DeepRacer: Autonomous Racing Platform for Experimentation with Sim2Real Reinforcement Learning

Bharathan Balaji[1*], Sunil Mallya[1*], Sahika Genc[1*], Saurabh Gupta[1], Leo Dirac[1], Vineet Khare[1], Gourav Roy[1], Tao Sun[1], Yunzhe Tao[1], Brian Townsend[1], Eddie Calleja[1], Sunil Muralidhara[1], Dhanasekar Karuppasamy[1]

***Abstract*— DeepRacer is a platform for end-to-end experimentation with RL and can be used to systematically investigate the key challenges in developing intelligent control systems. Using the platform, we demonstrate how a 1/18th scale car can learn to drive autonomously using RL with a monocular camera. It is trained in simulation with no additional tuning in the physical world and demonstrates: 1) formulation and solution of a robust reinforcement learning algorithm, 2) narrowing the reality gap through joint perception and dynamics, 3) distributed on-demand compute architecture for training optimal policies, and 4) a robust evaluation method to identify when to stop training. It is the first successful large-scale deployment of deep reinforcement learning on a robotic control agent that uses only raw camera images as observations and a model-free learning method to perform robust path planning. We open source our code and video demo on GitHub[2].**

I. Introduction

Reinforcement Learning (RL) has been used to accomplish diverse robotic tasks: manipulation [1], [2], [3], [4], locomotion [5], [6], navigation [7], [8], [9], [10], flight [11], [12], interaction [13], [14], motion planning [15], [16] and more. Due to high sample complexity and safety requirements, it is common to train the RL agent in simulation [1], [5], [17]. To reduce training time and encourage exploration, the agent is usually trained with distributed rollouts [18], [19], [20], [21]. For a successful transfer to the real world, researchers use calibration [2], [22], domain randomization [23], [24], [25], [12], fine tuning with real world data [9], and learn features from a combination of simulation and real data [26], [27].

To experiment with robotic reinforcement learning, one needs to have expertise in many areas, access to a physical robot, an accurate robot model for simulations, a distributed training mechanism and customizability of the training procedure such as modifying the neural network and the loss function or introducing noise. For the uninitiated, dealing with this complexity is daunting and dissuades adoption. As a result, much of prior work is limited to a single robot [1], [23], [28] or a few robots [16]. We reduce the learning curve and alleviate development effort with DeepRacer.

DeepRacer supports state-of-the-art deep RL algorithms [29], simulations with the OpenAI Gym [17] interface, distributed rollouts and integration with cloud services. We introduce a training mechanism that decouples RL policy updates with the rollouts, which enables independent scaling of the simulation cluster and supports popular simulators like Gazebo [30]. The DeepRacer 1/18th scale car is one realization of a physical robot in our platform that uses RL for navigating a race track with a fisheye lens camera. The car hardware includes a GPU for executing the neural network policy locally, live streams the camera view over WiFi, the compute battery supports $\sim$6 hours of development time and retails at $400. We have a corresponding robot model in simulation, along with rendering for multiple race tracks. We can train the RL policy with different simulation parameters and multiple tracks in parallel using distributed rollouts.

We learn an end-to-end policy for navigating a race track. We use a single grayscale camera image as observation and discretized throttle/steering as actions. We train in simulation using the Proximal Policy Optimization (PPO) algorithm [31], which can converge in $<$5 minutes and $\sim$5000 simulation steps. With no pre-processing, real world data or expert labeling, the learned policy successfully transfers from simulation to real tracks (sim2real [32]). The entire process from training a policy to testing in the real car takes $<$30 minutes. Multiple models can be trained in parallel with on-demand compute and stored in the car. Thousands of users have designed their own reward functions, trained their models on our platform, and demonstrated real track navigation. To the best of our knowledge, this is the first demonstration of model-free RL based sim2real at scale.

DeepRacer serves as a testbed for many areas of RL research such as reducing sample complexity [33], sim2real [34] and generalizability [35]. The car can log camera images, inertial sensor measurements, policy decisions. Simulations can be randomized with different tracks, lighting, sensor and actuator noise. The learned policy can underfit/overfit to the simulation settings. We use a robust evaluation method to identify when the learned policy will generalize to the real world. We evaluate multiple checkpoints of the saved policy with domain randomization such as action noise and different starting points. Models that give good results in robust evaluation generalize well to the real world. Our policies trained with domain randomization generalize to multiple cars, tracks and to variations in speed, background, lighting, track shape, color and texture.

II. Related Work

RL has been used in robotics for several decades [36], [37], [38], [39]. Initial works used low dimensional state spaces due to scalability challenges. RL concepts were generalized to high dimensional problems with deep networks [40], [41], [42]. High variance, sample complexity

[1]Authors are employees of Amazon Web Services. $*$ contributed equally. Send all correspondence to: bhabalaj@amazon.com

[2]DeepRacer training source code: https://git.io/fjxoJ

and replicability challenges [43] in deep RL algorithms led to development of simulators [44], benchmarks [17], [45] and libraries [46], [47]. We build upon these works to create a platform for experimentation with simulation and real robots.

