Data-Driven Policy Transfer With Imprecise Perception Simulation

Martin Pecka, Karel Zimmermann, Matěj Petrlík, and Tomáš Svoboda

***Abstract*—This letter presents a complete pipeline for learning continuous motion control policies for a mobile robot when only a nondifferentiable physics simulator of robot–terrain interactions is available. The multimodal state estimation of the robot is also complex and difficult to simulate, so we simultaneously learn a generative model which refines simulator outputs. We propose a *coarse-to-fine* learning paradigm, where the coarse motion planning is alternated with guided learning and policy transfer to the real robot. The policy is jointly optimized with the generative model. We evaluate the method on a real-world platform.**

***Index Terms*—Learning from demonstration, learning and adaptive systems, reactive and sensor-based planning, domain transfer.**

I. Introduction

HIGH-DIMENSIONAL reactive motion control of complex unmanned ground robots which substantially interact with unstructured terrain is complicated. Main difficulties are threefold: (i) the sample inefficiency and local optimality of state-of-the-art reinforcement learning methods make direct policy optimization on a real platform inconceivable, (ii) the curse of dimensionality of planning methods [1] makes direct search prohibitively time-consuming, and (iii) the simulation inaccuracy of robot–terrain interactions often makes direct usage of simulator-learned policies impossible [2]. We propose a complete policy learning–planning–transfer loop, which addresses all of these issues simultaneously.

The aim of this work is to learn motion control policy for four independently articulated flippers of a tracked skid-steering robot shown in Figure 2. The proposed method exploits an analytically non-differentiable dynamics-engine–based simulator of the real platform [3]. The learned policy maps the local height map and pose of the robot to desired motion of the flippers, which assures smooth traversal over complex unstructured terrain.

The complexity of track–terrain interactions [3] slows the simulation speed down to real-time, therefore collecting a huge number of samples needed for accurate learning is impossible. Consequently, we propose coarse-to-fine policy learning, where the coarse motion planning is alternated with guided learning and policy transfer to the real robot.

The proposed method starts by planning trajectories, which approximately optimize traversal of randomly generated terrains. Then guided learning provides a coarse initial policy. Since it is impossible to simulate the state estimation described in Section IV accurately, the state estimated on the real platform significantly differs from the simulated state. Instead of precise simulation, we suggest learning a conditional generative model of the state estimation procedure, which comprises both the underlying noise of different sensors and the errors caused by fusion of multi-modal measurements. This generative model is optimized together with the policy. In addition to that, the successively learned policy allows to guide the node expansion during planning which helps to obtain more accurate plans faster. This procedure is iterated until convergence.

Contribution of the letter lies in proposing the new self-contained learning-planning-transfer loop which simultaneously learns and transfers the policy using the generative model, which refines imprecise perception in simulation. The method is evaluated on a real platform.

II. Related Work

Direct policy transfer methods: Oßwald *et al.* [4] demonstrated direct transfer of motion navigation policy for Nao humanoid robot. Policy was learned in a precise simulator and then directly used on the real platform and it performed well. Christiano *et al.* [5] suggest learning an inverse dynamics model that can adjust actions from the simulator to execute in the real world as intended. They however require a way to transfer the real-world state into the simulator to execute their algorithm. Nemec *et al.* [6] used value function learned in simulation to bootstrap the real robot learning. We also initialize the policy from the simulator.

Model-based reinforcement learning methods learn simultaneously model and the policy. Since the model learned from the scratch on real trajectories is typically a fast differentiable function [7], [8], direct policy optimization is

Manuscript received February 24, 2018; accepted July 11, 2018. Date of publication July 20, 2018; date of current version August 8, 2018. This letter was recommended for publication by Associate Editor Prof. S. Oh and Editor Prof. D. Lee upon evaluation of the reviewers' comments. This work was supported in part by the European Union under Grant FP7-ICT-609763 TRADR, in part by the Czech Science Foundation under Project GA14-13876S, in part by OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics", and in part by the Grant Agency of the CTU Prague under Project SGS18/138/OHK3/2T/13. *(Corresponding author: Martin Pecka.)*

The authors are with the Faculty of Electrical Engineering, Department of Cybernetics, Czech Institute of Informatics Robotics and Cybernetics, Czech Technical University in Prague, Prague 12135, Czech Republic (e-mail: peckama2@fel.cvut.cz; zimmerk@cmp.felk.cvut.cz; petrlmat@fel.cvut.cz; svobodat@fel.cvut.cz).

