IMITATION LEARNING FOR GENERALIZABLE SELF-DRIVING POLICY WITH SIM-TO-REAL TRANSFER

Anonymous authors
Paper under double-blind review

ABSTRACT

Imitation Learning uses the demonstrations of an expert to uncover the optimal policy and it is suitable for real-world robotics tasks as well. In this case, however, the training of the agent is carried out in a simulation environment due to safety, economic and time constraints. Later, the agent is applied in the real-life domain using sim-to-real methods. In this paper, we apply Imitation Learning methods that solve a robotics task in a simulated environment and use transfer learning to apply these solutions in the real-world environment. Our task is set in the Duckietown environment, where the robotic agent has to follow the right lane based on the input images of a single forward-facing camera. We present three Imitation Learning and two sim-to-real methods capable of achieving this task. A detailed comparison is provided on these techniques to highlight their advantages and disadvantages.

1 INTRODUCTION

Imitation Learning (IL) uses demonstrations of an expert to uncover the optimal policy. Due to this, the agent can achieve expert-like behavior in the given environment.

IL is a feasible approach for problems where collecting labeled data is complicated, however acquiring expert demonstrations is a straightforward process. One such area is robotic control, as it is usually challenging to solve the particular task with a rule-based policy, but collecting demonstrations is in most cases is uncomplicated. As a result, IL has been widely used in this area.

ALVINN (Pomerleau, 1988) was one of the first imitation learning based self-driving solutions. It used Behavioural Cloning (BC) to carry out a real-world lane following task. Ross et al. (2010) presented improvements to the Behavioural Cloning formula and solved a self-driving task in a 3D racing game. Li et al. (2017) extended the Generative Adversarial Imitation Learning algorithm to learn a policy that can distinguish certain behaviors in human driving in a racing simulator.

Despite all of these, solving a real-world robotics task using solely Imitation Learning is problematic. The preferred approach is to train a model in a simulator and deploy it in the real-world domain. However, this is challenging as the model's performance usually declines in the real-world due to the differences of the two environments. To address this issue, it is recommended to apply sim-to-real techniques. The aim of these methods is to augment the learning process in a way that an algorithm trained in one domain would achieve similar performance in a different domain.

One such technique is Domain Randomization. Instead of training the model in a single simulated environment, different parameters of the simulator are randomized to expose the model to a wide range of environments at training time. With enough variability, the real world may appear to the model as just another variation of the simulator. This way the model will learn general features that are applicable to the real world as well. The randomized variables of the simulator are usually either visual parameters (e.g. textures, lighting conditions, etc.) or physical parameters (e.g. friction coefficients, the gravitational acceleration, masses, sizes or other attributes of objects, etc.).

Several works have demonstrated that Domain Randomization can successfully solve the simulator-to-real problem. Tobin et al. (2017) uses this method for object localization on real-world images by training a neural network in a simulator with highly manipulated images. Peng et al. (2017) uses randomization of the simulator's dynamics to train a neural network which moves objects to the

assigned locations using a robotic arm. OpenAI et al. (2019) uses Automatic Domain Randomization to train a robotic arm to solve a Rubik's cube. This technique incrementally increases the applied Domain Randomization and thus the difficulty of the environment, as the model learns to perform well in the previous environments. By performing the task in more and more difficult conditions, the model learns to generalize. As a result, the model trained in the simulator can successfully work on the physical, real-world robot.

In case of vision-based algorithms, a feasible way to perform the domain transfer is by applying Visual Domain Adaptation. The aim of this technique is to transfer the observations from the training and testing domains to a common domain, which is then used to train the agent to perform the given task.

Due to recent advances in image-to-image translation, this approach is becoming more and more popular. Bewley et al. (2018) uses Visual Domain Adaptation to train a self-driving agent that achieves equally good performance in both the simulated and real-world domains. Their model uses an Unsupervised Image-to-Image Translation Network (Liu et al., 2017) to translate images between the real-world operating domain and the generated simulation environment, while also learns to predict control decisions from the ground truth labels from the simulator.

In this work, several experiments were carried out in the Duckietown (Paull et al., 2017) simulator environment to solve the presented lane following task using Imitation Learning. After the agent's performance was satisfactory in the simulator, we applied different sim-to-real techniques to bridge the gap between the simulation and the real environment.

