Learning from Simulation, Racing in Reality

Eugenio Chisari
IfA - ETH Zurich
chisarie@ethz.ch

Alexander Liniger
CVL - ETH Zurich
aliniger@ethz.ch

Alisa Rupenyan
IfA - ETH Zurich
ralisa@ethz.ch

Luc Van Gool
CVL - ETH Zurich
vangool@ethz.ch

John Lygeros
IfA - ETH Zurich
jlygeros@ethz.ch

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 simple vehicle model, including model randomization, can be successfully transferred to the real robotic setup. If a combination of our novel policy output regularization approach and a lifted action space is used, it enables smooth actions but still aggressive race car driving. Refining the policy with three hours of real-world interaction data allows the reinforcement learning policy to achieve lap times similar to a state of the art MPC controller while reducing track constraint violations by 50%. Videos of the experiments can be found at https://youtu.be/Z2A82AkT7GI.

1 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 and hard to predict. 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. 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 [8, 9, 10, 11, 12, 13, 14, 15, 16, 17], 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 [18] 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 [18, 9] often drastically help the sim-to-real transfer, by randomizing the observation and the model in the simulator. In [19, 20, 21] it was shown that sim-to-real works for autonomous driving, however, their main challenge was to close the vision domain gap, whereas 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 [22] is commonly used to encourage exploration. Recently, the effect of other regu-

NeurIPS 2020 3rd Robot Learning Workshop: Grounding Machine Learning Development in the Real World.

larization strategies popular in deep learning has been studied in [23]. This paper shows 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. Finally, a refinement in the policy through interaction with the real environment reaches performance comparable to that of a state of the art predictive controller in terms of lap time, while achieving 50% less constraint violations.

2 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 camera system at 100Hz. Due to their small size and high relative speeds (3m/s), the cars are challenging to control because their behavior is highly non-linear and stochastic, making it difficult to model and predict the dynamics accurately.

2.1 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 [24] (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. We did not experience a noticeable slow down in learning because of this lifting.

2.2 Reward Function

The goal in racing is to drive as quickly as possible, while not violating the track limits. We formulate this objective using a dense reward function that penalizes violating two constraints. The first is a track constraint which is triggered if the deviation to the center line is larger than half the track width $|n_{t+1}| \leq w_{\text{tr}}/2$, which was implemented by adding a safety margin to the actual track width. The second constraint 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. We include a penalization in the reward, using the tire ellipse constraint described in [25]. 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 a good dense approximation of the sparse minimum lap time reward. The combined reward function r_t is defined as $-c$ if constraints violated, and $p_{t+1} - p_t$ otherwise. The proposed function rewards fast driving and penalizes deviating from the track or exploring poorly modelled state space regions.

3 Sim-to-Real

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). Each episode starts at a random initial state and uses relative short episodes, to improve exploration [26]. We use Soft Actor Critic (SAC) [27] 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 C. The SAC implementation is based on Stable Baselines [28], and we implement our simulator, explained in Section A, as an OpenAI Gym environment [29]. 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 using the CPU (Intel i9-9900K), which takes about 36 hours to complete.

3.1 Simulator and Model Randomization

Our simulator is based on the dynamic bicycle model proposed in [1], using Pacejka tire models [30] to model the interaction with the ground. However, this simple vehicle model is over-optimistic and does not capture the stochasticity 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. It captures well the effect of increased uncertainty in case of rapid changes in the system. 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 and include 80% of all recorded errors. Including all errors, or overstating the noise level, would result in over-conservative policies.

3.2 Policy Regularization

Regularizing action sequences is motivated by avoiding excessive and jerky inputs, which can damage actuators and degrade performance. Furthermore, excitation of a system with high-frequency inputs often results in difficult to model behaviors (e.g. load transfer in driving). In model-based control approaches it is common to consider an input cost that penalizes the control inputs. In RL-based approaches such regularizers are less common, but have recently attracted some interest [31].

One standard approach to regularize the inputs is to augment the reward function, and to add a quadratic cost on the applied actions, which we call *reward action regularization*. Another common 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.

3.2.1 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. The goal is to penalize high values of the actions applied on the first step. This is achieved by adding a quadratic term to the SAC policy update step, which becomes

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

while the (action) value function update steps remain unchanged. Moreover, through the choice of the positive semidefinite matrix M, the regularization can be tuned separately for each action.

3.3 Sim-to-Real Experimental Results

The performance on the real platform of the policies with different regularizers trained in simulation is shown in Figure 1. We compare the mean lap time and time outside of the track constraints achieved by the most successful learned policies (all including model randomization), and by a benchmark MPC controller [32]. Large lap times indicate collisions with the track walls, while lower lap times and constraint violations correspond to better 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 brings an improvement, but the best performance is achieved by the policy 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.

