Learning to Steer on Winding Tracks Using Semi-Parametric Control Policies

Ken Alton and Michiel van de Panne
Department of Computer Science
University of British Columbia
Vancouver, BC, V6T 1Z4, Canada
{kalton,van}@cs.ubc.ca

***Abstract*— We present a semi-parametric control policy representation and use it to solve a series of nonholonomic control problems with input state spaces of up to 7 dimensions. A nearest-neighbor control policy is represented by a set of nodes that induce a Voronoi partitioning of the input space. The Voronoi cells then define local control actions. Direct policy search is applied to optimize the node locations and actions. The selective addition of nodes allows for progressive refinement of the control representation. We demonstrate this approach on the challenging problem of learning to steer cars and trucks-with-trailers around winding tracks with sharp corners. We consider the steering of both forwards and backwards-moving vehicles with only local sensory information. The steering behaviors for these nonholonomic systems are shown to generalize well to tracks not seen in training.**

***Index Terms*— nonholonomic systems, reinforcement learning, policy search, hybrid control, vehicle steering**

I. Introduction

Learning control policies for dynamic systems that operate in continuous state and action spaces is a challenging problem. Methods that work for systems with two to four-dimensional state spaces often do not scale well to systems of higher dimension. Delayed rewards also make it difficult to appropriately assign credit or blame to a sequence of control actions.

Reinforcement learning (RL) and direct policy search methods have been applied to continuous control problems with some success. Producing a workable solution generally requires defining an appropriate set of input state variables, choosing an appropriate control policy representation, and shaping the problem with a suitable reward function.

In this context, the principle contributions of our paper are as follows. First, we introduce a compact control policy representation that is semi-parametric, meaning that the representation grows with and adapts to the complexity of the problem being solved. Second, we demonstrate the effectiveness of this representation and our policy search algorithm in solving a series of challenging nonholonomic control problems.

Reinforcement learning applied to continuous state spaces continues to be an active area of research. A common approach is to manually discretize the input state space using grids [1]. This approach, however, scales poorly to higher-dimensional problems and it is difficult to choose the granularity of the grid without having intimate knowledge about the problem. Adaptive discretizations have been introduced more recently [2], [3], although it is not clear that the grid-based axis-aligned structures employed by these methods can be made to work effectively for non-holonomic control problems that have hard constraints on the allowable directions of movement in their state space.

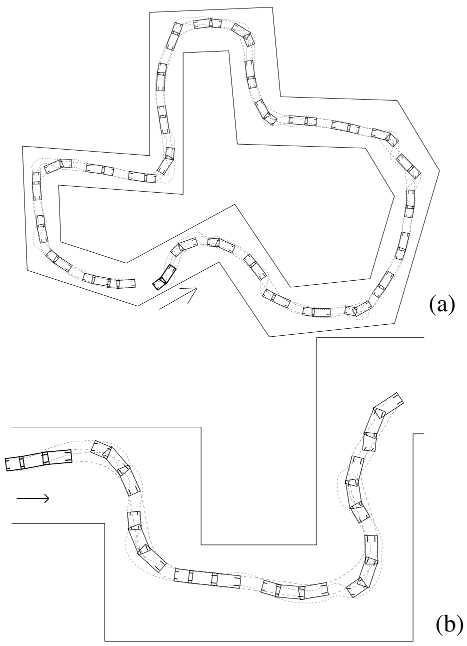

Fig. 1. (a) Trajectory of a backwards-driving one-trailer truck; (b) Backing up a two-trailer truck.

Recently, RL methods have been proposed that compute value functions and control policies directly in the continuous domain without requiring discretization of the state space [4], [5]. These techniques are promising, although open questions remain regarding scalability, convergence, and their successful application to nonholonomic control problems.

A novel trajectory-based nonparametric approach is presented in [6] and is applied to the control of a two-link planar robot. The trajectory-based representation for value functions and policies is useful for overcoming interference in learning and the "curse of dimensionality".

Nonholonomic systems, such as the truck-and-trailer configurations with which we experiment, have been the focus of much research over the past 15 years. One common class of approach is to use a trajectory-planning stage followed by the use of a trajectory-tracking scheme [7]–[9]. Trajectory-planning requires a world model, which is something we wish to avoid in our reactive, sensor-based approach to steering.