Distributed Rollouts: Algorithms that use distributed rollouts, where multiple simulations are executed in parallel to collect experience data, were introduced to reduce training time [2], [20], [48]. OpenAI Baselines [47] uses OpenMPI [49] to support distributed gradient algorithms, where each worker computes gradients on data collected. OpenAI Rapid [2] generalizes it to a distributed system for the PPO algorithm and demonstrate sim2real transfer on dextrous manipulation. Flex [19] extends the same distribution mechanism to use GPUs for simulation and hence can run 750 humanoid MuJoCo simulations with a single GPU. Chebotar et al. [50] use Flex to demonstrate sim2real transfer for manipulation. Surreal [18] uses a decoupled rollout mechanism to support the experience replay algorithms, where each worker stores the experience data in a buffer and a separate training worker computes gradients. Ray RLlib [21], [51] introduces a stateful actor framework to support distributed rollouts. DeepRacer integrates with Intel Coach library [29] that supports >20 deep RL algorithms in an easy-to-use, modular interface. DeepRacer uses the same rollout mechanism as Surreal, and extends support for Gazebo. Similar to Rapid, DeepRacer can use different simulation settings for each worker and have separate evaluation workers that validate the performance of the current policy.

Sim2Real: Training RL policies in the real world is challenging due to high sample complexity and safety issues. Simulations alleviate these concerns and serve as a testbed to experiment with algorithms and debug software. However, sim2real transfer is challenging because of differences in dynamics, imagery and as simulated models are just approximations of the real world [23], [24], [34]. Domain randomization, where simulation parameters are perturbed during training, has been used for successful sim2real transfer for various robotic tasks [2], [12], [50]. Methods include adding noise in dynamics [23], [2] and imagery [12], [52], learning model ensembles [53], [54], adding adversarial noise [25], [55] and assessing simulation bias [24]. Domain adaptation [56] has also been used for sim2real, particularly to address the visual reality gap [26], [57], [58], [59]. DeepRacer serves as a platform to reproduce and experiment with sim2real methods. We demonstrate various forms of domain randomization in our experiments. Navigation with the DeepRacer car can be structured from simple, low speed, lane following to complex tasks such as high speed racing or commuting in traffic.

Our distributed rollout mechanism facilitates iterative experimentation as policies converge faster and helps identify underfitting/overfitting. Prior sim2real works use a fixed number of simulation steps [2], [23], [60], [61]. We show that policies can both underfit and overfit to the simulation while training, as identified by prior works [24], [35], [62]. We use a separate robust evaluation to identify the policy checkpoints that are likely to transfer well to the real world.

TABLE I: Comparison of DeepRacer with contemporary RL and self-driving platforms. ★indicates partial support.

Platform	Simulation	Rigid Body Dynamics	Deep RL	Distributed Rollouts	Physical Robot	Sim2Real Demo	GPU on Robot	Robot Cost (USD)
AutoRally [63]	✓	✓	✓	✗	✓	✗	✓	10K
BARC [64]	✗	✗	✗	✗	✓	✗	✗	500
Blue [65]	✗	✗	✗	✗	✓	✗	✗	5K
CARLA [66]	✓	★	✓	✓	✗	✓	✓	✗
DonkeyCar [67]	✓	★	✓	✗	✓	✓	✗	200
Duckietown [68]	✓	★	✓	✗	✓	✓	✗	150
F1/10 [69]	✗	✗	✗	✗	✓	✗	✓	3600
Fetch [1], [23]	✓	✓	✓	✓	✓	✓	✗	100K
Flex [19]	✓	✓	✓	✓	✗	✓	✗	✗
RACECAR [70]	✓	✓	✗	✗	✗	✗	✓	2600
MuSHR [71]	✓	✗	✗	✗	✓	✓	✓	900
Poppy [72]	✓	✓	✓	✓	✓	✓	✗	350
RLlib [21]	✓	✓	✓	✓	✗	✗	✗	✗
Surreal [18]	✓	✓	✓	✓	✗	✗	✗	✗
DeepRacer	✓	✓	✓	✓	✓	✓	✓	400

Sim2Real Navigation: Many works rely on simulators only for testing and use methods such as state estimation, motion planning and model predictive control (MPC) [68], [73], [74] for navigation. Other works have used imitation learning, where expert demonstrations are given either by a person [67], [75] or with an MPC algorithm [76], [77]. Kahn et al. [10] directly learn the RL policy in the real car, with a fixed maneuver when collision occurs. Domain randomization and image segmentation in simulations have been used to close the visual reality gap with a model based controller [12], [66], [78]. Image pre-processing [79], learned embeddings [9] and depth camera [80] have been used to achieve sim2real transfer. Bharadhwaj et al. [27] demonstrate sim2real transfer by mixing expert demonstrations with simulations. We observe that prior sim2real works rely on a model based controller for high speed navigation [78], [81] or achieve slow speeds because of poor transfer of dynamics [79], [80]. With DeepRacer, we demonstrate speeds of 1.6m/s with a single grayscale monocular image as input and discretized steering/throttle as output. We use simple, non-recurrent networks for our policy and still demonstrate robustness in the real world to multiple cars, tracks, and variations in the environment. We also achieve slow speed (0.5m/s) sim2real transfer with <5 minutes of training.