This letter has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. The Supplementary Materials contain two videos. This material is 28 MB in size.

Digital Object Identifier 10.1109/LRA.2018.2857927

often possible. But learning the motion and perception model from real trajectories (i) endangers the robot and (ii) requires prohibitively high number of trajectories. In contrast to these approaches, we already make use of a sophisticated motion model, and mainly focus on the perception transfer.

Data-driven refinement of perception simulator: The problem of transferring perception between different domains is well studied. In computer vision Generative Adversarial Nets [9] (GANs) have been recently used for generating synthetic training images. Shrivastava *et al.* [10] have shown significant performance boost if GANs are used to refine graphics-engine–based images. Similarly, we also refine simulator-generated data.

Guided policy search: In Guided Policy Search [11], guiding samples are utilized in a loop to guide direct policy search into areas of search space which yield the highest reward. However, it does not account for the reality gap between the simulated and real world, and it is impossible to run the algorithm directly on the real platform, since it requires too many samples.

A similar approach to our pipeline was tested by Bousmalis *et al.* [12] for grasping. They use (non-cycle) GAN to transform mostly static simulated images into the real domain, and then a deep network that benefits from the simulated data. In this work we show that using CycleGAN helps the domain transfer even more.

III. Pipeline Overview

Our pipeline follows three main assumptions: (i) the physics-based simulator is slow and analytically non-differentiable, (ii) simulation of the exteroceptive perception such as mapping from multi-modal sensor fusion is not realistic, and (iii) there exists an unknown generative model G which corrects the simulated perception to be close to the real perception. Under these assumptions, we search for control policy π^*, which minimizes the expected sum of traversal costs c of the real robot.

Let us denote p_r^π the probability distribution of trajectories $\tau_r = \{(\mathbf{x}_r^i, \mathbf{a}_r^i)\}_i$ generated by the real robot under policy π, and $p_s^\pi(G)$ the probability distribution of trajectories τ_s generated by the simulator with generative model G under policy π. Each trajectory $(\mathbf{x}^i, \mathbf{a}^i)$ is a sequence of state vectors $\mathbf{x}^i$ and action vectors $\mathbf{a}^i$. We search for policy

$$\pi^* = \arg\min_\pi \mathbb{E}_{\tau_r \sim p_r^\pi}\{c(\tau_r)\}. \tag{1}$$

Using assumption (iii), we rewrite the optimization problem using the simulator distribution $p_s^\pi(G)$ in the objective as follows

$$\arg\min_{\pi,G} \{\mathbb{E}_{\tau_s \sim p_s^\pi(G)}\{c(\tau_s)\} \mid \text{s.t. } p_s^\pi(G) = p_r^\pi\}. \tag{2}$$

Since trajectories collected with the simulator and with the real robot are unpaired, direct supervised training of the generative model is impossible. Consequently, we replace constraint $p_s^\pi(G) = p_r^\pi$ by the saddle point constraint on GAN-like loss $\mathcal{L}_{\mathrm{GAN}}(G, D, \pi)$ induced under policy π

$$\arg\min_{\pi,G} \mathbb{E}_{\tau_s \sim p_s^\pi(G)}\{c(\tau_s)\} \quad \text{s.t. } G = \arg\min_{G'} \max_D \mathcal{L}_{\mathrm{GAN}}(G', D, \pi), \tag{3}$$

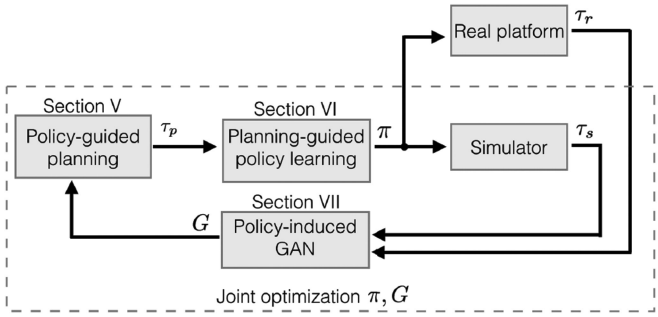

Fig. 1. **Proposed coarse-to-fine policy learning paradigm:** the coarse policy-guided motion planning is alternated with guided learning and policy transfer to the real robot.

where D denotes a discriminator.