A second class of approach involves learning a *control policy*, or simply *policy*[1], which maps input states to actions. Such an approach is used to solve variations of the "truck backer-upper" problem presented in [10]–[13]. In general, the goal is to backup a one or two-trailer truck such that the end of the last trailer approaches a goal position, which represents a loading dock. Most versions of this problem assume knowledge of the global state of the truck [10], [11], or at least the angle with respect to the goal state [12], [13]. In contrast, our steering problem is defined in terms of steering performance around a sharply-curved track rather than that of reaching a single well-defined goal state. Furthermore, we require our controller to work with only local sensory information instead of global state information.

The problem of car steering has been examined in the context of RARS, the Robot Auto Racing Simulator [5], [14]. Optimal actions are learned based upon information about the car's position and velocity relative to local track features. In [5], the car's curvilinear distance from the start line of the track is also available to the policy. Both of these methods assume knowledge of the local track curvature and learned policies are only tested on the same tracks that were used for training. We develop solutions based on simpler sensory information provided by a set of distance sensors. Also, we demonstrate that a learned steering policy is capable of generalizing to new tracks that are similar variations of the training track. Moreover, we apply our steering method to significantly more challenging models, such as backwards-moving tractor-trailers.

Two examples of the type of steering problem we solve are illustrated in Figure 1. This shows a single and double-trailer truck being controlled to drive backwards on previously unseen tracks. In general, the vehicles are equipped with four distance sensors with which to observe the local track. The goal is to drive the vehicle, quickly and without collisions, either backwards or forwards (depending on the problem instance) around a track chosen from a specified class.

The remainder of this paper is organized as follows. In Section II, we define our policy representation. We describe our policy search method in Section III, including a description of the adaptive refinement that is a key feature supported by our policy representation. Section IV demonstrates the application of this representation and our policy search method to several nonholonomic vehicle steering problems. We give a detailed description of the problem and present results. Section V provides conclusions and future work.

[1]We consider the terms *controller*. *control policy*, and *policy* to be synonyms, but we mostly use *policy* in this paper.

II. Nearest-Neighbor Control Policy

The design of a policy begins with a choice of the representation to be used. The requirements of the control representation for our steering problem are as follows. The policy should allow for compact descriptions of the required control actions as a function of state. Particularly, it should scale well to working with continuous state spaces of medium dimension (i.e., 4 to 7 dimensions). The policy should allow for the addition of local detail in the representation where this is required or in order to support coarse-to-fine policy learning.

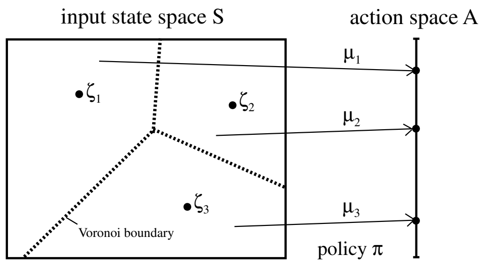

Fig. 2. An example nearest-neighbor policy of 3 nodes with a 2-dimensional input state space and 1-dimensional action space.

Our nearest-neighbour policy representation consists of a set of nodes, Ω, that are used to model the policy, $\pi : S \rightarrow A$, where S and A are the input state space[2] and action space, respectively. Each node i has a location, $\zeta_i \in S$, and an output, $\mu_i \in A$, as illustrated by the example policy in figure 2. A specific policy is defined by Θ, a vector of all the policy parameters, $[\zeta_1, \ldots, \zeta_n, \mu_1, \ldots, \mu_n]$, where $n = |\Omega|$ is the number of nodes. The value of ζ_i locates node i in the continuous state space S, inducing a tesselation of S into Voronoi cells with the use of a nearest-neighbor metric. The policy is then defined as $\pi(\mathbf{x}) = \mu_{c(\mathbf{x})}$, where $c(\mathbf{x})$ is determined by the Voronoi cell containing the system state, $\mathbf{x}$, i.e., $c(\mathbf{x}) = argmin_i(||\mathbf{x} - \zeta_i||)$.

