Learning an Optimal Race Driver Policy using TORCS

I. Introduction

According to [1] Americans drive approximately 3 billion miles per year. One way to increase the productivity of our society as a whole is to increase the speed with which people reach their intended destinations. Having an autonomous vehicle capable of high speed travel opens up new possibilities for fast travel by car.

Control of a car at high speeds is a hard problem which has sparked competitions for creating the fastest autonomous driver in an open source race car simulation called *The Open Racing Car Simulator (TORCS)*[8]. Many autonomous drivers exist, however most of them use knowledge of the whole track and other parameters that are not available to a driver in a real world. Furthermore, near optimal controllers for robotics applications have been learned using reinforcement learning in the past [5]. This project will show that it is feasible to learn a driving policy that allows a race car to travel quickly, but that the state representation for the learner needs to be carefully designed.

II. Related work

The control issues of autonomous high speed driving have been studied in [3]. In this paper, the optimal path around a track was determined by hand, and a control scheme autonomously commanded an Audi TTS to follow the optimal path. Using this approach, an autonomous vehicle is able to come within 5-10% of the performance of an expert human. The drawback of this approach is that it cannot be applied to a different course without re-determining the optimal path by hand. To improve this controller, the optimal path should be automatically generated. The problem of computationally determining an optimal path in TORCS was studied in [4]. In this paper, the authors assume that the best path around a track is a compromise between the shortest path and the path of minimum curvature (cutting the corners). Genetic algorithms are used to search for the best such compromise in each section of a track. One drawback of this method is that exact geometry of the entire track must be known. In a real driving situation, the algorithm would not have accurate information about the entire driving course at any given point on the course; local information would be more accurate than global information. A local search can be performed using policy search techniques as described in [5] and combined with the techniques of [3].

III. Approach

A. Reinforcement Learning for Autonomous Driving

In this project we would like to learn the controls of the car given the current state of the car and the track. This can be achieved by reinforcement learning which learns which actions to perform in a given state solely based on a reward signal. However, classic approaches are based on approximating dynamic programming and require learning a value function which in turn requires samples that are distributed over the entire state-action space. For multidimensional, continuous state-action spaces, obtaining the required number of samples can be difficult. Policy search is a class of RL algorithms which attempts to remedy this issue by using parametric policies and learning only the parameters of that model; this requires a significantly lower number of samples. The parametric low-level policy that we are using for this project is a deterministic car controller that is described in section III-B. The parameters of this model are cubic splines defined by 6 equidistant points. Furthermore, we want the car to drive differently based on the immediate future of the track. Since the low-level policy does not take that information into account, we need to learn a parametric high-level policy that outputs different splines based on a high-level state (context) which consists of the position of the car and curvature of the track at 6 equidistant points in from of the car. The form of this high-level parametric policy is a linear Gaussian model, i.e. a Gaussian distribution whose mean is a linear combination of some features of the state. One algorithm that can learn both policies at the same time is contextual episodic relative entropy policy search.[6][7]

In [6], Peters et al. introduce Relative Entropy Policy Search (REPS), in [7], Kupcsik et al. adapt it to the episodic case with a high-level state. REPS finds the parameters of the high-level distribution by maximizing the expected reward with respect to the policy and the distribution over contexts.

$$\max_{\pi,\mu^\pi} J(\pi) = \sum_{\mathbf{s},\mathbf{a}} \mu^\pi(\mathbf{s})\pi(\mathbf{a}|\mathbf{s})r(\mathbf{s},\mathbf{a}) \tag{1}$$

Where π denotes the high-level policy, μ^π the distribution over contexts, $\mathbf{s}$ the context, $\mathbf{a}$ the spline and $r(\mathbf{s},\mathbf{a})$ the reward of executing the spline in a given context. For the reward function, we choose the time that the car takes to drive part of the spline $\mathbf{a}$ (planning the spline a little further allows it to be smoother). Since the context-distribution is given by the track, it cannot be freely chosen by the optimization. The distribution is only known at sample-points, therefore the distribution has to be constrained on the samples. However, as a more efficient approximation, only the averages of features of the context are matched. For the features used for matching and the linear Gaussian model that forms the high-level policy, we use second order polynomials. This means that the constraint matches mean and variance of the context distribution.

Furthermore, as the optimization problem can only be solved on samples, the optimization is only good when the resulting policy is creating samples in the vicinity of the samples that have already been observed. As covering the whole state-action space is not data-efficient, Peters et al. constrain the KL-divergence between the distribution that generated the samples and the solution. The optimization process can then be iterated to find the optimal solution while always staying close to the observed data.

This optimization problem can be solved with the method of Lagrangian multipliers and it can be shown that the closest sample-based parametric distribution is equal to the weighted maximum likelihood of the parametric (linear Gaussian) model where the weights of each sample are a function of the obtained reward, the context and the Lagrangian parameters that are obtained by numerically optimizing the dual function induced by the constrained optimization problem.