Table I compares DeepRacer with other platforms for RL, sim2real and autonomous driving. The other simulation platforms can also be used with DeepRacer. We provide an easy-to-use, economical and flexible platform with support for distributed RL, domain randomization and robust evaluation. DeepRacer tools have enabled us to replicate sim2real RL policy transfer with consistency and at scale.

III. Autonomous Racing with RL

In our formulation, the agent steers the car and the environment is the race track. The track is marked by white lanes, there is a single car on track with no obstacles and the

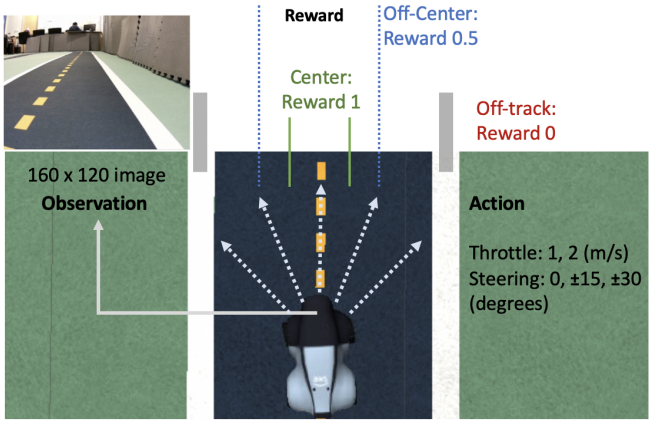


Fig. 1: Observation, action and reward for DeepRacer agent

car only moves forwards. The image from the car's camera is the observation, and actions are the throttle/steering of the car. As the agent does not receive the full state such as the the track layout, this is a partially observed Markov Decision Process. An episode starts with the car somewhere on track and finishes when the car goes off-track or finishes a lap.

The images from the camera are streamed at 15 fps, downsized to 160 x 120 pixels and converted to grayscale. We discretize the actions to 10 values, with 2 levels for throttle and 5 for steering. Users can customize this discretization, which get mapped to low level controls. We use discrete actions as the mapping to low level control is non-linear and challenging to calibrate in the continuous space. We fix the maximum throttle in simulation and set it manually in the real car. We incentivize the agent to stay close to the center line of the track. If the car is at the edge of the track, a small deviation can off-road the car and the track is not visible in the image. Staying close to the center of the track leads to a stable policy. Users can customize this reward function. Figure 1 illustrates our problem formulation.

A. Reinforcement Learning Algorithm

We use PPO, a state-of-the-art policy gradient algorithm [31]. The algorithm uses two neural networks during training – a policy network and a value network. The policy network decides which action to take given an image as input and the value network estimates the expected cumulative discounted reward given the image. The agent initializes a policy that takes random actions. The policy network is used to interact with the simulation environment to collect data. The resulting dataset is used to update the policy and value networks as per the algorithm's loss function. The updated policy is used to interact with the environment to collect more data and the training cycle continues until a time limit.

The policy loss function maximizes the actions that give higher rewards on average as given by the generalized advantage estimation algorithm [82] and applies a clipped importance sampling weight as the policy that collects the dataset is an older version of the policy being updated. The value loss function uses the mean squared error between the predicted value and the observed value. Only the policy

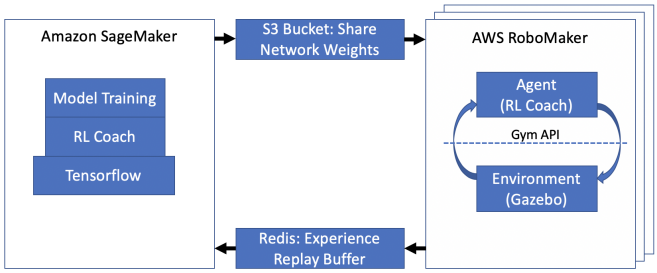


Fig. 2: Training the agent with DeepRacer distributed rollouts

network gets deployed in the real car. By default, we use three convolutional layers and two fully connected layers for both networks. We train a new policy every 20 episodes. The full list of hyperparameters is given in our source code.

IV. DeepRacer Design and Implementation

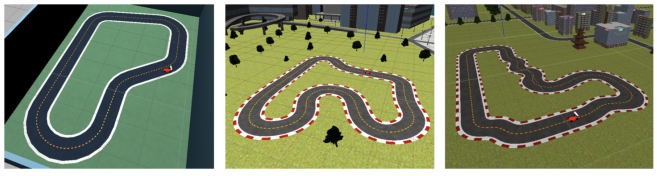
We decouple the simulation data collection from the policy updates. We use RoboMaker [83] for our simulations with Gazebo and SageMaker [84] to train our policy with the RL Coach [29] library. Simulations help us train without manual effort. The decoupled training allows us to use separate machines which are specialized for simulations (e.g. license, Mac/Windows OS) and neural network training (e.g. GPU, large RAM) respectively. We also get the flexibility to launch multiple simulations each with their own settings for domain randomization as well as evaluate policies in parallel.