2 Proposed Methods

In this work, we implemented and evaluated three different Imitation Learning techniques for the self-driving task of right-lane following in the Duckietown simulator environment (Chevalier-Boisvert et al., 2018). We chose the best performing method and applied two sim-to-real methods to solve the sim-to-real transfer problem.

2.1 Imitation Learning in the simulation

The general training procedure of the IL experiments was the following. First, the expert demonstrator was rolled out in the environment to collect training data - demonstrations. Next, the agent was trained based on the IL algorithm. Finally, the agent was released in the environment and evaluated using the official Duckietown metrics.

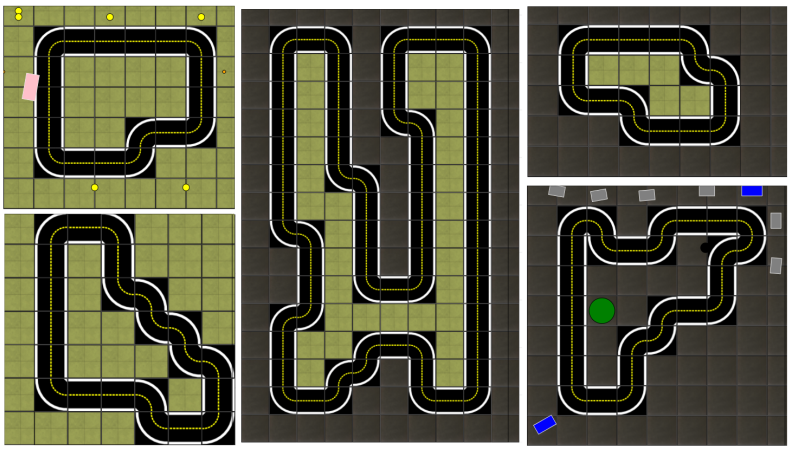

Figure 1: Training maps

The demonstrations were collected on multiple maps. When the expert completed its current trajectory, the environment was reset and a new map was randomly selected from the available set of

maps (see Figure 1. Most of the maps contain several objects on the side of the roads to increase variability. This multi-map training approach further improves the model's generalization ability and provides robustness on unseen track layouts.

2.1.1 Applied Imitation Learning algorithms

During our work we experimented with three IL algorithms: Behavioral Cloning (BC) (Bain & Sammut, 1999), Dataset Aggregation (DAgger) (Ross et al., 2010) and Generative Adversarial Imitation Learning (GAIL) (Ho & Ermon, 2016).

Behavioral Cloning: is the simplest form of IL. It focuses on learning the expert's policy using Supervised Learning. Expert demonstrations are divided into state-action pairs, these pairs are treated as independent and identically distributed examples and finally, Supervised Learning is applied.

DAgger: this method assumes, that there is access to an interactive demonstrator at training time. The algorithm starts with the initial predictor policy that had been uncovered from the initial expert demonstrations using Supervised Learning. Then, the following loop is executed until the algorithm converges. In each iteration, trajectories are collected by rolling out the current policy that had been obtained in the previous iteration and the state distribution is estimated. For every state feedback is collected from the expert and using this, a new policy is trained. For the algorithm to work efficiently, it is important to use all the previous training data during the teaching, so that the agent remembers all the errors it made in the past. Therefore, DAgger trains the actual policy on all the previous training data.

GAIL: is an Inverse Reinforcement Learning (IRL) algorithm. It aims to uncover a reward function by the means of the demonstrations, which is then used to learn the policy using Reinforcement Learning. GAIL adopts the Generative Adversarial Networks (GAN) (Goodfellow et al., 2014) architecture to carry out IRL. Similarly to GANs, the GAIL architecture consists of two neural networks: the policy network (or the generator) and the discriminator. The policy network acts as the agent's policy: it receives the agent's state in the environment as an input and outputs the adequate actions. The discriminator is a binary classifier which tries to distinguish the received state-action pairs from the trajectories generated by the agent and the expert. This network can be interpreted as the cost function that provides the learning signal to the policy.

2.1.2 Expert demonstrator

The Duckietown software stack contains an implementation a pure pursuit controller. The algorithm uses the Duckiebot's relative position and orientation to the center of the right driving lane to calculate the adequate actions of Pulse Width Modulation (PWM) signals. It selects a point on the ideal driving line at a certain distance from the agent and controls the robot to move towards this point. This is demonstrated by Figure 2.

