Learning from Simulation, Racing in Reality

Eugenio Chisari[1], Alexander Liniger[2], Alisa Rupenyan[1], Luc Van Gool[2,3], John Lygeros[1]

***Abstract*— We present a reinforcement learning-based solution to autonomously race on a miniature race car platform. We show that a policy that is trained purely in simulation using a relatively simple vehicle model, including model randomization, can be successfully transferred to the real robotic setup. We achieve this by using novel policy output regularization approach and a lifted action space which enables smooth actions but still aggressive race car driving. We show that this regularized policy does outperform the Soft Actor Critic (SAC) baseline method, both in simulation and on the real car, but it is still outperformed by a Model Predictive Controller (MPC) state of the art method. The refinement of the policy with three hours of real-world interaction data allows the reinforcement learning policy to achieve lap times similar to the MPC controller while reducing track constraint violations by 50%.**

I. Introduction

Autonomous racing is a quickly growing sub-field of autonomous driving where the goal is to drive around a race track as fast as possible. The control policy must be able to perform strategic planning, so that the car is always at the right place on the track with the correct speed. At the same time, the policy has to control the vehicle at the limit of handling, where the behavior of the model is highly nonlinear. Currently, the most advanced methods use model-based predictive control techniques [1], [2], [3], [4]. Several groups showed that the performance of such methods can be increased by the use of data to improve the prediction model [5], [6]. Direct learning of a policy that can control an autonomous race car is not much explored, an exception being [7], where a pixel-to-action policy is trained through imitation learning on a 1:5 scale car and [8] where a policy is learned in a race car simulation game. Compared to model-based approaches, Reinforcement Learning (RL) does not require an accurate vehicle model or a well tuned reward function. Instead, the RL agent learns to race by interacting with the environment using high level rewards. This has been demonstrated on real robots for several robotics applications [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], in areas like robot manipulation, quadcopters, and wheeled robots. When a model of the system is available, the RL agent can be trained in simulation with the purpose to transfer the learned policy on a robotic system [19] using the *Sim-to-Real* approach. However, the RL agent learns to interact with a limited representation of the environment and is prone to over-fitting with respect to the simulation. Tools such as domain or model randomization [19], [10], [20] often drastically help the sim-to-real transfer, by randomizing the observation and the model of the simulator. In [21], [22], [23] it was shown that sim-to-real works for autonomous driving, however, their main challenge was to close the vision domain gap, where in our autonomous racing case the domain gap in the vehicle model is the main challenge.

Regularization is an essential ingredient guiding the learning process, shaping the corresponding policy, and incorporating domain knowledge in the RL problem. In RL, the entropy maximization regularizer [24] is commonly used to encourage exploration. Recently, the effect of other regularization strategies popular in deep learning has been studied in [25]. They showed that using regularizers such as weight clipping, dropout, or batchnorm can be beneficial, but had the best results when using a L_2 regularization on the policy weights, even outperforming the classical entropy regularizer.

This paper shows how we can train an autonomous racing policy for a miniature racing car using sim-to-real. Directly using an RL method to learn this task is challenging because the system has continuous states and actions, requires a refresh rate of 100 Hz, has highly uncertain dynamics, and finally, the boundaries of the race track are walls, making exploration difficult. We show that a policy learned with an off-the-shelf actor critic method can be generalized to the real system, even if the agent is only trained in simulation with a simple vehicle model. We achieve this by using a well-suited state and action space in combination with model randomization, and by incorporating a novel policy output regularization technique, motivated by control theory. This regularizer, combined with input-lifting techniques where the actions are the changes in the physical inputs, significantly improves the sim-to-real transfer. We compare the proposed regularizer with established techniques such as reward shaping or L_2, and show that our method performs better in simulation and on the real robotic platform. Finally, we show that the policy learned in simulation can be refined by interacting with the real environment, achieving a performance comparable to that of a state of the art predictive controller in terms of lap time, while achieving 50% less constraint violations.

II. Autonomous Racing

We now introduce the experimental setup, and define the related Markov Decision Process (MDP). The experimental platform consists of miniature 1:43 remote controlled Kyosho dnano cars. The cars are controlled via Bluetooth by a remote PC and the position, heading and velocities are estimated by an overhead infra-red camera system at 100Hz.