A. Training Workflow

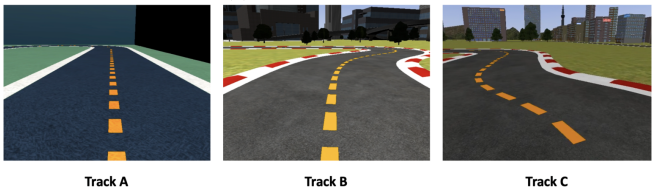
Figure 2 shows the DeepRacer training workflow. The training starts by initializing the policy/value network models and hyper-parameters in SageMaker. The neural network models are saved in S3 [85], an object store service. RoboMaker initializes the simulation, the agent and loads the models from S3. The agent interacts with the simulation over the OpenAI Gym interface. The agent takes actions a (steering/throttle) based on the observation o (camera image). The simulator updates the position of the car based on the action and returns with the updated camera image and reward r. The experiences collected in the form of $\langle o_t, a_t, r_t, o_{t+1} \rangle$ are stored in Redis [86], an in-memory database. SageMaker trains the neural networks with data collected in Redis and saves the models in S3. RoboMaker copies the model from S3 and creates more experience data. The cycle continues until training stops. The models in S3 are continually evaluated in a separate simulation to assess convergence and generalizability. Models in S3 can be deployed on the real car. While we show our results with the PPO algorithm, our architecture can be used for various experience replay based algorithms such as DQN [40], DDPG [87] and SAC [88]. Robomaker can be replaced with other simulators that can integrate with the Gym interface.

B. Training with Amazon SageMaker

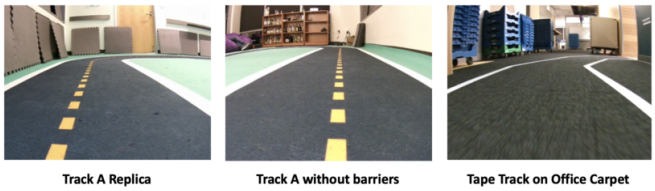
SageMaker is a platform to train and deploy machine learning models at scale using the Jupyter Notebook [89] as interface. SageMaker integrates RL algorithms using Coach



(a) Simulation tracks



(b) Camera view of simulation tracks



(c) Camera view of real world tracks

Fig. 3: We train in multiple tracks and evaluate with a replica track as well as a track made with duct tape.

and RLlib [21] libraries that build on top of existing deep learning frameworks. SageMaker uses RL Coach to support the decoupled simulation based training used in DeepRacer, and RLlib for integrated simulation and training. The libraries are packaged in a Docker container [90] and training can be launched in a cluster of machines with different configurations (CPU/GPU/RAM). The training clusters are created on-demand and billed per second, freeing users from infrastructure maintenance. Metrics such as rewards per episode, the policy entropy, cpu/memory use are visualized, source code is saved and logs are recorded. Users can launch experiments in parallel and search across experiment metadata. In addition to autonomous racing, SageMaker contains RL examples for HVAC control, robot locomotion, portfolio management and more.

C. Simulation with AWS RoboMaker

RoboMaker is a cloud service to develop, test and deploy robot software. It uses Gazebo for simulation. A *robot model* describes each component of the DeepRacer car - the chassis, wheels, camera, Ackermann steering - their dimensions, how they link together, their properties such as mass and camera angle. We create our tracks and background environment in Blender, a 3D modeling software and import it into Gazebo. We use the ODE physics engine that simulates the laws of physics using the robot model and takes into account factors like collision, friction, acceleration, etc. A rendering engine, OGRE, visualizes the graphics. We use Gazebo plugins to add the camera and light sources. We use ROS [91] for communication between the agent and the simulation. The agent uses ROS to place the car in the track at the beginning of an episode, get images from the camera module, get the

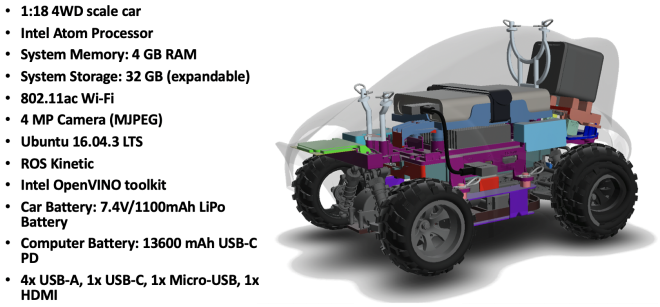


Fig. 4: DeepRacer Hardware Specifications

car's position, velocity, and send throttle, steering commands to control the car. Users can customize the simulation in Gazebo with their own robot models and environments.