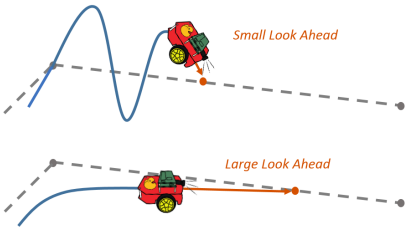

Figure 2: The operation of the pure pursuit controller.

Furthermore, we modified the pure pursuit controller to use different velocity and steering gain values for straights and for corners. We also extended the original proportional controller with a derivative gain, which managed to further improve the controller's performance.

We manually fine-tuned the pure pursuit PD controller and used it as the expert demonstrator in our Imitation Learning experiments, as this algorithm greatly outperforms a possible human demonstrator (controlling the robot with a joystick or a keyboard).

2.1.3 SIMPLIFYING OBSERVATIONS AND ACTIONS

The demonstrations are sequences of state-action pairs. In the case of the Duckietown simulator, the states are observations of the environment: images from the Duckiebot's front camera; and the actions are PWM signals that specify the Duckiebot's left and right motor velocities. To achieve better performance at training and inference, we simplified both the observations and the actions during experiments by applying a preprocessing step to the images and a postprocessing step to the actions.

Observations taken from the Duckietown simulator or from the Duckiebot are RGB images with the resolution of 480x640 (height × width). Images of this size introduce a few problems to the learning algorithms. The high-resolution results in a high-dimensional state-space, which makes it harder for the algorithm to learn a proper feature extractor. It also slows down the inference and training time of the neural network. Therefore, before feeding the images to the models, two preprocessing steps are performed in order to reduce image complexity and increase training and inference speed. These are the following:

- *Downscaling:* The images are resized to a smaller resolution of 60×80 to reduce the dimensionality of the state-space and increase training speed.
- *Normalization:* The pixel values are converted to floating-point numbers and are normalized to the $[0.0, 1.0]$ range. This is a commonly used data preprocessing method that helps the training process by alleviating numerical problems of the optimization.

In case of the GAIL algorithm, these preprocessing steps are followed by feeding the preprocessed image through feature extractor: a ResNet (He et al., 2015) network that was pretrained on the ImageNet (Deng et al., 2009) dataset.

In both simulation and the real Duckietown environments, the Duckiebots are controlled by actions of PWM signals, which represent the left and right motor velocities. However, during experiments the models are trained to predict two actions: throttle and steering angle. The throttle action is a scalar value between 0.0 and 1.0, where 0.0 causes the agent to stop and in case of 1.0 the agent moves at full speed. The steering angle action is a scalar value between -1.0 and 1.0, where -1.0 and 1.0 causes the agent to turn fully to the left and right respectively, and in case of 0.0 the agent moves in a straight line. The actions predicted by the networks are then converted to PWM signals to suit the Duckiebots.

2.2 SIM-TO-REAL METHODS

The agents trained in the simulation failed to perform the lane following task in the real Duckietown environment, as they could not generalize to the different, previously unseen real-world environment. Therefore, it was necessary to apply sim-to-real techniques to bridge the gap between the two environments. The aim of these techniques is to augment the learning process in a way that an algorithm trained in one visual domain would achieve similar performance in a different visual domain. We used two methods to solve the sim-to-real problem: Domain Randomization (DR) and Visual Domain Adaptation using Unsupervised Image-to-Image Translation Networks (Liu et al., 2017) (VDA-UNIT).

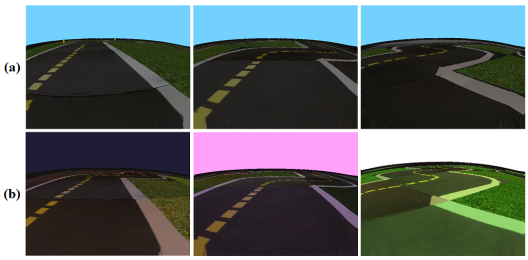

Figure 3: Observations from the standard (a) and the domain randomized (b) environment.

The Duckietown simulator has a built-in Domain Randomization functionality, which changes the parameters of the simulation (e.g. lighting conditions, textures, camera parameters, size of the robot, physical parameters, etc.) each time the simulation is reset (see Figure 3). We applied this technique during the process of collecting demonstrations, so that later the agent would be trained on domain randomized observations.