III. Policy Evaluation and Policy Search

The representation of the task to be accomplished is an important choice for designing a policy. The *policy evaluation function*, or simply *evaluation function*, $E(\Theta)$, measures how well a given policy accomplishes the desired task. In specifying the evaluation function, there should be an ability to establish an appropriate compromise between the size of the domain over which the policy is capable of

[2]We use the term *input state* to refer to the aspects of system state which are input to the policy. This is synonymous with *observable state*.

operating successfully and the performance of the policy. Thus, a steering policy that yields higher performance in terms of its ability to drive quickly around a winding track may have a smaller domain of states from which it can successfully recover without crashing. Our specific choice of $E(\Theta)$ for vehicle steering is discussed in Section IV-B.

We optimize our policy using direct policy search. The goal is to find the best policy given a particular evaluation function, $E(\Theta)$:

$$\Theta^* = argmax_\Theta E(\Theta),$$

where Θ^* is the optimal policy parameter vector. The success of policy search techniques is largely dependent on the existence of meaningful and well-defined local optima in the search space. The choice of policy representation and of the evaluation function, $E(\Theta)$, can both significantly affect the structure of the local optima. We use the term *shaping* to refer to all factors that impact upon the shape of the optimization landscape.

Shaping using progressive refinement of a policy can be achieved by treating the policy search as a series of optimization problems, $O_1, \ldots, O_m$, where the policy parameters resulting from O_i are used to bootstrap the the next episode of optimization, O_{i+1}. With our policy representation, this is supported through the incremental allocation of new nodes to a policy with each optimization episode, as shown in Algorithm 1, our policy search routine. Thus, degrees-of-freedom are progressively added to the policy as it becomes more refined. This yields a semi-parametric approach as opposed to the parametric approach that would result from using a fixed number of nodes[3].

Algorithm 1 Policy Search with Adaptive Refinement

```
input $\pi$ {policy}
$E_{best} \leftarrow E(\pi.\Theta)$
repeat
    $\pi_{new} \leftarrow add\_node(\pi)$
    $\pi_{new} \leftarrow optimize(\pi_{new})$
    $E_{new} \leftarrow E(\pi_{new}.\Theta)$
    if $E_{new} > E_{best}$ then
        $\pi \leftarrow \pi_{new}$
        $E_{best} \leftarrow E_{new}$
    end if
until convergence
output $\pi$
```

It is important to add a node at a location in the input space where refinement may be needed, as well as to initialize its action to a sensible value. In the *add_node* routine of Algorithm 1, the location, ζ_{new}, of a newly added node is drawn uniformly from the distribution of observations encountered during the evaluation of the previous policy. This simple heuristic biases the refinement to regions of the input state space S that are encountered frequently. The location of the first-added node is sampled from the observations of a policy that always steers straight. The action, μ_{new}, is initialized to be the same as that of the closest node, or in other words the action that would be taken by the previous policy at the state ζ_{new}. This bootstraps the action to be a reasonable value, which is then further refined by optimization. The action of the first-added node is sampled from a uniform distribution on the range [-1.0,1.0] radians.

Within an optimization episode, i.e., in the *optimize* routine, we employ a stochastic greedy ascent algorithm to search for a local maxima, Θ^*. Our current implementation applies a stochastic perturbation $\delta\Theta$ to Θ, assigning $\Theta \leftarrow \Theta + \delta\Theta$ when $E(\Theta + \delta\Theta) > E(\Theta)$. This optimization routine is decribed further in [15].

IV. Application to Vehicle Steering

We apply our policy representation to the problem of steering cars, single-trailer trucks, and double-trailer trucks both forwards and backwards on serpentine tracks, as shown in Figures 1, 5, 6, 7, and 8[4]. Our goal is to develop general driving behaviors that can drive well on new, unseen tracks of the same class as the training track.

A. Problem Description

We consider nonholonomic systems in the form of cars, single-trailer trucks, and double-trailer trucks. The control input to the system is the steering angle of the front wheels of the truck, α, as shown in Figure 3. In our simulation, α is controlled indirectly by using a PD controller, $\ddot{\alpha} = k_p(\alpha - \hat{\alpha}) - k_d\dot{\alpha}$, where k_p and k_d are the spring and damper constants[5], respectively, and $\hat{\alpha}$ is the desired steering angle. This acts as a low-pass filter that prevents unrealistic instantaneous changes in the steering direction. The policy action provides the desired steering angle, $\hat{\alpha}$. We assume a constant driving speed in our model.