D. Sim2Real Calibration

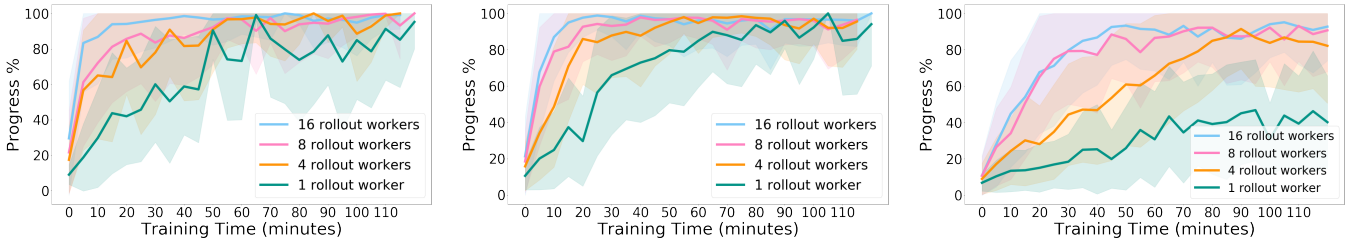
We have matched the URDF robot model to the measured dimensions of the car. We compared images from the real camera and calibrated the height, angle and the field of view of the simulation camera to match the real images. As DeepRacer camera can capture 15 fps, we match the simulation environment to use the same frame rate and use a producer-consumer mechanism to ensure one action per image. We map the agent's action space to the motor control commands by measuring the steering angles and speed of the car under different settings. We have created a real world track that is identical in color, shape and dimensions with one of the simulation tracks. We use barricades around this track to reduce visual distractions. In addition, we have eight other tracks with varying shapes, backgrounds and textures.

E. Calculating Rewards

We compute an ordered set of points along the middle of the track, called *waypoints*, to estimate the relative position of the car on track. The track and the background are modeled as a polygon mesh. We separate the track mesh from the background and identify the border edges as those which belong to a single triangle. We get two boundaries corresponding to inner and outer part of the track by grouping the border vertices. We construct a bipartite graph from the two sets of vertices and compute the linear sum assignment using the Euclidean distance as edge length. This gives us border vertices parallel to each other on both sizes of the track. The waypoints are the mean of the vertices connected by each edge. The spline is the line joining the waypoints. The car starts an episode at a waypoint. We flag the car as off-track when it deviates from the spline by more than half the track width. We measure the car's progress by the relative distance it covers compared to the length of the spline.

F. DeepRacer Hardware

Figure 4 gives an overview of DeepRacer hardware. We have designed the car for experimentation while keeping the cost nominal. The Intel Atom processor with a built-in GPU can perform $>$15 inferences per second with our default five layer neural network. The motors are equipped with



(a) Training with Track A and maximum throttle of 1 m/s

(b) Training with Track A and maximum throttle of 1.67 m/s

(c) Training with Track B and maximum throttle of 1.67 m/s

Fig. 5: Training with multiple rollout workers. Progress on track is reported across two runs in a ml.p3.2xlarge instance in SageMaker, which has one NVIDIA V100 GPU. Each rollout is a separate simulation job in RoboMaker.

electronic speed controllers. We can use the car as a regular computer with a monitor, mouse and keyboard connected via HDMI and USB. The camera connects over USB and there are three USB ports for extensions. The 13600 mAh compute battery lasts ∼6 hours. The 1100 mAh drive battery lasts for ∼45 minutes in typical experiments. The WiFi chip enables remote monitoring and programming. We built the car software on top of ROS. We can load multiple trained models over WiFi. We use Intel OpenVino to convert our Tensorflow models to an optimized binary for fast inference. The camera images are fed to the OpenVino inference engine and a real-time video feed on a browser. There is a web UI for calibrating steering and throttle. The model inference results are converted to motor control commands based on the calibration and action space mapping. In addition, the browser has an interface for manual joystick like control.

V. Evaluation

We evaluate our track navigation policies extensively across multiple tracks, with domain randomization in both simulation and real world. We have created a replica of Track A with the track printed on carpet with the same dimensions as in simulation. We place barriers around the track to reduce distractions and evaluate performance both with and without barriers as well as different speeds and lighting conditions. We also made a custom "tape track" with 2 inch white duct tape in our office corridor to test model robustness. The track is roughly 24 inches wide, 12m in length, traverses both carpet and concrete, has multiple turns and the car camera is exposed to clutter and bright lights in the background.

A. Training with Multiple Rollouts

We train policies with three different conditions: on Track A with a maximum throttle of 1 m/s, on Track A with throttle 1.67 m/s and on Track B with throttle 1.67 m/s. The task gets harder at higher speeds. Track B is more difficult to navigate because of background with buildings and higher number of turns. Each episode starts with a different waypoint so that all parts of the track are experienced by the policy. We use p3.2x instance for training in SageMaker and run each experiment twice for 2 hours. Figure 5 shows the progress on track during training with different number of rollout workers.

As we expect, more rollout workers lead to faster convergence. There is diminishing returns as we increase workers, 16 workers give a slightly faster convergence compared to 8. Somewhat surprisingly, the higher throttle of 1.67 m/s helped speedup convergence in Track A. We hypothesize that the agent collects more uniform experience with the faster speed and this helps with convergence. Track B takes longer to converge but follows similar trends as Track A.