Our second solution adapts a UNIT networks to transfer the observations from the simulated X_{sim} and the real X_{real} domains into a common latent space Z. After the UNIT network is properly trained and the quality of the image-to-image translation is satisfactory, the control policy is trained from this common latent space Z using the demonstrations $\boldsymbol{c}$ from the expert in the simulation. The method is demonstrated by Figure 4.

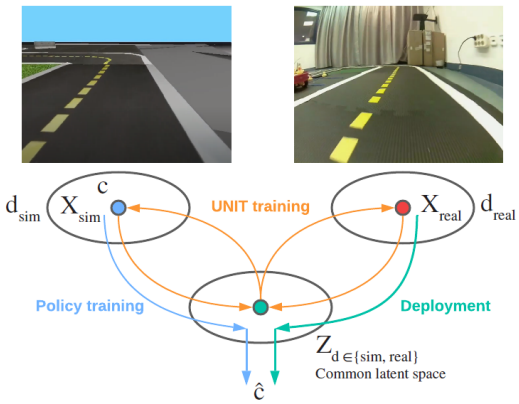

Figure 4: UNIT network-based Visual Domain Adaptation.

The main advantage of this method is that it does not require pairwise correspondences between images in the simulated and real-world training sets to perform the image-to-image translation. Furthermore, it does not require real-world labels either, the lane-following agent can be trained by using only the demonstrations from the simulation.

3 Training and Evaluation Setup

3.1 Training procedure

We have carried out following IL experiments: BC with DR, BC without DR, DAgger with DR, DAgger without DR and GAIL with BC-based pretraining. We also trained the Duckietown software stacks' DAgger algorithm as a baseline solution.

For the BC experiments and the GAIL pretraining phase 98304 demonstrations were collected (128 episodes × 768 timesteps). Throughout the DAgger experiments, the agent was rolled out for additional 128 episodes, for 512 timesteps per each episode. The acquired 65536 demonstrations were annotated by the expert and combined with the initial demonstrations, which resulted in 163840 training examples. The collected demonstrations were randomly shuffled and split into training and validation datasets using 80% and 20% of the training data.

The training of the BC and the DAgger algorithms was performed using early stopping with patience set to 25 epochs.

The training of the GAIL method started by pretraining the policy network using BC. After this, the entire GAIL algorithm was trained for 30 epochs. In each epoch, the agent was rolled out in the environment for 15 times, each trajectory consisted of 256 timesteps. The replay buffer could store 75 trajectories from the agents, which is 19200 observation-action pairs.

The models were trained with the Adam (Kingma & Ba, 2015) optimizer with the learning rate set to 0.0001. The batch size was set to 32. The Duckietown DAgger baseline was trained for 50 epochs with the default parameters.

The training of the VDA-UNIT sim-to-real method was conducted in two steps.

As the first step, we trained the UNIT network, which was responsible for the image-to-image translation between the simulated and real images. For each environment, 1024 images were randomly sampled from the datasets of over 30000 images each. In the case of the real domain, the images were extracted from video feeds of real robots, while the simulated images were simply generated by running an agent in the simulation. The test datasets were set up similarly, by randomly sampling 256 images from each dataset (excluding those images that were already sampled for the training sets). Next, the model was trained for 200 epochs. We used the Adam optimizer with the learning rate set to 0.0001. After 100 epochs, a linear learning rate decay was applied. The batch size was set to 4.

As the second step, we selected the best performing IL algorithm (DAgger) as the controller's policy and we trained it using the training procedure described earlier.

All of the experiments were ran on a single NVIDIA GeForce RTX 2060 GPU.

3.2 Evaluation procedure

To evaluate our algorithms in the simulation, we used the official Duckietown metrics. For the evaluation in the real-world environment, we defined custom metrics that could be easily measured. The metrics are presented in sections 3.2.1 and 3.2.2 respectively.

The Duckietown software environment provides an evaluation interface, which deploys the given submission in the simulation, measures its performance by calculating the official performance metrics and creates a final report that contains all the results. The evaluation procedure runs the submission for 5 episodes in the environment, which means that the robot starts from a random position and operates for a fixed amount of time. The median values of the metrics are calculated from the results of these 5 runs. We used the official evaluation tool to evaluate our models in the simulation.

To evaluate the real-world algorithms the following procedure was used. For each model, two episodes were run, during which the robot was started from once in the inner and once in the outer loop. Each episode lasted for maximum of 60 seconds. If the robot left the track, the episode was terminated. In each episode the custom metrics were measured. Finally, the metrics during the 2 episodes were averaged.