The track is perceived using 4 distance sensors, d_1–d_4, as illustrated in Figure 3, each of which has a 22.5° angular sweep. Each sensor returns the distance to the closest object, i.e., point on the track edge, seen within its angular range. The sensors serve the dual purposes of locating the vehicle on the track as well as providing information about upcoming turns on the track. The distance sensors are located in the center of the vehicle segment that is leading the motion, i.e., in the last trailer for a backward-driving truck or the cab for a forward-driving truck. The sensors face in the direction of motion. These four measurements are the only representation of the environment that is available,

[3]Note that a semi-parametric policy representation can be distinguished from a nonparametric representation, which typically stores experience data directly rather than consolidating it.

[4]Animations of the resulting motions are available at www.cs.ubc.ca/~kalton/icra.html.

[5]The values of these constants were chosen to be $k_p = 10.0$ and $k_s = 3.5$ for most steering problems and they remained fixed during the policy search.

and are combined with the current steering angle, α, and the current hitch angles (e.g., β for the one-trailer truck) in order to produce a combined system-and-environment state descriptor. Currently, we do not consider error in the sensor values.

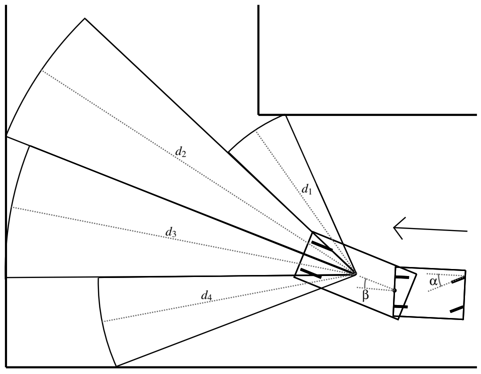

Fig. 3. Truck sensory information.

As an example, the single-trailer truck shown in Figure 3 operates in a 6-dimensional continuous input state space. An input state to the policy is given by $[\alpha, \beta, d_1, d_2, d_3, d_4]$. Each policy node i thus has 7 associated parameters: one for the control action, μ_i, which is the target steering angle, $\hat{\alpha}$, and the 6 parameters for the location of the node in the input state space, $\zeta_i \in S$.

We enforce left-right symmetry on our control policy by introducing a symmetric copy of each node. This allows any progress made in learning left turns to immediately be applied to right turns, and vice-versa.

The tracks are produced using a randomized track-generator. This allows us to generate new test tracks that are similar, but not identical to tracks used for training. A family of tracks is specified according to their width and inter-turn distances. In our experiments, all training tracks have 90 degree turns[6], which have equal probability of turning left or right. The length of the straight track segments placed between turns is drawn uniformly from $[d_{min}, d_{max}]$, where d_{min} and d_{max} are given as part of the track family specification. The details of how a track is generated to be a complete circuit without self-intersections are beyond the scope of this paper.

B. Policy Evaluation Function

Our general goal is to create policies that steer the vehicles around the tracks in minimal time without crashing and without jack-knifing[7]. To this end, we compute a policy evaluation function that measures distance travelled along the track center-line during a fixed simulation duration, T_{eval}. Figure 4 shows how this distance between an initial state s and an final state s' is computed.

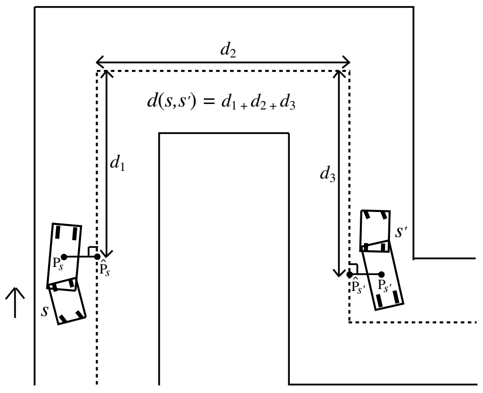

Fig. 4. Distance measured along the center of the track.

T_{eval} defines the time horizon of the evaluation, which determines its short-sightedness. We choose T_{eval} to be of sufficient duration for the vehicle to navigate around at least two corners. This discourages the policy from making bad short-sighted choices. We do not use a discounting scheme, as is often done in reinforcement learning.