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

	Avg. Lap Time[s]	Avg. Time Out of Constraints[s]
SAC	13.53	2.15
SAC + π weight reg	13.18	1.79
SAC + π output reg	11.43	1.07
MPC	10.57	1.53

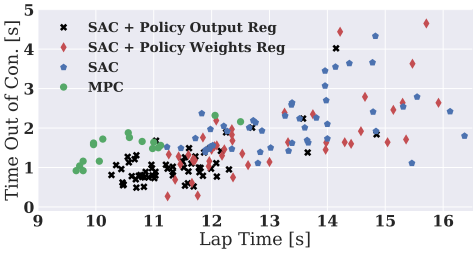

Figure 1: Performance of different policies trained in simulation.

4 Policy Refinement on the Car

To further refine the learned policy, while avoiding learning from scratch, we introduce interaction with the real physical system. We propose an approach, 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. SAC is running as a secondary task on the same computer, updating its policy and (action) value function based on the replay buffer. After 100 steps of SAC (2.5s), 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 the low power Intel i7-10510U CPU.

4.1 Policy Refinement Experimental Results

We start with the sim-to-real policy that can already drive and is less damaging to the robotic platform than a random policy. We show the learning performance in Figure 2, where each data-point is the average of the last 12 minutes of driving (the interval between changing the car batteries). Figure 2 shows a performance deterioration at the beginning of the learning phase, followed by a recovery, and a drastic improvement by the end of the three-hour interaction with the real car. Both the lap time and the constraints violation benefit from the policy refinement, as shown in Figure 3. The policy refinement reduced the mean lap time achieved by sim-to-real by 0.42 s, achieving 11.01 s per lap, and the constraint violations by 0.63 s, achieving 0.44 s per lap. This brings the refined policy close to the state-of-the-art MPC method in terms of lap time, while reducing the number of constraint violations by 71%. Interestingly, 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.

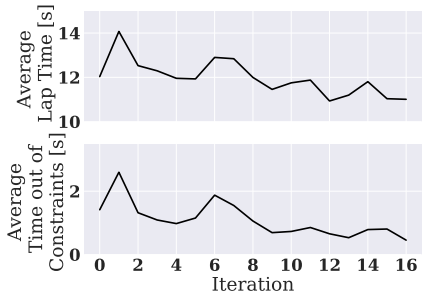

Figure 2: Performance during policy refinement.

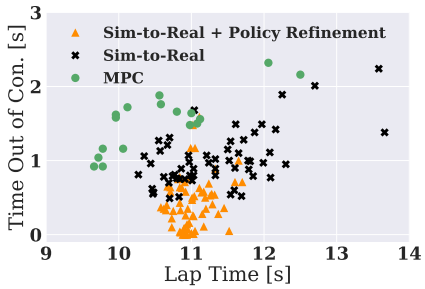

Figure 3: Performance of the policy refined on the car.

5 Conclusion

In this work we successfully applied the RL algorithm SAC to the autonomous racing problem on a miniature race track. Model randomization and policy output regularization strategy enable sim-to-real with good performance on the racing platform. We further demonstrate a performance improvement 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.

References

[1] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015.

[2] Ugo Rosolia, Ashwin Carvalho, and Francesco Borrelli. Autonomous racing using learning model predictive control. In *American Control Conference (ACC)*, 2017.

[3] Robin Verschueren, Stijn De Bruyne, Mario Zanon, Janick V Frasch, and Moritz Diehl. Towards time-optimal race car driving using nonlinear mpc in real-time. In *Conference on Decision and Control (CDC)*, 2014.

[4] Craig Earl Beal and J Christian Gerdes. Model predictive control for vehicle stabilization at the limits of handling. *IEEE Transactions on Control Systems Technology*, 21(4):1258–1269, 2012.

[5] Juraj Kabzan, Lukas Hewing, Alexander Liniger, and Melanie N Zeilinger. Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, 2019.

[6] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic mpc for model-based reinforcement learning. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[7] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A Theodorou, and Byron Boots. Imitation learning for agile autonomous driving. *The International Journal of Robotics Research*, 39(2-3):286–302, 2020.

[8] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *International Conference on Robotics and Automation (ICRA)*, 2018.

[9] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[10] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[11] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[12] Steven Lake Waslander, Gabriel M Hoffmann, Jung Soon Jang, and Claire J Tomlin. Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning. In *International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[13] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

[14] Haitham Bou-Ammar, Holger Voos, and Wolfgang Ertel. Controller design for quadrotor uavs using reinforcement learning. In *International Conference on Control Applications*, 2010.

[15] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[16] Jacky Baltes and Yuming Lin. Path tracking control of non-holonomic car-like robot with reinforcement learning. In *Robot Soccer World Cup*, 1999.