B. Robust Evaluation

We test whether robust evaluation in simulation is indicative of real world performance. If true, we can identify when to stop training in simulation and avoid underfitting/overfitting. We can tune our hyper-parameters entirely in simulation and avoid extensive testing in the real world. We train policies with increasing levels of domain randomization and evaluate the policy in both simulation and real.

Our baseline case is trained on Track A with no domain randomization and throttle of 1 m/s. For domain randomization, we train policies on Track A with (i) up to 10% uniform random noise to steering and throttle (action noise), (ii) reverse direction of travel each episode (reverse), (iii) include both action noise and reverse, and (iv) train on Track B with both action noise and reverse. For robust evaluation, we add uniform random noise to actions, evaluate in multiple starting positions and both directions of travel on Track A. For naive evaluation, we evaluate on Track A with a fixed starting point without randomization. Both evaluations test each checkpoint 10 times in simulator. We pick six policies during training from checkpoints 5 through 30, and test their sim2real performance in the Track A replica with 3 trials for each direction of travel. The model performance varies with speed, but it is difficult to maintain a constant speed due to changing battery levels and as the model switches between throttle levels. For sim2real experiments we ensure the model completes a lap in 18 to 22 seconds (0.8-1 m/s). In simulation, the models complete the lap in ∼35 seconds, so we test the policy at about double speed in the real track.

Figure 6 shows the experimental results. The model that perform consistently well with robust evaluation also perform well on the real track. The models are particularly robust when a sequence of checkpoints perform well in simulator. Reversing the direction of travel significantly improves model performance. Action noise does not help by itself, but improves performance when combined with reverse. Policies trained on Track B do not perform well for checkpoints in

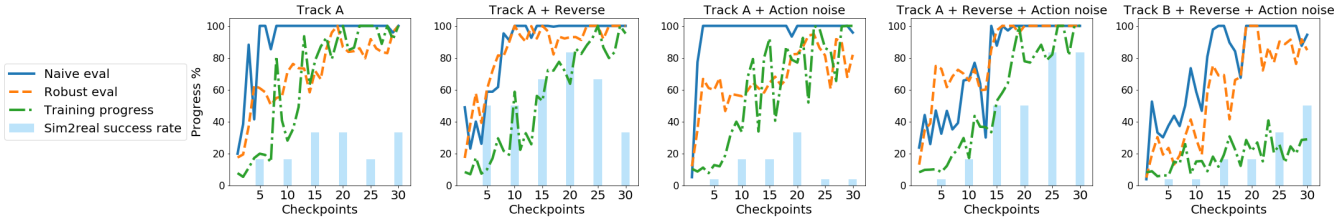


Fig. 6: Robust evaluation with domain randomization as a criteria to select policy checkpoints for sim2real transfer.

TABLE II: Sim2Real for policies trained with regularization and domain randomization. Results are out of 6 trials.

Training Track	Type of Training	Checkpoint # (Progress %)	Training A Replica				Tape Track 0.7-0.9 m/s	Total
			0.5 m/s	1 m/s	Sunlight 0.8-1 m/s	No Barriers 0.8-1 m/s		
B	Default	54 (100)	5	3	0	1	3	12
C		53 (99.7)	5	3	2	3	3	16
D		50 (100)	5	3	3	3	0	13
B	L2=2e-5	53 (100)	5	4	2	4	2	17
	Dropout=0.3	49 (100)	**6**	3	5	5	4	23
	BatchNorm	41 (100)	4	2	1	4	2	13
	Throttle=0.33 m/s	21 (100)	2	0	0	0	2	4
	Throttle=1.67 m/s	72 (91.1)	**6**	4	5	**6**	2	23
	Throttle=2.33 m/s	79 (57.9)	**6**	5	5	**6**	2	24
B, D	Default	41 (100)	3	3	3	3	1	13
	Color Aug.	49 (100)	**6**	5	**6**	**6**	3	26
	Translation	37 (100)	**6**	5	5	3	3	22
	Shadow	46 (100)	5	3	5	3	2	18
	Sharpen	48 (89.5)	4	4	5	4	0	17
	Pepper	53 (98.9)	**6**	3	4	2	1	16
	All image aug	48 (100)	5	**6**	3	4	0	18
C	Best combo, Throttle=2.33 m/s	67 (91.7)	**6**	**6**	**6**	5	4	**27**

Figure 6, but with more training start performing well in both robust evaluation and real track, policy checkpoint 35 traversed the real track successfully 5 out of 6 trials.

The performance of the model changes dramatically at slower speeds (35s lap, 0.5 m/s), even checkpoint 5 of the policy trained on Track A with no randomization traverses the real track. This model is trained in <5 minutes. All the above policies were trained in <1 hour with 4 rollouts.

C. Robust Sim2Real

We test the robustness of sim2real by training on multiple tracks, with multiple speeds, regularization and domain randomization in actions and observations. By default, we train on Track B with throttle of 1 m/s, with action noise and reverse direction each episode. We pick model checkpoints based on performance in robust evaluation and test the policy on Track A replica in two speeds (0.5 m/s, 1 m/s), with bright sunlight, with no barriers and on tape track.