[1] Automatic Control Lab, ETH Zurich, Switzerland
chisarie@student.ethz.ch,
{ralisa,lygeros}@control.ee.ethz.ch
[2] Computer Vision Lab, ETH Zurich, Switzerland
{alex.liniger,vangool}@vision.ee.ethz.ch
[3] PSI, KU Leuven, Belgium

The cars reach speeds of above 3 m/s, which correspond to an upscaled speed of 465 km/h. The cars drive around a 18 m long race track with 13 turns, see [26] for more details. Due to their small size and high relative speeds, the cars are challenging to control because their behavior is highly non-linear, making it difficult to model and predict the dynamics accurately. Previous research showed that considering this uncertainty can drastically help [27], [28].

A. State Space and Action Space

We are interested in finding an action space where smoothness can be enforced in the actions, and a state space where discontinuities in the (action) value function and in the policy can be avoided. We therefore represent the state in a curvilinear coordinate system [29] (Frenet frame), which transforms the position and heading relative to a reference path, in our case the center line of the race track. More specifically, we have that p is the progress along the track, n is the deviation from the reference path, and μ the heading relative to the path. Following standard vehicle dynamics, we also consider the longitudinal and lateral velocity in the vehicle's frame, v_x and v_y respectively, and the yaw rate ω as states. The physical control inputs to our car are the duty cycle to the rear wheel drive motor d, and the steering angle δ. Since jerky inputs could damage the actuators and harm the performance, we lift our inputs and add d and δ to the states and consider the rate of change of the physical inputs d_{rate} and δ_{rate} as our actions. In summary, our state is given by $s = [p, n, \mu, v_x, v_y, \omega, d, \delta]$, and the actions by $a = [d_{rate}, \delta_{rate}]$. This approach allows to efficiently penalize jerky inputs to our autonomous car (as we will discuss in Section III-C). We did not experience a noticeable slow down in learning because of this addition. Finally, the actions (input rates) as well as the physical inputs are both constrained. In the case of our miniature race cars we have the following bounds, $d \in [-0.2, 1.0], \delta \in [-0.35, 0.35], d_{rate} \in [-17.5, 17.5]$, and $\delta_{rate} \in [-3.5, 3.5]$. We implement the constraints on the input rates using a $\tanh$ output layer in our policy and the physical input constraints by clipping. Note that we assume that we can observe the state, which is reasonable given the overhead camera system, but even for for full-size race cars an autonomous driving stack similar to [30] could be used to achieve the same.

B. Reward Function

The goal of a race car is to drive around the track as quickly as possible, while not violating the track limits. In this work we propose to formulate this objective using a dense reward function that penalizes constraint violations. Therefore, let us first define the used constraints. The first is a track constraint which is triggered if the deviation to the center line is larger than half the track width $|n_{t+1}| \leq w_{\text{tr}}/2$. Note that in practice we added a safety margin to the actual track width, since in our experimental setup the boundaries are physical walls, and touching them either leads to an accident or drastically slows down the car. The second constraint we introduce is added to facilitate the sim-to-real transfer. When driving at the limit of handling, combined slip of the tires is important, but it is not modeled in our simulator. Thus, we include a constraint in our reward that penalizes excessive combined slip. We model this using the following tire ellipse constraint $F_{y,R}^2 + (p_{\text{long}} F_x)^2 \leq (p_{\text{ellipse}} D_{\text{R}})^2$, where $p_{\text{long}} = 0.9$ and $p_{\text{ellipse}} = 0.95$ are tyre specific parameters [30]. To model the lap time minimization objective, we use an incremental progress reward $p_{t+1} - p_t$, where the agent is rewarded to drive as far as possible with respect to the center line. This reward is popular in autonomous racing [1], and it is a good dense approximation of the sparse minimum lap time reward. Thus our reward function is defined as

$$r_t = \begin{cases} -c & \text{if constraints violated} \\ p_{t+1} - p_t & \text{otherwise} \end{cases},$$

where $c = 0.01$ and $p_{t+1} - p_t$ is in the range between 0 and 0.05. The proposed reward penalizes deviations from the track and driving in regions of the state space where our simulator is not precise, while it rewards fast driving.

III. Sim-to-Real

In the following section we explain the necessary steps to perform the sim-to-real transfer for our autonomous racing task and discuss both simulation and experimental results. We also introduce a novel policy regularization approach to facilitate the sim-to-real transfer.