[17] Danial Kamran, Junyi Zhu, and Martin Lauer. Learning path tracking for real car-like mobile robots from simulation. In *European Conference on Mobile Robots (ECMR)*, 2019.

[18] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[19] Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real world labels. In *International Conference on Robotics and Automation (ICRA)*, 2019.

[20] Błażej Osiński, Adam Jakubowski, Piotr Miłoś, Paweł Ziecina, Christopher Galias, and Henryk Michalewski. Simulation-based reinforcement learning for real-world autonomous driving. *arXiv preprint arXiv:1911.12905*, 2019.

[21] Alexander Amini, Igor Gilitschenski, Jacob Phillips, Julia Moseyko, Rohan Banerjee, Sertac Karaman, and Daniela Rus. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robotics and Automation Letters*, 5(2):1143–1150, 2020.

[22] Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[23] Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization. *arXiv preprint arXiv:1910.09191*, 2019.

[24] José L Vázquez, Marius Brühlmeier, Alexander Liniger, Alisa Rupenyan, and John Lygeros. Optimization-based hierarchical motion planning for autonomous racing. *arXiv preprint arXiv:2003.04882*, 2020.

[25] Juraj Kabzan, Miguel de la Iglesia Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, et al. Amz driverless: The full autonomous racing system. *arXiv preprint arXiv:1905.05150*, 2019.

[26] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[27] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[28] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[29] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[30] Hans B Pacejka and Egbert Bakker. The magic formula tyre model. *Vehicle system dynamics*, 21(S1):1–18, 1992.

[31] Richard Cheng, Abhinav Verma, Gabor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel W. Burdick. Control Regularization for Reduced Variance Reinforcement Learning. *arXiv e-prints*, page arXiv:1905.05380, May 2019.

[32] Alexander Liniger and John Lygeros. Real-time control for autonomous racing based on viability theory. *IEEE Transactions on Control Systems Technology*, 27(2):464–478, 2017.

A Simulator and Model Randomization

The uniform distributions we use to randomize the simulated dynamics are $\varepsilon_{v_x} = \mathcal{U}(-1.5, 1.5)$, $\varepsilon_{v_y} = \mathcal{U}(-2.5, 2.5)$, and $\varepsilon_{\omega} = \mathcal{U}(-2.0, 2.0)$. To qualitatively demonstrate the effect of our model randomization technique, we estimate the tire forces from state-action sequences, and compare these force estimates obtained from experimental and simulation sequences. Figure 4 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 4 on the right. We see that the uncertainty we use in our simulator resembles the one appearing in the real recorded data.

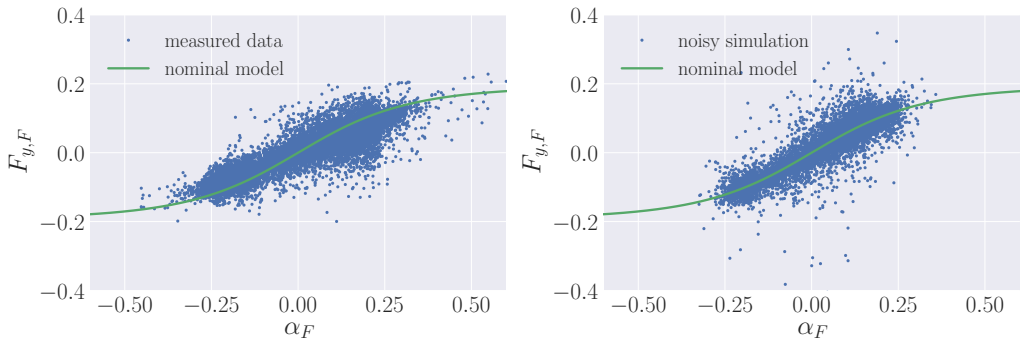

Figure 4: Estimated tire forces vs Pacejka tyre model, real measurements left and noisy simulation right, both using our trained sim-to-real policy.

B Policy Output Regularization

In order to achieve smooth actions, 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,

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

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

$$Q^\pi_M(s_0,a_0) = -a_0^T M a_0 + \underset{\tau\sim\pi}{\mathbb{E}}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H\left(\pi(s_t)\right)\right] \tag{2}$$

$$Q^\pi_M(s_0,a_0) = -a_0^T M a_0 + Q^\pi(s_0,a_0), \tag{3}$$

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

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

Equation (3) shows a tight relation between the standard action value function Q^π and the regularized version Q^π_M. Given this relation, the policy loss (4) can be rewritten as

$$\nabla_\theta \frac{1}{|B|}\sum_{s_t\in B} -\left[Q_{\phi}\left(s_t,\pi_\theta(s_t)\right) + \alpha H\left(\pi_\theta(s_t)\right)\right] + \mathbb{E}[\pi_\theta(s_t)]^T M \mathbb{E}[\pi_\theta(s_t)]. \tag{5}$$

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

Note that 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.