An evaluation function that measures the policy performance from a single initial state is inadequate in that the solutions may become overly tuned to the situations encountered from that initial state. To overcome this, we evaluate the policy from multiple samples of an initial state distribution, D. We then construct a more comprehensive evaluation function that sums the evaluations from each sampled initial state:

$$E(\Theta) = \sum_{i=1}^{N} d(s_i, s_i') + P(s_i'),$$

where s_i is the i^{th} initial state, s_i' is the i^{th} final state, and $P(s_i')$ is crash penalty function that is k_{crash} if a collision or jack-knifing occurs and is otherwise zero[8]. The initial states s_i are sampled once and then held constant throughout the optimization process in order to fix the evaluation function for convergence. A more formal analysis of this type of sample-based reward function can be found in [16].

The specific choice of the initial state distribution can strongly influence the end result. It can be used to manipulate the tradeoff between providing control over a large domain versus providing higher-performance control over a smaller domain. In the former case, a learned policy may be able to guide not-often-seen but difficult initial states towards successful track navigation. In the latter case, a higher-performance solution could be achieved, one that perhaps does faster, more refined turns around corners at the expense of having to start the vehicle in a familiar situation, i.e., located near to the track center.

[6]However, testing has been applied on more general tracks with varying turn angles as shown in Figure 1(a).

[7]Trucks are considered to be jack-knifed when a trailer hitch angle exceeds 90 degrees.

[8]In our experiments, we use a value of $k_{crash} = -10$ for the crash penalty. For reference, T_{eval} is typically between 5 and 10 s and the fixed vehicle speed is 1 unit/s.

C. Experimental Results

1) Forwards Driving: The task of forwards-driving around the track is simpler than the equivalent backwards-driving task because of the stability provided by having the steered wheels lead the motion. Because the goal is to travel around the track in minimal time, we would expect to see behaviors such as steering close to inside corners and steering directly (diagonally) between corners. A critical issue to be addressed is that of avoiding collisions with corners, which is especially interesting in the case of the trucks-with-trailers given that the trailers do not directly follow the path of the cab. For this reason, the cab must swing wide in a turn in order to provide sufficient corner clearance for the trailer. Steering close enough to the corner so that near-maximum progress around the track is made, while not approaching so close that a collision is risked is a compromise that is defined by the evaluation function and learned during the policy search process.

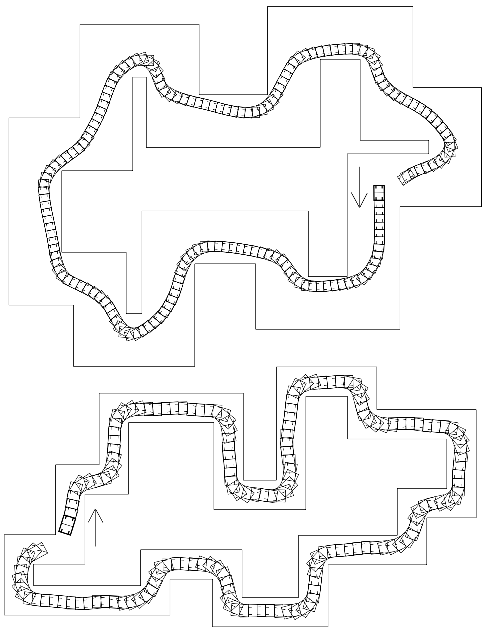
Fig. 5. Car driving forwards on wide and narrow tracks.

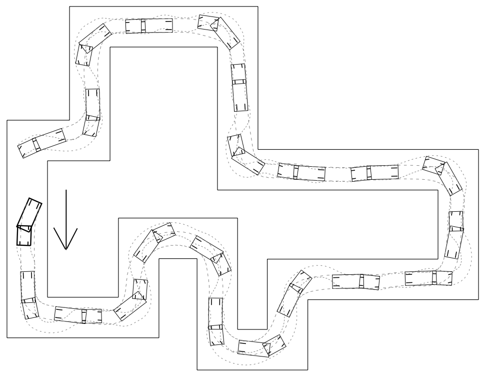
Fig. 6. Truck with single-trailer driving forwards.