3.2.1 Performance metrics in the simulation

To evaluate our IL models in the simulation, we have used the four official Duckietown metrics. These are the following:

- *Traveled distance:* This is the median distance traveled, along a lane. (That is, going in circles will not make this metric increase.) This is discretized to tiles. This metric only measures the distance that was travelled continuously (without cease) in the right driving lane. This metric encourages both faster driving as well as algorithms with lower latency.
- *Survival time:* This is the median survival time. The simulation is terminated when the robot goes outside of the road or it crashes with an obstacle.
- *Lateral deviation:* This is the median lateral deviation from the center line. This objective encourages "comfortable" driving solutions by penalizing large angular deviations from the forward lane direction to achieve smoother driving.
- *Major infractions:* This is the median of the time spent outside of the drivable zones. This objective means to penalize "illegal" driving behavior, for example driving in the wrong lane.

3.2.2 Performance metrics in the real-world environment

During the AI Driving Olympics (Zilly et al., 2019) competitions, it is possible for the organizers to calculate the official Duckietown metrics, as the Duckietown tracks, where the submissions are evaluated, are equipped with complex positioning systems consisting of cameras, markers and precise computer vision algorithms. However, without access to the AIDO real-world evaluation system,

these metrics are impossible to calculate, as the Duckiebots are not equipped with any positioning system or sensor. Therefore, we defined two custom real-world performance metrics that can be feasibly measured without such system:

- *Survival time:* The time until the robot left the track or the time of the evaluation procedure (if the robot did not make a mistake).
- *Visited road tiles (in the correct driving lane):* The number of visited road tiles during the evaluation procedure. Only those tiles are counted, where the robot traveled inside in the right driving lane. Measuring the traveled distance of the robot is problematic, but counting the tiles instead is quite simple, therefore this is a feasible alternative.

3.2.3 BASELINE ALGORITHMS

The Duckietown software stack contains different baseline solutions for the challenges [1]. One of the Imitation Learning baselines is a DAgger algorithm, which has a training procedure that is similar to our implementation. We trained this model with the default parameters, based on the instructions that were provided in the authors' description. We used the resulted model as a baseline to measure and compare the performance of our algorithms in the simulation.

The baseline for the real-world experiments was a model trained using the best performing IL algorithm (DAgger) without applying any of the sim-to-real techniques.

4 RESULTS

4.1 RESULTS IN THE SIMULATION ENVIRONMENT

The models were evaluated with the Duckietown evaluator tool using the AIDO performance metrics. Table 1 presents the best results for each training algorithm.

Table 1: Evaluation in the simulation

IL method	Survival Time	Traveled Distance	Lateral deviation	Major Infractions
BC				
w/ DR	15	5.26	0.65	0.23
w/o DR	15	5.44	0.75	0.63
DAgger				
w/ DR	15	5.32	0.71	**0**
w/o DR	15	**5.67**	0.63	**0**
GAIL				
w/ DR	13.55	4.78	0.71	1.27
BASELINE	15	3.97	**0.35**	**0**

All three algorithms managed to train a reasonably well performing model. The agents were able to follow the right driving lane, without committing any crucial mistakes such as leaving the road.

The algorithms managed to outperform the baseline model in terms of the traveled distance and survival time (except GAIL). The baseline, however, has a significantly lower lateral deviation. This is due to the fact that the baseline agent moves a lot slower than the trained agents.

As we can see, the GAIL algorithm has a slightly worse performance compared to DAgger and BC. The reason for this phenomenon might be complexity of the training procedure: the parameters of the training process are probably not well chosen. Therefore, further optimization is needed for the GAIL algorithm.

4.2 RESULTS IN THE REAL ENVIRONMENT

The models were evaluated based on the procedure described in section 3.2 using the custom real-world metrics. Table 2 presents the best results for each training algorithm. It is important to note, that the robot's driving speed was fine-tuned for each algorithm in order to maximize the survival time metric.

[1] `https://docs.duckietown.org/daffy/AIDO/out/embodied_strategies.html`

Table 2: Sim-to-real experiments

Sim-to-Real method	**Average survival time**	**Average visited road tiles**
DR	60	20
VDA-UNIT	60	18.5
DAgger w/o sim2real	*1*	*4.5*

Both methods managed to successfully solve the sim-to-real problem, as the real-world robots could properly follow the right driving lane, without committing crucial mistakes. It is also straightforward, that in the real environment these techniques are not only useful, but necessary, as the model without any form of Transfer Learning completely failed at the lane-following task.