A. RL Setup

We formulate our problem as an episodic RL problem, where an episode is either terminated if the car violates the track constraints for more than double the track width or if it lasts for 600 time steps (equivalent to 6 s of driving). To improve exploration of the state space we start each episode at a random initial state and use relative short episodes, following [31]. We use Soft Actor Critic (SAC) [32] due to its superior performance compared to other state of the art RL methods on our setup, and due to its off-policy capabilities which we will exploit in Section IV. Our implementation of SAC is based on Stable Baselines [33], and we implement our simulator, explained in Section III-B, as an OpenAI Gym environment [34]. The control policy and (action) value functions, are implemented as fully connected neural networks with two hidden layers and 256 neurons per layer. We use a learning rate of $3 \cdot 10^{-4}$, a batch size of 512, a replay buffer of length 10^6, and a discount factor of 0.99. We run SAC at every time step of the simulation and train for $1.8 \cdot 10^7$ time steps. We run the training on a single desktop computer, where we only use the CPU (Intel i9-9900K), which takes about 36 hours to complete.

B. Simulator and Model Randomization

Our simulator is based on the dynamic bicycle model proposed in [1], which uses Pacejka tire models [35] to model the interaction with the ground. This model is simpler than full vehicle dynamic models, which makes it computationally

more tractable. However, it is generally over-optimistic and does not capture the uncertainty of the real cars.

Therefore, we use model randomization to train a policy that can transfer to the real system, by reducing the reality gap. We propose to use a multiplicative uncertainty model of the form $\dot{s} = f(s,a)(1+\varepsilon)$, where $f(s,a)$ is the nominal dynamic bicycle model and ε the uncertainty. The advantage of a multiplicative uncertainty is that in the case of rapid changes in the system, which are difficult to model, the effect of the uncertainty is even increased. Since the position and orientation are only the integration of the velocities, we consider only uncertainty on the velocity states. We model this uncertainty as a uniform distribution, where the bounds are determined from real driving data recorded using a benchmark MPC. Given the state-action sequence the one-step model error can be computed using an Euler forward integrator. Based on these errors we chose the noise distribution to approximately include 80% of all recorded errors. When using uncertainty bounds that include all recorded errors or even overestimate the noise level, the resulting policies are over conservative. The resulting uniform distribution we use in training are $\varepsilon_{v_x} = \mathcal{U}(-1.5, 1.5)$, $\varepsilon_{v_y} = \mathcal{U}(-2.5, 2.5)$, and $\varepsilon_{\omega} = \mathcal{U}(-2.0, 2.0)$.

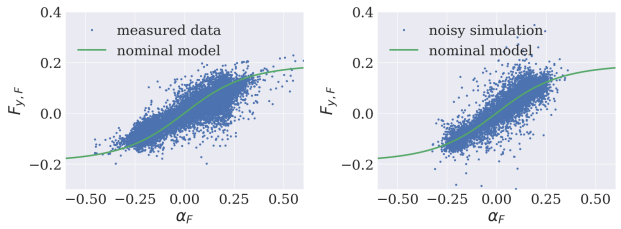

Fig. 1 Estimated tire forces vs Pacejka tyre model, real measurements left and noisy simulation right, both using our trained sim-to-real policy.

To qualitatively demonstrate the effect of our model randomization technique, we estimate the tire forces from state-action sequences, using a model inversion technique, and compare these force estimates obtained from experimental and simulation sequences. Figure 1 on the left shows the front wheel force estimates versus the slip angle when driving the real car as well as the nominal Pacejka tire model used in our simulator for reference. It is clearly visible that the real platform has a considerable degree of uncertainty, which makes model randomization a simple but necessary tool to bridge the sim-to-real gap. The force estimate of our model randomization approach is shown Figure 1 on the right. We see that the uncertainty we use in our simulator resembles the one appearing in the real recorded data.

C. Policy Regularization

There are two main motivations for regularizing action sequences. First, excessive and jerky inputs can potentially damage actuators, and in autonomous racing, the fast changing inputs can also introduce drag that degrades performance. The second reason is related to the transfer from simulation to reality. Excitation of a system with high-frequency inputs can result in behaviors that are difficult to model. In our experimental setup this is for example the load transfer, which is not considered in the simulation model. To mitigate these effects, it is common in model-based control approaches to consider an input cost that penalizes the control inputs. In RL-based approaches, however, such regularizers are less common, but have recently attracted some interest [36].

1) Reward and Policy Weight Regularization: Let us first introduce two standard approaches to regularize the inputs. First, we can augment the reward function, and add a quadratic cost on the applied actions, $r_t^{\text{rar}} = r_t - a^T \tilde{M} a$. The matrix $\tilde{M}$ is a hyper-parameter and we call this method *reward action regularization*. The second approach is to use standard regularizers from deep learning, the most popular one being the L_2 penalty on the policy weights, which we call from now on *policy weight regularization*.