In Figure 5, we see the resulting steering behavior of policies developed for forward car-driving on wide and narrow tracks. For the wide track, the path of the car cuts to the inside on corners to progress more quickly around the track. This policy uses 15 nodes and is computed using 1342 function evaluations (i.e., computations of $E(\Theta)$). For the narrow track, the turns must be executed with precision in order to avoid collisions with the sides of the track. This policy uses 21 nodes and required 3158 function evaluations.

Finally, Figure 6 shows a forwards-driving single-trailer truck. Note the necessity of the truck cab to *swing wide* on turns in order to avoid the trailer clipping the corner. The policy has 20 nodes and required 2801 function evaluations.

All the results shown are for tracks that are novel, but that belong to the same family of tracks used during training. To test for robustness with respect to modeling errors, we made alterations to the wheel-base of the car of up to 20% and found that the policy still steered well.

2) Backwards Driving: Backwards-driving around winding tracks is a task of significant difficulty. This is particularily true of trucks-with-trailers, as the policy needs to cope with steering an unstable system in addition to following the winding track. We note that this problem is significantly different from the "truck backer-upper" problem in that there is no useful global representation of the state and that the problem cannot be represented in terms of a goal state or goal trajectory.

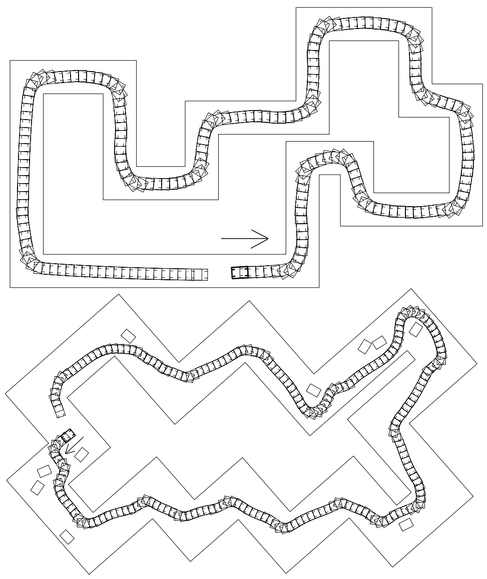
Fig. 7. Car driving backwards.

Figures 1, 7, and 8 show steering of policies produced for the car, one-trailer, and two-trailer trucks. The policy search required 3665 function evaluations for the car, 13476 for the single-trailer truck, and 15482 for the double-trailer truck. The resulting policies use 22, 14, and 30 nodes, respectively.

The backwards-driving car policy shown in Figure 7 is particularly robust to changes in the track width, turn angles, and obstacles on the track. This policy was trained on a

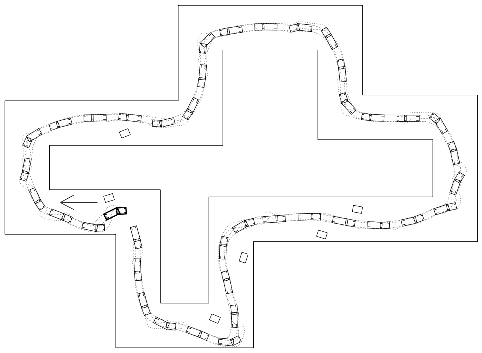

Fig. 8. Single-trailer truck driving backwards.

track of the same class as that illustrated in Figure 7(top). However, the policy was able to generalize well to steering on tracks with obstacles and a modified width, such as that shown in the Figure 7(bottom). This car policy can also successfully navigate a "Y" or "T" branch in the track.

As Figure 1(a) shows, the one-trailer truck policy is reasonably robust to changes in track width and various turn angles. Also, the truck policy is capable of avoiding objects on the track, as shown in Figure 8, although it was never trained for these conditions. This policy was trained on an obstacle-free track with a fixed width and 90 degree turns. In order to test robustness with regard to errors in the truck model, we applied the computed policy to one-trailer trucks with the position of the rear trailer wheels varied. The truck policy still steers well when this wheel position is changed by as much as 10% measured from the middle of the trailer.

Figure 9 shows the improvement of the policy evaluation, $E(\Theta)$, as the policy search for the single-trailer truck progresses. Each solid line in the figure corresponds to a single optimization episode after a new node has been added (see Algorithm 1).

