Mining*, John Wiley & Sons, Chichester.

Breiman, L. (1996). Bagging predictors, *Machine Learning* **24**(2): 123–240.

Breiman, L. (2001). Random forests, *Machine Learning* **45**(1): 5–32.

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J. (1984). *Classification and Regression Trees*, Chapman and Hall, New York, NY.

Buehler, M., Iagnemma, K. and Singh, S. (Eds.) (2007). *The 2005 DARPA Grand Challenge: The Great Robot Race*, Springer, Berlin.

Buehler, M., Iagnemma, K. and Singh, S. (Eds.) (2009). *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, Springer, Berlin.

Cardamone, L., Loiacono, D. and Lanzi, P. (2009a). On-line neuroevolution applied to The Open Racing Car Simulator, *Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC-09), Trondheim, Norway*, pp. 2622–2629.

Cardamone, L., Loiacono, D. and Lanzi, P. (2010). Learning to drive in The Open Racing Car Simulator using online neuroevolution, *IEEE Transactions on Computational Intelligence and AI in Games* **2**(3): 176–190.

Cardamone, L., Loiacono, D. and Lanzi, P.L. (2009b). Learning drivers for TORCS through imitation using supervised methods, *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG-09), Milano, Italy,* pp. 148–155.

Chambers, R.A. and Michie, D. (1969). Man-machine co-operation on a learning task, *in* R. Parslow, R. Prowse and R. Elliott-Green (Eds.), *Computer Graphics: Techniques and Applications*, Plenum, London, pp. 179–186.

Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning, *Journal of Artificial Intelligence Research* **2**: 287–318.

Cichosz, P. (2007). *Learning Systems*, 2nd Edn., WNT, Warsaw, (in Polish).

D'Este, C., O'Sullivan, M. and Hannah, N. (2003). Behavioural cloning and robot control, *Proceedings of the International Conference on Robotics and Applications, Salzburg, Austria*, pp. 179–182.

Dietterich, T.G. (2000). Ensemble methods in machine learning, *Proceedings of the 1st International Workshop on Multiple Classifier Systems, Cagliari, Italy*, pp. 1–15.

Esposito, F., Malerba, D. and Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(5): 476–491.

Forbes, J.R.N. (2002). *Reinforcement Learning for Autonomous Vehicles*, Ph.D. thesis, University of California at Berkeley, Berkeley, CA.

Guizzo, E. (2011). How Google's self-driving car works, *IEEE Spectrum*, `http://spectrum.ieee.org`.

Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*, 2nd Edn., Morgan Kaufmann, San Francisco, CA.

Hertz, J., Krogh, A. and Palmer, R.G. (1991). *Introduction to the Theory of Neural Computation*, Addison-Wesley, Boston, MA.

John, G.H. (1996). Robust linear discriminant trees, *in* D. Fisher and H. Lenz (Eds.), *Learning from Data: Artificial Intelligence and Statistics V*, Springer, New York, NY, pp. 375–385.

Kaelbling, L.P., Littman, M.L. and Moore, A.W. (1996). Reinforcement learning: A survey, *Journal of Artificial Intelligence Research* **4**: 237–285.

Kohl, N., Stanley, K., Miikkulainen, R., Samples, M. and Sherony, R. (2006). Evolving a real-world vehicle warning system, *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO-06), Seattle, WA, USA,* pp. 1681–1688.

Krödel, M. and Kuhnert, K.-D. (2002). Reinforcement learning to drive a car by pattern matching, *Proceedings of the 24th DAGM Symposium on Pattern Recognition, Zurich, Switzerland,* pp. 322–329.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M. and Thrun, S. (2011). Towards fully autonomous driving: Systems and algorithms, *Proceedings of the IEEE Intelligent Vehicles Symposium (IV-11), Baden-Baden, Germany*, pp. 163–168.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomForest, *R News* **2/3**: 18–22.

Loiacano, D., Cardamone, L. and Lanzi, P.L. (2009). Simulated car racing championship 2009: Competition software manual, *Technical report*, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano.