Both approaches have some disadvantages. The *reward action regularization* approach results in slow learning, because the RL agent has to learn if a low reward is caused by non promising states or too excessive inputs. The *policy weight regularization* approach is tailored to reduce over-fitting to the training data but it is not effective at achieving our goal of smooth actions.

2) Policy Output Regularization: Given these insights we propose a novel regularization approach, *policy output regularization*, which directly regularizes the policy output without slowing down learning by using gradient information in the policy update step. More precisely, we try to find a policy that minimizes the following objective, where M is a positive semidefinite matrix and $\alpha H\left(\pi(s_t)\right)$ is the standard SAC entropy term,

$$\pi_M^*(s_0) = \underset{\pi}{\operatorname{argmax}} \; -\mathbb{E}[\pi(s_0)]^T M \mathbb{E}[\pi(s_0)] + \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(s_t)\right)\Big)\right].$$

Compared to SAC, we only add the first quadratic term that penalizes high values of the actions applied on the first time step. Through the choice of M, the regularization can be tuned independently for each action. We can derive the regularized action value function as

$$Q_M^\pi(s_0,a_0) = -a_0^T M a_0 + \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H\left(\pi(s_t)\right)\right] \quad (1)$$

$$Q_M^\pi(s_0,a_0) = -a_0^T M a_0 + Q^\pi(s_0,a_0), \quad (2)$$

where Q^π is the standard SAC action value function. Analogously to the standard SAC policy gradient update, the update of the regularized policy is

$$\nabla_\theta \frac{1}{|B|}\sum_{s_t\in B} -\left[Q_{M,\phi}\left(s_t,\pi_\theta(s_t)\right) + \alpha H\left(\pi_\theta(s_t)\right)\right]. \quad (3)$$

Equation (2) shows a tight relation between the standard action value function Q^π and the regularized version Q_M^π.

Given this relation, the policy loss (3) can be rewritten as

$$\nabla_\theta \frac{1}{|B|} \sum_{s_t \in B} -\left[Q_\phi\left(s_t, \pi_\theta(s_t)\right) + \alpha H\left(\pi_\theta(s_t)\right)\right] + \mathbb{E}[\pi_\theta(s_t)]^T M \mathbb{E}[\pi_\theta(s_t)]. \quad (4)$$

Therefore, to train a regularized policy, we can use the standard SAC algorithm steps, see Algorithm 1 in [32], to learn the Q^π function and add a simple quadratic term in the policy gradient step.

This regularizer can be used without input lifting. For our system and other robotic systems, however, the combination of considering input rates as actions and regularizing the policy output is preferable, since this approach does not penalize physical inputs but the rate at which they change. Therefore, a constant steering or the use of a high torque when holding a heavy object with a robot arm is not penalized, but a jerky steering or torque is. We study the discussed regularizers in more details on common RL benchmarks in Appendix A and demonstrate that the policy output regularizer has a positive effect both on the smoothness of the actions and on the reward, and outperforms the other regularizers in the more complex tasks.

TABLE I Comparison of performance and RMS action smoothness metrics for different policies

	SAC	Reward Reg.	Weights Reg.	Output Reg.	MPC
Lap Time $[s]$	10.0	11.3	9.8	9.8	8.7
Constraint Viol. $[s\times10^{-2}]$	3.6	15.6	9.0	1.7	8.6
$d_{\text{rms}}[10^{-1}]$	5.8	4.5	5.9	5.5	7.6
$\delta_{\text{rms}}[rad\times10^{-1}]$	2.0	2.0	2.0	2.0	2.3
$\dot{d}_{\text{rms}}[s^{-1}\times10^{0}]$	6.8	2.8	6.3	3.6	5.5
$\dot{\delta}_{\text{rms}}[\frac{rad}{s}]$	2.3	1.9	1.7	1.8	1.7
$\ddot{d}_{\text{rms}}[s^{-2}\times10^{1}]$	66.6	12.6	55.0	22.1	30.8
$\ddot{\delta}_{\text{rms}}[\frac{rad^2}{s^2}\times10^{1}]$	14.8	12.6	9.7	10.5	4.9

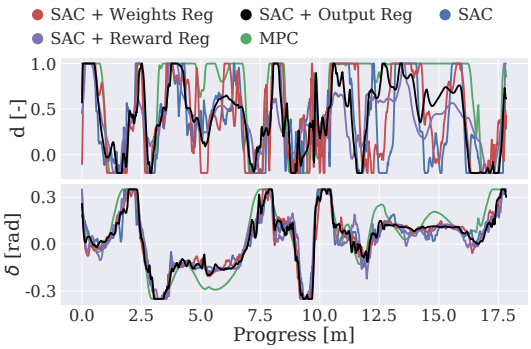

Fig. 2 Comparison of d and δ over one lap.