If the GAN loss $\mathcal{L}_{\mathrm{GAN}}(G, D, \tau_r, \tau_s)$ is pure GAN loss [9]

$$\mathbb{E}_{\tau_r \sim p_r^\pi} \log D(\tau_r) + \mathbb{E}_{\tau_s \sim p_s^\pi(G)} \log(1 - D(G(\tau_s))),$$

the saddle-point generator provides samples from the true distribution and the equivalence between eq. (2) and eq. (3) holds. In order to achieve fast convergence on the high-dimensional unpaired data, we use CycleGAN loss [13], therefore eq. (3) is an approximation of the original problem.

By assumption (i), any direct optimization of eq (3) is technically intractable. We propose approximated optimization scheme, which minimizes the interaction with the slow simulator and the real robot.

The optimization alternates between (i) planning guiding samples τ_p, which approximately optimize objective

$$\arg\min_{\tau_p'} \mathbb{E}_{\tau_p'}\{c(\tau_s)\}, \tag{4}$$

(ii) collecting real and simulated trajectories τ_r, τ_s, and (iii) searching for the control policy and the generative model which minimize the locally approximated criterion

$$J(\pi, G, \tau_p) = \sum_{(\mathbf{x},\mathbf{a}) \in \tau_p} \|\pi(G(\mathbf{x})) - \mathbf{a}\| \tag{5}$$

subject to locally approximated GAN loss $\mathcal{L}_{\mathrm{GAN}}(G, D, \tau_r, \tau_s)$ around the collected trajectories τ_r, τ_s. The proposed pipeline is summarized in Figure 1 and Algorithm 1.

The generative model G^0 is initialized as identity. The initial policy π^0 is initialized by guided learning (i.e.,we plan initial trajectories τ_p and estimate $\pi^0 = \arg\min_\pi J(\pi, G^0, \tau_p)$). Given the initial policy, real trajectories are collected and alternated optimization (lines 3–8) with K iterations is performed. Finally, a new set of real test trajectories is collected and the whole process is repeated until a satisfactory behavior of the real robot is observed.

IV. Real Platform and Its Simulation Model

The real robot used in our experiments is the Absolem tracked vehicle used in Urban Search and Rescue scenarios [3], [14], which is depicted in Figure 3. It is equipped with a gyro providing its spatial orientation and with a rotating 2D lidar which provides full 3D laser scans at rate 0.3 Hz. The point map built from

Fig. 2. **Robot surmounting unstructured terrain during USAR mission.**

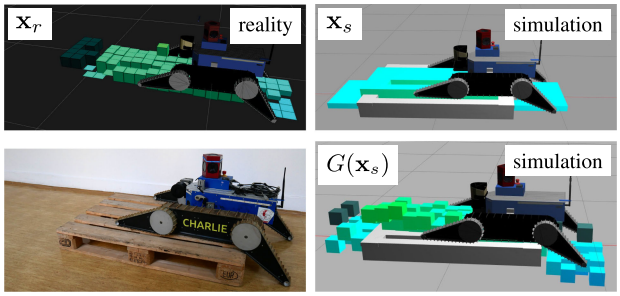

Fig. 3. **Real and simulated DEMs.** A visualization of Digital Elevation Maps (DEMs) is shown above. Dark green cells represent NaNs. **Top left:** DEM captured by the real platform. **Bottom left:** The real pose of the robot on an obstacle. **Top right:** DEM from simulator. The shapes are ideal and all measurements are available. **Bottom right:** DEM from simulator transformed by G to appear realistic.

lidar scans by the state-of-the-art SegMatch algorithm [15] is combined with high-precision track odometry in a multi-modal fusion pipeline [16].

For simulation, we use our custom tracked vehicle dynamics model implemented in the Gazebo simulator [3]. Parts of the simulation are randomized or pseudo-randomized (e.g., search of contact points of colliding bodies, solving of the underlying dynamics equations), so every execution of even a deterministic policy results in slightly different outcomes. This is useful for us, because our pipeline requires a multitude of different trajectories for every control policy. To achieve fast simulation, several simplifications were implemented in the simulated perception pipeline.