Loiacono, D., Prete, A., Lanzi, P. L. and Cardamone, L. (2010). Learning to overtake in TORCS using simple reinforcement learning, *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC-2010), Barcelona, Spain*, pp. 1–8.

Mitchell, T. (1997). *Machine Learning*, McGraw Hill, New York, NY.

Munoz, J., Gutierrez, G. and Sanchis, A. (2009). Controller for TORCS created by imitation, *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games (CIG-09), Milano, Italy*, pp. 271–278.

Park, B.-H. and Kargupta, H. (2002). Constructing simpler decision trees from ensemble models using Fourier analysis, *Proceedings of the 7th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, Madison, WI, USA*, pp. 18–23.

Pomerleau, D. (1988). ALVINN: An autonomous land vehicle in a neural network, *Advances in Neural Information Processing Systems 1 (NIPS-88), Denver, CO, USA*, pp. 305–313.

Quinlan, J.R. (1986). Induction of decision trees, *Machine Learning* **1**(1): 81–106.

Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA.

Quinlan, J.R. (1999). Simplifying decision trees, *International Journal of Human-Computer Studies* **51**(2): 497–491.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, `www.R-project.org`.

Sammut, C. (1996). Automatic construction of reactive control systems using symbolic machine learning, *Knowledge Engineering Review* **11**(1): 27–42.

Sammut, C., Hurst, S., Kedzier, D. and Michie, D. (1992). Learning to fly, *Proceedings of the 9th International Conference on Machine Learning (ML-92), Aberdeen, UK,* pp. 385–393.

Stavens, D.M. (2011). *Learning to Drive: Perception for Autonomous Cars*, Ph.D. thesis, Stanford University, Stanford, CA.

Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

Therneau, T.M. and Atkinson, E.J. (1997). An introduction to recursive partitioning using the RPART routines, *Technical report*, Mayo Clinic, Rochester, MN.

Thrun, S. (2010). What we're driving at, Google Official Blog, `http://googleblog.blogspot.com/2010/10/what-were-driving-at.html`.

Togelius, J., De Nardi, R. and Lucas, S.M. (2006). Making racing fun through player modeling and track evolution, *Proceedings of the SAB-06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games, Rome, Italy*, pp. 61–70.

Triviño Rodriguez, J.L., Ruiz-Sepúlveda, A. and Morales-Bueno, R. (2008). How an ensemble method can compute a comprehensible model, *Proceedings of the 10th International Conference Data Warehousing and Knowledge Discovery (DaWaK-08), Turin, Italy*, pp. 368–378.

Urbancic, T. and Bratko, I. (1994). Reconstructing human skill with machine learning, *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94), Amsterdam, The Netherlands*, pp. 498–502.

Utgoff, P. E. (1989). Incremental induction of decision trees, *Machine Learning* **4**(2): 161–186.

Van Assche, A. and Blockeel, H. (2007). Seeing the forest through the trees: Learning a comprehensible model from an ensemble, *Proceedings of the 18th European Conference on Machine Learning (ECML-07), Warsaw, Poland*, pp. 418–429.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edn., Morgan Kaufmann, San Francisco, CA.

Wymann, B. (2006). TORCS manual installation and robot tutorial, `http://www.berniw.org/aboutme/publications/torcs.pdf`.

Zajdel, R. (2013). Epoch-incremental reinforcement learning algorithms, *International Journal of Applied Mathematics and Computer Science* **23**(3): 623–635, DOI: 10.2478/amcs-2013-0047.

Paweł Cichosz received his M.Sc. and Ph.D. degrees in computer science from the Warsaw University of Technology in 1994 and 1998, respectively. He is an assistant professor at the Institute of Electronic Systems of the same university. His areas of research interests include machine learning, data mining, and artificial intelligence.

Łukasz Pawełczak received his B.Eng. and M.Sc. degrees in electronics and computer engineering from the Warsaw University of Technology in 2011 and 2013, respectively. He is interested in applications of machine learning algorithms. His participation in this research was part of his diploma programmes.

Received: 31 January 2013
Revised: 16 October 2013
Re-revised: 21 January 2014