4.2.1 THE QUALITY OF THE IMAGE-TO-IMAGE TRANSLATION

As it can be observed on Figure 4.2.1, the UNIT network achieves high image-to-image translation quality. The network managed to learn how to remove the background that is above the horizon and replace it with the blue sky when performing the real-to-sim translation. This is also true for the other way around: during the sim-to-real translation, the network removes the sky and replaces it with background objects.

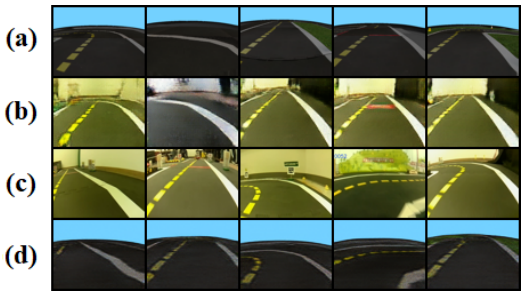

Figure 5: a) simulation b) sim-to-real translation c) real d) real-to-sim translation

5 CONCLUSION AND FUTURE WORK

In this paper, we used Imitation Learning techniques to solve a complex self-driving robotics task in the Duckietown environment. We trained the models in the simulator and applied sim-to-real methods to ensure that the algorithms achieve equally good performance in the real-world environment. We evaluated the performance of the models in both environments using the Duckietown metrics. We showed that using our approach, trained agents were able to follow the right driving lane in both the simulated and real-world domains.

Our results demonstrate that it is favorable to use DAgger as it achieves the best performance with slightly more training time compared to BC. It is challenging to reach good performance with GAIL, as the training times are fairly longer and the hyperparameters are difficult to fine-tune.

In the future, we would like to continue fine-tuning the presented solutions in a hope of achieving even better results. Performing further optimizations on the GAIL algorithm should also be advantageous, as this model was the one with the poorest performance. In addition to this, we plan to apply Curriculum Learning to solve the more complex Duckietown challenges. The source code of our work will be published after acceptance of our paper.

REFERENCES

Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15, Intelligent Agents [St. Catherine's College, Oxford, July 1995]*, pp. 103–129, GBR, 1999. Oxford University. ISBN 0198538677.

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. *CoRR*, abs/1812.03823, 2018. URL http://arxiv.org/abs/1812.03823.

Maxime Chevalier-Boisvert, Florian Golemo, Yanjun Cao, Bhairav Mehta, and Liam Paull. Duckietown environments for openai gym. https://github.com/duckietown/gym-duckietown, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016. URL http://arxiv.org/abs/1606.03476.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Yunzhu Li, Jiaming Song, and Stefano Ermon. Inferring the latent structure of human decision-making from raw visual inputs. *CoRR*, abs/1703.08840, 2017. URL http://arxiv.org/abs/1703.08840.

Ming-Yu Liu, Thomas M. Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. *CoRR*, abs/1703.00848, 2017. URL http://arxiv.org/abs/1703.00848.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019. URL http://arxiv.org/abs/1910.07113.

Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, Daniel Hoehener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, Christopher Carr, Maria Zuber, Sertac Karaman, Emilio Frazzoli, Domitilla Del Vecchio, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. Duckietown: An open, inexpensive and flexible platform for autonomy education and research. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1497–1504, 2017. doi: 10.1109/ICRA.2017.7989179.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017. URL http://arxiv.org/abs/1710.06537.

Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky (ed.), *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010. URL http://arxiv.org/abs/1011.0686.

Joshua 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. *CoRR*, abs/1703.06907, 2017. URL http://arxiv.org/abs/1703.06907.

Julian G. Zilly, Jacopo Tani, Breandan Considine, Bhairav Mehta, Andrea F. Daniele, Manfred Diaz, Gianmarco Bernasconi, Claudio Ruch, Jan Hakenberg, Florian Golemo, A. Kirsten Bowser, Matthew R. Walter, Ruslan Hristov, Sunil Mallya, Emilio Frazzoli, Andrea Censi, and Liam Paull. The AI driving olympics at neurips 2018. *CoRR*, abs/1903.02503, 2019. URL http://arxiv.org/abs/1903.02503.