3) Regularization Comparison: To compare the different regularization methods we train four different policies, (i) vanilla SAC, (ii) SAC with reward action regularization (3) ($\tilde{M} = \text{diag}(0.005, 0.001)$), (iii) SAC with policy weight regularization (L_2 penalty 10^{-4}), and (iv) our proposed method SAC with policy output regularization ($M = \text{diag}(50, 10)$). Finally we compare the results with the state of the art MPC method [1]. All five methods are evaluated in the same simulation environment *with* model randomization present. In Table I we can see the results, where we show our two main driving objectives, lap time and constraint violation. The latter shows how many seconds the car is closer than 2 cm to the wall per lap. Since the goal of this study is also to evaluate the policy regularizers, we report the Root Mean Square (RMS) of the physical inputs (d, δ), and the RMS of the first $(\dot{d}, \dot{\delta})$ and second derivative $(\ddot{d}, \ddot{\delta})$. We can see that the MPC has the lowest lap time, but the second highest number of constraint violations. The four RL based methods are significantly slower, with method (ii) having the highest lap time, while also learning very slowly. The number of constraint violation of the four methods spans a large range, again with (ii) showing the worst performance. However, method (i) and especially our proposed method (iv) show significantly less constraint violations compared to the MPC controller. Method (iii) stands in between with the same lap time as (iv) but higher constraint violations. When comparing the smoothness metrics, we can mainly investigate the second order derivative, where we can see that vanilla SAC (i) is the worst performing, and the RL methods with additional regularization clearly generate smoother inputs. This is also shown in Figure 2, where SAC has clearly the most jittery actions. Method (ii) achieves a smooth duty cycle by driving slow, but has only a slightly smoother steering than (i). Method (iii) has a similar pattern as the MPC which is desired, but the smoothness scores are twice as high, which is problematic for the sim-to-real transfer. Finally, method (iv) has a smooth duty cycle, but aggressive steering. Even though this is a strategy different than the MPC, it allows for a successful sim-to-real transfer. We believe that the RL agent has learned that a fast, aggressive steering that corrects errors quickly, coupled with a relatively smooth duty cycle input works well. The input trajectories of our proposed method (iv) in Figure 2 look visibly better than the other RL based methods, especially the d action is smoother and has significantly less jitter. The steering input has a shape similar to that of the other RL agents, but again with reduced jitter.

D. Sim-to-Real Experimental Results

We now show that model randomization and the proposed *policy output regularization* facilitate sim-to-real transfer, by testing the trained polices discussed in the previous section on our experimental platform. The performance on the real platform of the policies trained in simulation is shown in Figure 3. We compare the mean lap time and time outside of the track constraints achieved by different learned policies, and by a benchmark MPC controller [37]. The benchmark MPC is a hierarchical method consisting of a recursive roadmap motion planner with a terminal viability constraint and a 0.32s look ahead, combined with nonlinear MPC trajectory tracker. Note that for both the MPC and the RL policy the time delays are compensated using a Kalman filter approach [26]. Lower lap times and constraint violations correspond to better performance. Data points with large lap time are an indication of collisions with the track walls. All policies in the figure are trained with model randomization.

Without model randomization (not shown), the policy is not able to generalize, and the car is not able to complete a lap. We also exclude the experimental results of the reward action regularization approach due to its bad performance. Using vanilla SAC[1] results in a 28% higher mean lap time, and 40% higher constraint violations, compared to the MPC benchmark policy. Policy weight regularization of the policy results in 24% slower lap times and 17% more constraint violations compared to the MPC, showing that policy regularization can help. The policy that performs best is the one trained with policy output regularization. It achieves a performance comparable to that of the MPC controller we use as benchmark, with a 30% improvement in constraint violations, and only 8% slower lap times on average. While the performance achieved with policy output regularization is not far from the performance of the MPC controller, it cannot match the peak performance of the MPC.