Table II summarizes our results. Training on a different track gives good sim2real results, but vary track to track. For regularization, we used L2 norm, dropout, batch normalization and an entropy bonus to the policy loss. We tested the models that give best performance in robust evaluation. Reducing the entropy bonus to 0.001 (it is 0.1 by default) and dropout with probability 0.3 were particularly effective. Larger throttle speeds in training increased the robustness of the model dramatically but also increased convergence time in the presence of action noise. Mixing multiple tracks during training did not lead to improvement in performance. We perturb the observation images with random color, horizontal translation, shadow, and salt and pepper noise, each with 0.2 probability. For random color, we combine the effects of random hue, saturation, brightness and contrast to create variations in observation. Random color was the most effective method for sim2real transfer.

We combine the best of our parameters and train a model on Track C with L2 regularlization, lower entropy bonus, dropout, color randomization and a maximum throttle of 2.33 m/s. This model performed the best overall in our experiments. The model consistently completed 11 second laps (1.6 m/s) in our Track A replica.

VI. Conclusion

DeepRacer is an experimentation platform for sim2real reinforcement learning. The platform integrates state-of-the-art Deep RL algorithms, multiple simulation engines with OpenAI Gym interface, provides on-demand compute, distributed rollouts that facilitates domain randomization and robust evaluation in parallel. We demonstrate DeepRacer platform features with a 1/18th scale car that navigates a race track using reinforcement learning. We have created a calibrated robot model for the car in Gazebo along with multiple race tracks. We demonstrate robust sim2real navigation performance trained in DeepRacer with PPO algorithm in both our real world replica track as well as a custom tape track. We achieve sim2real in real track with <5 minutes of training at slow speeds and achieve speeds of 1.6 m/s using models trained with tuned parameters. Thousands of users have replicated our model training and demonstrated sim2real RL navigation.

REFERENCES

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

[2] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, "Learning dexterous in-hand manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[3] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[4] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 262–270. [Online]. Available: http://proceedings.mlr.press/v78/rusu17a.html

[5] 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, 2019. [Online]. Available: https://robotics.sciencemag.org/content/4/26/eaau5872

[6] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne, "Feedback control for cassie with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1241–1246.

[7] S.-H. Hsu, S.-H. Chan, P.-T. Wu, K. Xiao, and L.-C. Fu, "Distributed deep reinforcement learning based indoor visual navigation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2532–2537.

[8] J. Choi, K. Park, M. Kim, and S. Seok, "Deep reinforcement learning of navigation in a complex and crowded environment with a limited field of view," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5993–6000.

[9] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[10] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[11] H. J. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng, "Autonomous helicopter flight via reinforcement learning," in *Advances in neural information processing systems*, 2004, pp. 799–806.

[12] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[13] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6015–6022.

[14] S. Christen, S. Stevsic, and O. Hilliges, "Guided deep reinforcement learning of control policies for dexterous human-robot interaction," *arXiv preprint arXiv:1906.11695*, 2019.

[15] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[16] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[18] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, "Surreal: Open-source reinforcement learning framework and robot manipulation benchmark," in *Conference on Robot Learning*, 2018, pp. 767–782.

[19] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 270–282. [Online]. Available: http://proceedings.mlr.press/v87/liang18a.html

[20] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1407–1416. [Online]. Available: http://proceedings.mlr.press/v80/espeholt18a.html

[21] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3053–3062. [Online]. Available: http://proceedings.mlr.press/v80/liang18b.html

[22] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[23] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[24] F. Muratore, F. Treede, M. Gienger, and J. Peters, "Domain randomization for simulation-based policy optimization with transferability assessment," in *Conference on Robot Learning*, 2018, pp. 700–713.

[25] A. Mandlekar, Y. Zhu, A. Garg, L. Fei-Fei, and S. Savarese, "Adversarially robust policy learning: Active construction of physically-plausible perturbations," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3932–3939.

[26] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner, "Darla: Improving zero-shot transfer in reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1480–1490.

[27] H. Bharadhwaj, Z. Wang, Y. Bengio, and L. Paull, "A data-efficient framework for training and sim-to-real transfer of navigation policies," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 782–788.

[28] K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running Dive into the Future of Infrastructure*, 1st ed. O'Reilly Media, Inc., 2017.

[29] I. Caspi, G. Leibovich, G. Novik, and S. Endrawis, "Reinforcement Learning Coach," Dec. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1134899

[30] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[32] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2real viewpoint invariant visual servoing by recurrent control," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4691–4699.

[33] S. M. Kakade *et al.*, "On the sample complexity of reinforcement learning," Ph.D. dissertation, University of London London, England, 2003.

[34] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*. Springer, 1995, pp. 704–720.

[35] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 1282–1289. [Online]. Available: http://proceedings.mlr.press/v97/cobbe19a.html

[36] M. J. Matarić, "Reinforcement learning in the multi-robot domain," in *Robot colonies*. Springer, 1997, pp. 73–83.

[37] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Purposive behavior acquisition for a real robot by vision-based reinforcement learning," *Machine learning*, vol. 23, no. 2-3, pp. 279–303, 1996.