B. *Car controller*

The low level controller is responsible for executing the path plan generated by the policy search. This plan is generated as a set of distances away from the centerline of the track and interpolated using cubic splines. The spline path is then fed into a controller using techniques from the BT TORCS robot (provided with the TORCS source code) and [2]. The controller is divided into two main parts, a steering controller and a velocity controller.

The velocity controller uses both a feed-forward and a feedback component. The feed-forward component looks at upcoming corners on the track to determine the maximum speed allowable at any point. To do this, a traction circle is used to derive total acceleration/deceleration constraints at any point on the track, then equations for motion are integrated from a look-ahead point on the path back to the current vehicle position. This gives a constraint for the maximum velocity at the current position. A feedback controller then uses this constraint to attempt to either accelerate or decelerate based on current vehicle velocity. The feed forward controller takes the following form, derived in [2]:

$$\frac{dU_x(s)}{ds} = \frac{1}{U_x(s)}\sqrt{(\mu g)^2 - (\kappa U_x(s)^2)^2} \tag{2}$$

where κ is the instantaneous path curvature, $\mathbf{U_x(s)}$ is the vehicle velocity along the path at position $\mathbf{s}$, and μ is the total (tire times road) coefficient of friction.

The steering uses a simple "follow-the-carrot" steering controller. A "carrot" is placed on the path in front of the robot a distance determined by

$$d_{look-ahead} = K_{look-ahead} + K_{speed} * U_x \tag{3}$$

where $\mathbf{K_{look-ahead}}$ and $\mathbf{K_{speed}}$ are tuning constants. The steering angle is then determined by the angle between the vector from the car's position to the look-ahead point and the car's heading.

C. *Integration with TORCS*

We have implemented this approach for the TORCS simulator. At 50Hz, TORCS polls our implementation for control inputs to the car while providing complete information about the position of the car on the track and the shape of the complete track (of which we only use a segment in front of the car as this is what would be available on a real autonomous car). Using this information, we divide the track into 300m long segments and detect whenever the car has entered a new segment. Once this happens we sample a new spline from the high-level policy and store it. We then utilize the low-level controller with this spline to output the car control signal.

We tested the efficiency of the learning by evaluating the lap race time for a Policy Search-based AI driver and compared the result with the lap race time of a car that drives using only the TORCS standard controller described in [2]. The policy search-based AI driver was allowed to learn over several thousand simulated laps. Tests were performed in the TORCS simulator on the simple and complicated tracks shown in Figure 2.

IV. Evaluation

Figures 2a and 2b show that the lap times of the policy search agent decreases as the number of laps increases. The initial random spread of times also improves, showing that a consistent solution is being reached. Each lap time of the AI agent is also compared to a baseline where the low-level controller is given policy directly down the centerline of the track. This policy is known to be sub-optimal, as the best policy is to attempt to cut the track corners. Once the policy search agent's lap time converges, it achieves a 2.7s faster lap time than the baseline on the oval track. Figure 2b shows the same comparison as Figure 2a except that the standard deviation parameter of the policy search method is increased from 2 to 7. This increase in standard deviation causes the wider spread of unconverged data points observed at the beginning of training. However, even with this greater initial deviation, the learner converges to the same final performance. Figure1 shows the same learning performed on the complicated track. The result here is less conclusive because the lap times of the

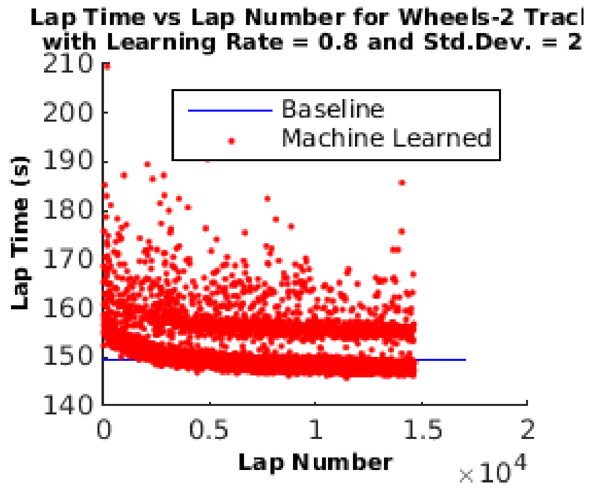

Figure 1: Lap Time of Policy Search AI Agent vs baseline on complicated "Wheel 2" track.