	Average Lap Time [s]	Average Time Out of Constraints [s]
SAC	13.53	2.15
SAC + Policy Weights Reg	13.18	1.79
SAC + Policy Output Reg	11.43	1.07
MPC	10.57	1.53

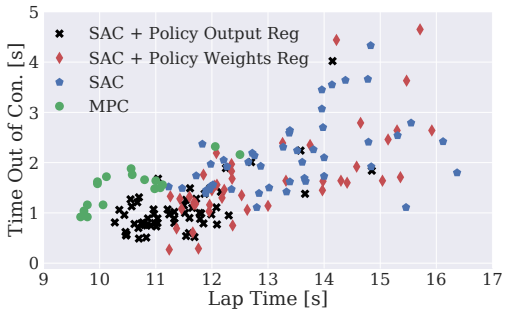

Fig. 3 Performance of different policies trained in simulation.

IV. Policy Refinement on the Car

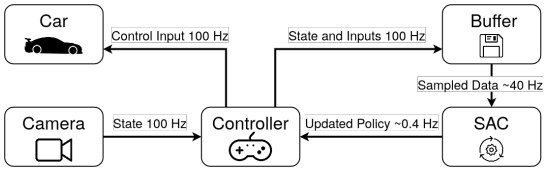

Fig. 4 Online reinforcement learning architecture

Given that we can transfer a policy learned in simulation to the real physical car, RL can now be used to refine this policy by using the real physical system, without the issues related to learning a policy from scratch. This allows to overcome the limitations of the simulator and train the system in a fully data-driven way. Therefore, we use an architecture as shown in Figure 4, where the current policy is controlling the car at 100Hz. At the same time we record the driving data and add it to a replay buffer which we use for off-policy updates using our *policy output regularized*-SAC method. We start with an empty replay buffer and collect data with our sim-to-real policy for roughly 70 laps (or one full battery charge). SAC, augmented with the policy output regularizer, is running as a secondary task on the same computer, updating its policy and (action) value function based on the replay buffer with 0.4 Hz. After 100 update steps of SAC, the controller policy is updated. Thus, we use the off-policy capability of SAC to have an approach that constantly updates the policy, while generating data that is close to on-policy which facilitates learning. Both the control loop and the SAC loop run on a laptop with a low power Intel i7-10510U CPU. On this hardware the SAC loop runs at roughly 40 Hz, and we update the policy about four times a lap.

A. Policy Refinement Experimental Results

We perform the experimental policy refinement as explained in Section IV, and show that this approach indeed can improve the autonomous racing policy, by updating the policy purely data-driven. More precisely, we interact with the real race car for three hours, which is only 6% of our simulation training. We warm-start with the sim-to-real policy that can already drive and is less damaging to the robotic platform than an initial random policy. Three hours of driving corresponds roughly to one million time steps, 450'000 SAC updates and 4'500 different polices.

We show the learning performance in Figure 5, where each data-point is the average of the last 12 minutes of driving, which is also the interval at which we change the car batteries. The first and the last data points in Figure 5 are for reference and show the initial sim-to-real policy and the final refined policy. In Figure 5 we can clearly see that the at the beginning of the learning phase the performance deteriorates, but the RL agent can recover and by the end of the three-hour interaction with the real car, the policy is drastically improved. Figure 6 shows that both the lap time and the constraints violation benefit from the policy refinement: the RL policy shines at achieving very regular lap times and low number of constraint violations. Quantitatively, the refined policy achieved an average lap time of 11.01 s, and violated the constraints for only 0.44 s per lap. When we compare these numbers to the sim-to-real results, we see that the policy refinement reduced the mean lap time by 0.42 s and the constraint violations by 0.63 s. This is a drastic improvement, bringing the refined policy close to the MPC method in terms of lap time, while drastically reducing the number of constraint violations by 71% compared the the state of the art MPC. It is especially interesting that the RL agent converged at a different type of solution than the hand-tuned MPC, emphasizing consistency and low constraint violations over raw peak lap times.