B.1 Regularization Comparison

To compare the different regularization methods we train four different policies, (i) vanilla SAC, (ii) SAC with reward action regularization (4) ($\tilde{M} = \text{diag}(0.005, 0.001)$), (iii) SAC with policy

	SAC	SAC + Reward Action Reg.	SAC + Policy Weights Reg.	SAC + Policy Output Reg.	MPC
Lap Time $[s]$	10.0	11.3	9.8	9.8	8.7
Con. Viol. $[s \times 10^{-2}]$	3.6	15.6	9.0	1.7	8.6
$d^2[10^{-1}]$	3.4	2.1	3.5	3.0	5.8
$\delta^2[rad^2 \times 10^{-2}]$	4.1	4.0	3.9	3.9	5.5
$\dot{d}^2[s^{-2} \times 10^{1}]$	4.6	0.8	4.0	1.3	3.0
$\dot{\delta}^2[\frac{rad^2}{s^2}]$	2.3	1.9	1.7	1.8	1.7
$\ddot{d}^2[s^{-4} \times 10^{4}]$	44.4	1.6	30.3	4.9	9.5
$\ddot{\delta}^2[\frac{rad^2}{s^4} \times 10^{3}]$	21.8	15.8	9.4	11.0	2.4

Table 1: Comparison of action smoothness metrics of different policies

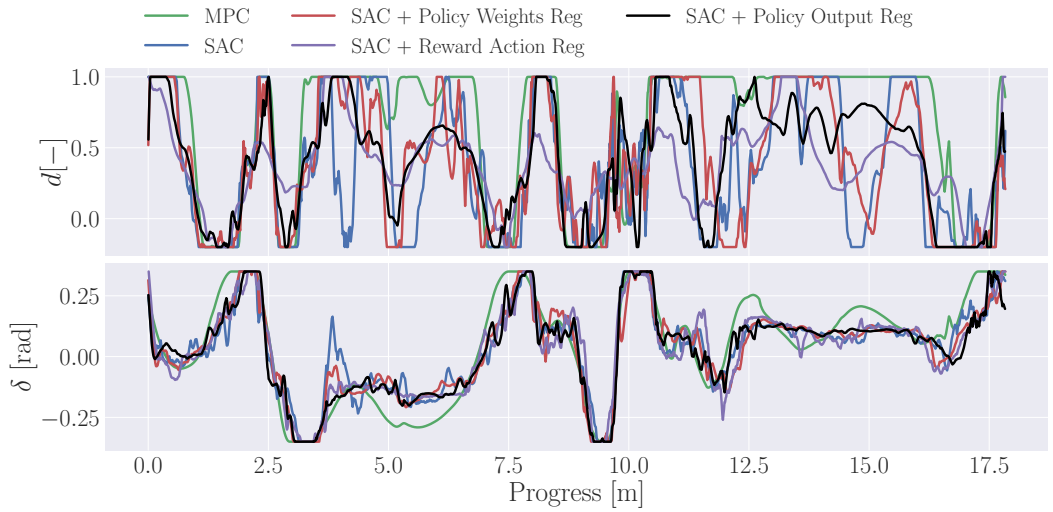

Figure 5: Comparison of dc and δ of different policies over one lap.

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 implemented in the same simulation environment with the model randomization technique explained in Section A. In Table 1 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 mean of the squared physical inputs (d, δ), and their squared 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 4 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 5, 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 3 times higher, which is problematic for the sim-to-real transfer. Finally, method (iv) has smooth duty cycle, but aggressive steering. Even though this is a strategy different than the MPC, it allows for a successful real-to-sim 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 5 look visibly better than the other RL based methods.

C Policy Refinement Setup

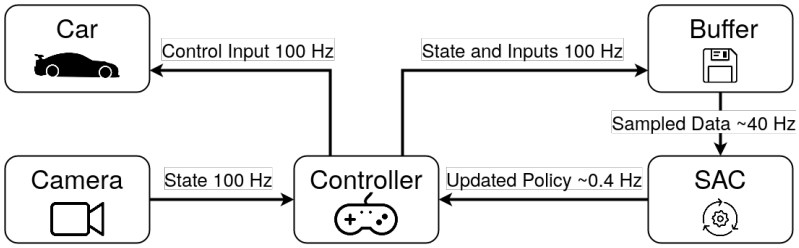

Figure 6: Diagram of the online reinforcement learning architecture

We perform the experimental policy refinement and show that this approach 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. More importantly, we start with a 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.