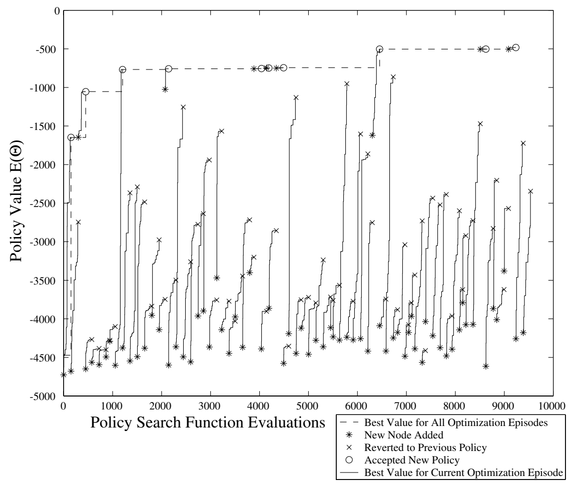

Fig. 9. Policy performance versus function evaluations during the policy search for a single-trailer truck driving backwards on a winding track.

V. Conclusions

We have described a semi-parametric control policy for learning the challenging task of efficiently steering zero, one, and two-trailer trucks forwards and backwards along families of winding tracks. We show that the resulting behaviors generalize to similar but novel tracks. The control policies have a highly compact representation that can be adaptively refined to suit the complexity of the control problem.

As future work, we wish to explore the use of smooth interpolation kernels to allow for better representation of continuously-valued actions. We also wish to consider how to develop steering policies that are usable on an even wider variety of tracks by finding an appropriate parameterization that reflects the current class of track. Lastly, we wish to develop solutions that also consider velocity control and skidding behaviors.

References

[1] A. Barto, R. Sutton, and C. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 834–846, 1983.

[2] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Machine Learning*, vol. 49, Numbers 2/3, pp. 291–323, November/December 2002.

[3] A. Moore and C. Atkeson, "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," *Machine Learning*, vol. 21, pp. 1–36, 1995.

[4] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, pp. 243–269, 2000.

[5] R. Coulom, "Reinforcement learning using neural networks, with applications to motor control," Ph.D. dissertation, 2002.

[6] C. G. Atkeson and J. Morimoto, "Nonparametric representation of policies and value functions: A trajectory-based approach," in *NIPS 15*, 2003, pp. 1611–1618.

[7] A. Divelbiss and J. T. Wen, "A path space approach to nonholonomic motion planning in the presence of obstacles," *IEEE Transaction on Robotics and Automation*, vol. 13, no. 3, pp. 443–451, June 1997.

[8] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *The International Journal of Robotics Research*, vol. 22, no. 7, pp. 583–601, July 2003.

[9] C. Altafini, A. Speranzon, and B. Wahlberg, "A feedback control scheme for reversing a truck and trailer vehicle," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 915–922, Dec. 2001.

[10] D. H. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural networks," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN89)*, 1989, pp. 357–363.

[11] J. R. Koza, "A genetic approach to finding a controller to back up a tractor-trailer truck," in *Proceedings of the American Control Conference*, 1992.

[12] D. F. Hougen, J. Fischer, M. Gini, and J. Slagle, "Fast connectionist learning for trailer backing using a real robot," in *IEEE International Conference on Robotics and Automation*, 1996, pp. 1917–1922.

[13] D. F. Hougen, M. Gini, and J. Slagle, "Rapid, unsupervised connectionist learning for backing a robot with two trailers," in *IEEE International Conference on Robotics and Automation*, 1997.

[14] L. D. Pyeatt and A. E. Howe, "Learning to race: Experiments with a simulated race car," in *11th Intl. Florida Artificial Intelligence Research Society Conference*, 1998.

[15] K. Alton, "Shaping and policy search for nearest-neighbour control policies with applications to vehicle steering," Master's thesis, University of British Columbia, 2004.

[16] A. Y. Ng and M. Jordan, "PEGASUS:A policy search method for large MDPs and POMDPs," in *Proceedings of UAI 2000*, pp. 406–415. [Online]. Available: citeseer.ist.psu.edu/ng00pegasus.html