V. Conclusion

In this work we successfully applied the RL algorithm SAC to the autonomous racing problem. All results were validated experimentally on a miniature race track. We were able to achieve sim-to-real through model randomization and

[1]What we call SAC in our results can be considered "standard" sim-to-real with model randomization

	No Reg.	Output Reg.				Reward Reg.				Weight Reg.			
Reg. value		1	5	10	50	0.1	0.2	1	2	1e-7	1e-5	1e-4	0.001
Mountain Car Cont.													
avg. reward	70	100	100	100	-	100	-	-	-	100	100	100	100
avg. $\dot{a}_{\text{rms}}$	5.2	2.5	4.2	1.8	-	3.6	-	-	-	1.5	1.7	1.7	1.6
Pendulum													
avg. reward	-147	-164	-152	-166	-152	-148	-151	-146	-153	-155	-148	-147	-150
avg. $\dot{a}_{\text{rms}}$	31.6	11.8	9.7	15.4	7.5	10.7	10.7	10	6.7	13.6	11.2	11.0	9.9
PyBullet Reacher													
avg. reward	14.0	15.4	14.7	13.7	16.0	12.5	14.7	14.4	-	12.4	12.5	14.5	13.1
avg. $\dot{a}_{\text{rms}}$	70.0	31.8	14.7	6.4	1.6	71.6	37.1	3.2	-	83.2	76.1	51.3	76.5

TABLE II Benchmark performance for different regularizers.

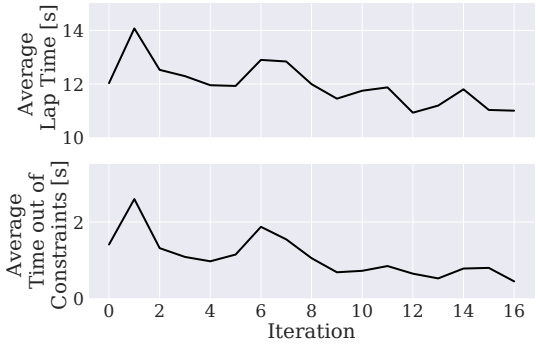

Fig. 5 Progress of the performance during policy refinement.

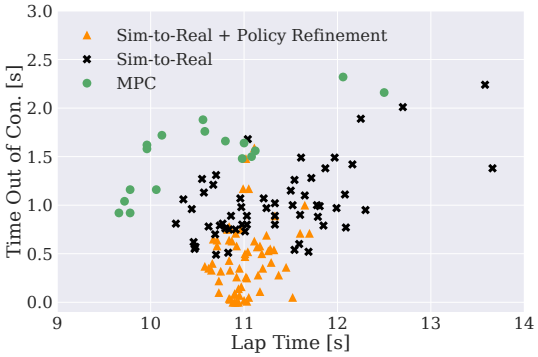

Fig. 6 Performance comparison of the policy refined on the car.

a policy output regularization strategy. Model randomization improves the generalization capability of the learned policy while policy output regularization helps smoothing the actions without affecting the convergence of the algorithm. We also show that it is possible to improve the performance of the policy trained in simulation by applying the same algorithm to the real car while it drives around the track. For this purpose an asynchronous strategy is adopted for storing and sampling driven trajectories to use for training. The achieved performance on the platform is comparable to that achieved by a state of the art model-based controller in terms of lap time, and shows an over three-fold improvement with respect to track constraints violations.

APPENDIX

A. Policy Regularization

To further highlight the importance of policy/action regularization we also studied the presented methods, *reward action*, *policy weight* and our proposed *policy output* regularization on three RL benchmark tasks. We tested two OpenAI gym problems [34], *MountainCarContinuous-v0* and *Pendulum-v0*, which are both simple RL problems. We followed the suggested RL-zoo [38] hyper-parameters and trained for $6 \cdot 10^4$ steps. The third benchmark is PyBullet Reacher [39], where we used the default SAC settings and trained the policy for $5 \cdot 10^5$ steps. For all three examples we tested vanilla SAC, and 4 different levels of regularization for the three tested regularizers, see Table II for the used regularization values[2]. After training, we evaluated 100 rollouts, with resulting average reward and mean RMS of the actions derivative reported in Table 3. Note that when the RL agent did not learn the task we report $-$ instead. The RMS of the actions derivative can be understood as a smoothness score, where lower values mean smoother actions. It is highly relevant for this work, as we showed in the race car application that smooth inputs facilitate the sim-to-real transfer. Note that compared to the race car setup we did not lift the actions, in order to leave the benchmarks unchanged.

In all three benchmarks, regularization helps not only to get smoother actions but also to improve the reward achieved by the policy. For the first two tasks *policy weight regularization* achieved the best results closely followed by our *policy output regularization*. Similar to our race car results *reward action regularization* often resulted in the RL agent not learning: there seems to be a very small range of hyper-parameters where this approach works, which makes it unpractical. In the last and hardest task, Reacher, we can clearly see that our propose *policy output regularization* achieved the best results. The main difference is that *policy output regularization* can achieve smooth actions without negatively impacting the learning. This stands in contrast to *reward action regularization* which is not designed to enforce smooth actions and failed in this task. Even though these results do not evaluate the effect on sim-to-real, we can see that our proposed regularizer can help in all tasks to achieve better policies which act in a smooth fashion.