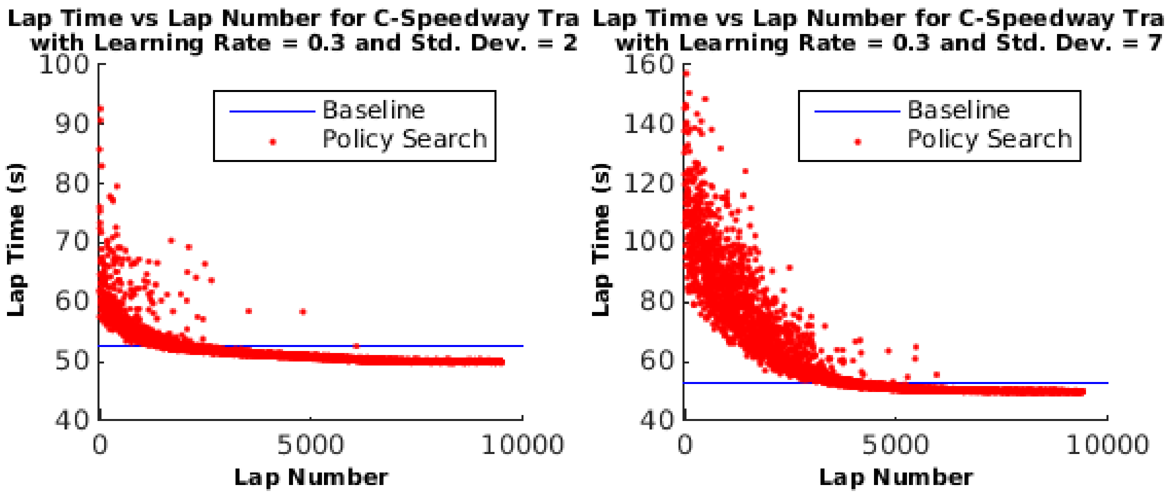

Figure 2: **a)** Lap Time of Policy Search AI Agent vs baseline. **b)** Lap Time of Policy Search AI Agent vs baseline on simple "C-Speedway" track. Standard deviation increased.

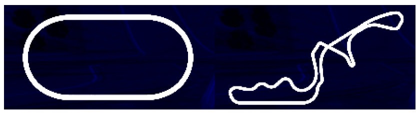

Figure 3: Comparison of the "C-Speedway" track and the "Wheel 2" track

policy search agent converge to a bi-modal distribution, with part of the distribution being slightly faster than the baseline.

V. Discussion

Results from Figures 2a and 2b are promising; the policy search agent outperforms the baseline after convergence. However, as can be seen in Figure 1, the policy on the more complex track improves only marginally. The core difference between more complicated tracks such as wheel-2 and the C-speedway track is the larger variety in contexts over which that the high-level policy has to generalize. We can therefore conclude that while learning parameters of the controller is generally possible, generalizing over different contexts requires either a different representation of the context or different features as the learned high-level policy is generating splines as a linear combination of the features of the context. A different representation such as the direction of the track (first derivative) instead of the curvature (second derivative) might be closer to a linear relation with the optimal policy. Different features, such as RBF features[9] on the other hand would allow us to learn more complex high-level policies, however, this comes at a large hit on the computational performance of the learning algorithm.

In Figure 1, the learner converges to a policy with multi-modal results. This is likely because the policy search trades a safe strategy for a risky one that is faster in some cases but causes crashes in other cases. This could be fixed by adding vehicle damage as part of the reward function. This result also points to our input state not accurately representing the state of the world. The learner requires an output which can be represented as a linear combination of the input state, so the input state would benefit from a more compact description.

Future work includes further investigating the multimodal distribution of the policy search results for the complicated track to ascertain its cause. Next, we will experiment with different state representations of the car to see how they affect lap times and convergence speed. We will also improve the low level controller by removing the limitation uncovered by the learner. Finally, we will make a more robust curvature calculation method.

Acknowledgment

Thanks to Gerhard Neuman (http://www.ausy.tu-darmstadt.de/Team/GerhardNeumann) and his colleagues for the REPS MATLAB implementation

References

[1] National Highway Traffic Safety Administration. (2013, November 13) *2012 Fatality Analysis Reporting System (FARS) Encyclopedia,* [Online]. Available: http://www-fars.nhtsa.dot.gov/Main/index.aspx
[2] K. Kritayakirana and J. C. Gerdes, "Autonomous vehicle control at the limits of handling" *Int. J. Vehicle Autonomous Systems*
[3] J. Funke et al., "Up to the Limits: Autonomous Audi TTS" *IEEE Intelligent Vehicles Symposium*, June 2012, pp. 541 - 547.
[4] L. Cardamone et al., "Searching for the Optimal Racing Line Using Genetic Algorithms" 2010 IEEE Conference on Computational Intelligence and Games (CIG10)
[5] Marc Peter Deisenroth. A Survey on Policy Search for Robotics. In: Foundations and Trends in Robotics 2.1-2 (2011), pp. 1142.
[6] Peters, J., Altun, Y. (2010). Relative Entropy Policy Search. In Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence (AAAI), Physically Grounded AI Track.
[7] Kupcsik, A., Deisenroth, M., Peters, J., & Neumann, G. (2013). Data-Efficient Generalization of Robot Skills with Contextual Policy Search. In Proceedings of the National Conference on Artificial Intelligence (AAAI).
[8] B. Wymann, E. Espi, C. Guionneau, C. Dimitrakakis, R. Coulom, A. Sumner. TORCS: The Open Racing Car Simulator, v1.3.6, 2014
[9] Sutton, R., and Barto, A. 1998. Reinforcement Learning. MIT Press.