The most important of all policy inputs is the *Digital Elevation Map* (DEM) of close robot neighborhood (visualized in Figure 3). It is a horizontal 2D grid of rectangular cells where each cell contains information about the highest 3D point located in it. When there is no point measured inside a cell, a *Not-a-number* (NaN) value is stored. The DEM is treated in the coordinate frame of the robot with pitch and roll angles zeroed out. On the real robot, DEM is constructed from the point map. In simulation, DEM measurement is done in a completely different way to avoid inefficient laser ray-tracing: we directly extract the height of the highest object (excluding robot body) in each DEM cell, which is a fast operation. That means there are no missing measurements in the simulator DEM, and also no noise.

V. Generating Guiding Plans

The simulator is utilized by the path planner to sample trajectories τ_p^k, which are further used in the pipeline as described in Algorithm 1.

Algorithm 1: Overview of the Real Policy Learning.

1: **Initialize:** G^0 as identity and policy π^0.
2: **Collect** real trajectories $\tau_r \sim p_r^{\pi^0}$
3: **for** $k = 0 \dots K$ **do**
4: **Plan** guiding traj. τ_p^k biased by π^k (Section V).
5: **Optimize** policy w.r.t. new generator (Section VI)

$$\pi^{k+1} \leftarrow \arg\min_{\pi} J(\pi, G^k, \tau_p)$$

6: **Collect** simulated trajectories: $\tau_s^{k+1} \sim p_s^{\pi^{k+1}}(G^k)$
7: **Find** trajectory-consistent saddle point (Section VII)

$$G^{k+1} \leftarrow \arg\min_{G} \max_{D} \mathcal{L}_{\mathrm{GAN}}(G, D, \tau_r, \tau_{s^{k+1}})$$

8: **end for**
9: $G^0 \leftarrow G^K$, $\pi^0 \leftarrow \pi^K$ and **repeat** from line 2.

The planner works on a multitude of randomly generated worlds (*training worlds*) with different obstacles, corresponding approximately to the expected real obstacles. Each training world has a predefined length of trajectories the robot has to *safely* traverse to consider the trajectories *valid* (a time limit is also in place).

Different definitions of valid trajectories can be used; they are always closely related to the particular task. We utilize the fact that if the flippers are controlled incorrectly, the robot is not able to overcome obstacles and gets stuck or damaged. Safety of trajectories is given implicitly by several criteria like maximum allowed accelerations, limits on pitch and roll angles, and parts of the robot body which cannot touch any part of the environment.

Input of the planner consists of the training world specification and possibly also a *guiding policy* π. The task is to find a *valid* trajectory τ while keeping planned actions as close to actions of π as possible (if π is given).

The planner uses an RRT-based algorithm of state space search. Planning nodes capture the simulated DEM, robot orientation and flipper configuration. Each expansion of a planning node is evaluated in the simulator and a new planning node is created for the returned state.

Even though a standard RRT planner can find a solution by exploring the state space uniformly in all dimensions, in reality it is often impractically slow. In high-dimensional applications with costly expansion (as in our case), a heuristic must be employed to reduce the required iterations. Kinodynamic RRT* [17] is widely used to compute asymptotically optimal trajectories for robots with linear differential constraints. The method, however, assumes the knowledge of explicit motion model. Another general approach is to first find a discrete geometric path in a simplified search space and then optimize it by generating multiple trajectories with added noise [18] or by biasing the sampling of a guided RRT planner [1], [19], which is the method we use. A whole set of (different) trajectories is expected to satisfy our validity criterion, so methods targeting at getting close to a single optimal trajectory are not suitable.

Policy π is used as a guide by sorting the actions by their similarity to what π would do (we use L_2 norm, but any meaningful norm can be used). If π is not given, actions are selected randomly. Node expansion is realized by executing the action in simulator and checking the feasibility of the obtained node. The tree cannot be optimized by RRT* rewiring [20], due to the uncertainty introduced by executing an action, which prevents connecting any two nodes of the tree. Trajectories generated from the guided RRT are similar to trajectories sampled from the guiding policy, but many sampled trajectories can be invalid, and using the planner filters these automatically out.