[38] V. Gullapalli, J. A. Franklin, and H. Benbrahim, "Acquiring robot skills via reinforcement learning," *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 13–24, 1994.

[39] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial intelligence*, vol. 55, no. 2-3, pp. 311–365, 1992.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[41] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[42] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[43] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[44] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[45] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[46] O. S. Oguz, "Setting up a benchmark environment for deep reinforcement learning."

[47] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," *GitHub, GitHub repository*, 2017.

[48] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[49] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[50] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8973–8979.

[51] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan *et al.*, "Ray: A distributed framework for emerging {AI} applications," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 561–577.

[52] 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 *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.

[53] I. Mordatch, K. Lowrey, and E. Todorov, "Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 5307–5314.

[54] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," *arXiv preprint arXiv:1610.01283*, 2016.

[55] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2817–2826.

[56] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa, "Visual domain adaptation: A survey of recent advances," *IEEE signal processing magazine*, vol. 32, no. 3, pp. 53–69, 2015.

[57] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

[58] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 627–12 637.

[59] G. J. Stein and N. Roy, "Genesis-rt: Generating synthetic images for training secondary real-world tasks," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7151–7158.

[60] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[61] J. Matas, S. James, and A. J. Davison, "Sim-to-real reinforcement learning for deformable object manipulation," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 734–743. [Online]. Available: http://proceedings.mlr.press/v87/matas18a.html

[62] S. Whiteson, B. Tanner, M. E. Taylor, and P. Stone, "Protecting against evaluation overfitting in empirical reinforcement learning," in *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2011, pp. 120–127.

[63] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg, "Autorally: An open platform for aggressive autonomous driving," *IEEE Control Systems Magazine*, vol. 39, no. 1, pp. 26–55, 2019.

[64] J. Gonzales, F. Zhang, K. Li, and F. Borrelli, "Autonomous drifting with onboard sensors," in *Advanced Vehicle Control: Proceedings of the 13th International Symposium on Advanced Vehicle Control (AVEC'16), September 13-16, 2016, Munich, Germany*, 2016, p. 133.

[65] D. V. Gealy, S. McKinley, B. Yi, P. Wu, P. R. Downey, G. Balke, A. Zhao, M. Guo, R. Thomasson, A. Sinclair *et al.*, "Quasi-direct drive for low-cost compliant robotic manipulation," *arXiv preprint arXiv:1904.03815*, 2019.

[66] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 1–16. [Online]. Available: http://proceedings.mlr.press/v78/dosovitskiy17a.html

[67] W. Roscoe, "Donkey car: An opensource DIY self driving platform for small scale cars," http://donkeycar.com, 2019.

[68] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang *et al.*, "Duckietown: an open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1497–1504.

[69] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio *et al.*, "F1/10: An open-source autonomous cyber-physical platform," *arXiv preprint arXiv:1901.08567*, 2019.

[70] S. Karaman, A. Anders, M. Boulet, J. Connor, K. Gregson, W. Guerra, O. Guldner, M. Mohamoud, B. Plancher, R. Shin *et al.*, "Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at mit," in *2017 IEEE Integrated STEM Education Conference (ISEC)*. IEEE, 2017, pp. 195–203.

[71] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. S. M. Rockett, J. R. Smith, S. Choudhury, C. Mavrogiannis, and F. Sadeghi, "Mushr: A low-cost, open-source robotic racecar for education and research," *arXiv preprint arXiv:1908.08031*, 2019.

[72] M. Lapeyre, P. Rouanet, J. Grizou, S. Nguyen, F. Depraetre, A. Le Falher, and P.-Y. Oudeyer, "Poppy project: open-source fabrication of 3d printed humanoid robot for science, education and art," 2014.

[73] N. Wagener, C. an Cheng, J. Sacks, and B. Boots, "An online learning

approach to model predictive control," in *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[74] 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 *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[75] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[76] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," in *Robotics: science and systems*, 2018.

[77] M. Mueller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," in *Conference on Robot Learning*, 2018, pp. 1–15.

[78] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *arXiv preprint arXiv:1905.09727*, 2019.

[79] Q. Zhang and T. Du, "Self-driving scale car trained by deep reinforcement learning," *arXiv preprint arXiv:1909.03467*, 2019.

[80] K. Wu, M. Abolfazli Esfahani, S. Yuan, and H. Wang, "Learn to steer through deep reinforcement learning," *Sensors*, vol. 18, no. 11, p. 3650, 2018.

[81] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Vision-based high-speed driving with a deep dynamic observer," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1564–1571, 2019.

[82] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[83] "AWS RoboMaker," https://aws.amazon.com/robomaker/, 2019.

[84] "Amazon SageMaker," https://aws.amazon.com/sagemaker/, 2019.

[85] "Amazon S3," https://aws.amazon.com/s3/, 2019.

[86] "Redis," https://redis.io, 2019.

[87] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[88] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1861–1870. [Online]. Available: http://proceedings.mlr.press/v80/haarnoja18b.html

[89] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows." in *ELPUB*, 2016, pp. 87–90.

[90] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[91] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.