[2]Diagonal matrices with these values when necessary.

REFERENCES

[1] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[2] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *American Control Conference (ACC)*, 2017.

[3] R. Verschueren, S. De Bruyne, M. Zanon, J. V. Frasch, and M. Diehl, "Towards time-optimal race car driving using nonlinear mpc in real-time," in *Conference on Decision and Control (CDC)*, 2014.

[4] C. E. Beal and J. C. Gerdes, "Model predictive control for vehicle stabilization at the limits of handling," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 4, pp. 1258–1269, 2012.

[5] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[6] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *International Conference on Robotics and Automation (ICRA)*, 2017.

[7] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, "Imitation learning for agile autonomous driving," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 286–302, 2020.

[8] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Duerr, "Super-human performance in gran turismo sport using deep reinforcement learning," *arXiv preprint arXiv:2008.07971*, 2020.

[9] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *International Conference on Robotics and Automation (ICRA)*, 2018.

[10] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[11] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2018.

[12] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[13] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin, "Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning," in *International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[14] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[15] H. Bou-Ammar, H. Voos, and W. Ertel, "Controller design for quadrotor uavs using reinforcement learning," in *International Conference on Control Applications*, 2010.

[16] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," in *International Conference on Robotics and Automation (ICRA)*, 2019.

[17] J. Baltes and Y. Lin, "Path tracking control of non-holonomic car-like robot with reinforcement learning," in *Robot Soccer World Cup*, 1999.

[18] D. Kamran, J. Zhu, and M. Lauer, "Learning path tracking for real car-like mobile robots from simulation," in *European Conference on Mobile Robots (ECMR)*, 2019.

[19] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[20] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 1–14, 2019.

[21] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall, "Learning to drive from simulation without real world labels," in *International Conference on Robotics and Automation (ICRA)*, 2019.

[22] B. Osiński, A. Jakubowski, P. Miłoś, P. Ziecina, C. Galias, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," *arXiv preprint arXiv:1911.12905*, 2019.

[23] A. Amini, I. Gilitschenski, J. Phillips, J. Moseyko, R. Banerjee, S. Karaman, and D. Rus, "Learning robust control policies for end-to-end autonomous driving from data-driven simulation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1143–1150, 2020.

[24] R. J. Williams and J. Peng, "Function optimization using connectionist reinforcement learning algorithms," *Connection Science*, vol. 3, no. 3, pp. 241–268, 1991.

[25] Z. Liu, X. Li, B. Kang, and T. Darrell, "Regularization matters in policy optimization," *arXiv preprint arXiv:1910.09191*, 2019.

[26] A. Liniger, *Path planning and control for autonomous racing*. ETH Zurich, 2018.

[27] J. V. Carrau, A. Liniger, X. Zhang, and J. Lygeros, "Efficient implementation of randomized mpc for miniature race cars," in *European Control Conference (ECC)*, 2016.

[28] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, 2019.

[29] J. L. Vázquez, M. Brühlmeier, A. Liniger, A. Rupenyan, and J. Lygeros, "Optimization-based hierarchical motion planning for autonomous racing," *arXiv preprint arXiv:2003.04882*, 2020.

[30] J. Kabzan, M. d. l. I. Valls, V. Reijgwart, H. F. C. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan *et al.*, "Amz driverless: The full autonomous racing system," *arXiv preprint arXiv:1905.05150*, 2019.

[31] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[32] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[33] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[34] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[35] H. B. Pacejka and E. Bakker, "The magic formula tyre model," *Vehicle system dynamics*, vol. 21, no. S1, pp. 1–18, 1992.

[36] R. Cheng, A. Verma, G. Orosz, S. Chaudhuri, Y. Yue, and J. W. Burdick, "Control Regularization for Reduced Variance Reinforcement Learning," *arXiv e-prints*, p. arXiv:1905.05380, May 2019.

[37] A. Liniger and J. Lygeros, "Real-time control for autonomous racing based on viability theory," *IEEE Transactions on Control Systems Technology*, vol. 27, no. 2, pp. 464–478, 2017.

[38] A. Raffin, "Rl baselines zoo," https://github.com/araffin/rl-baselines-zoo, 2018.

[39] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.