An important property of the guiding approach is that with more planning–learning iterations, the plans will be closer to the subspace representable by the chosen policy class, which should in return result in better fit of future policies to future planned paths. The speedup gained by the guiding is utilized to enlarge the searched action space or refine the time resolution.

We propose to start the planning in a reduced action space which is practical to be explored without guiding, and once a guiding policy is available, the dimensionality can be increased. We start with 9 actions and time resolution of 1000 ms, further we add more actions, and last, we refine the time resolution to 200 ms, which is more suitable for real-world execution (but the plans need to be 5-times longer, which would be a significant increase in computation time without guiding).

VI. Guided Learning

With a set of trajectories generated by the path planner, the guided learning phase can start. Generally, it is possible to use any kind of supervised learning in this part. We chose a deep neural network that is crafted to make use both of the 2D structure of DEMs and to handle correctly *Not-a-Number* (NaN) values.

Inputs to the network are DEM, orientation of the robot and current flipper positions. Outputs of the network are the 4 desired flipper positions. Normally, if a NaN value would enter as a part of the DEM, it would silently spread further and could eventually end up in one of the outputs, which is undesirable.

A standard approach is to replace NaNs with a neutral value (like 0) or interpolate them. In Section VIII we show that these approaches yield worse results. Thus, we decided to treat the NaN values as "first-class citizen" because they can also carry useful information (the fact that a measurement is missing can have geometrical reasons).

We propose the following input processing: the DEM is converted into two matrices of the same shape—one with NaNs replaced by zeros, and the other with ones in measured cells and zeros in cells with NaNs (this part of architecture is shared with the GANs described in Section VII). Each of these matrices is fed into its own convolutional layer, and their outputs are multiplied. This effectively means normalizing each patch covered by a convolutional filter by the number of measured values in this patch. From this layer on, no NaN values are in the network, the output of the convolution is flattened, concatenated with the 1D inputs (robot orientation, flipper angles) and finally enters a fully connected layer, whose output are the four desired flipper angles.

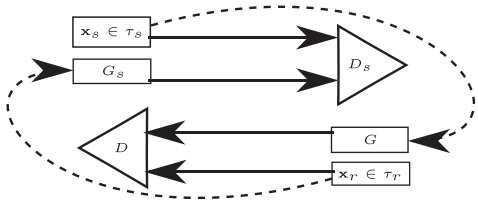

Fig. 4. **CycleGAN architecture.** Two GAN networks interconnected in such a way that input dimension of generator G_s is the same as output dimension of G and vice versa. The discriminators D_s and D serve both for evaluation of single generator loss and the cyclic loss.

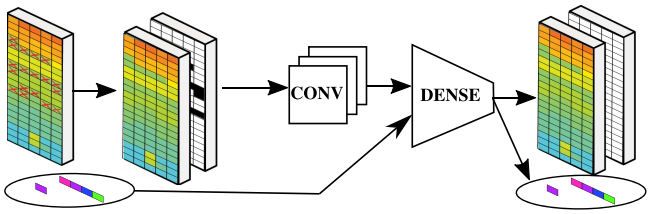

Fig. 5. **Generator architecture.** The raw input is preprocessed to yield a tensor of shape $21 \times 5 \times 2$ which is then used by the rest of the network.

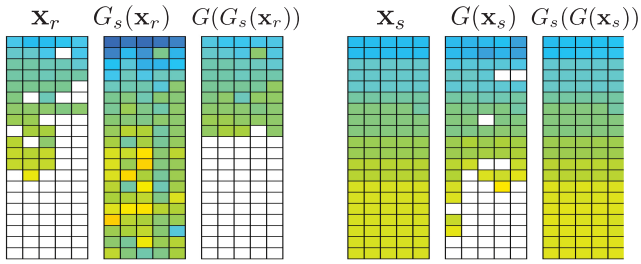

Fig. 6. **DEMs transformed by the generators.** Heights in the DEM: blue $= -1$ m, green $= 0$ m, red $= +1$ m, white $=$ NaN.

The regressor network is optimized using gradient descent to minimize the error between the predicted flipper target positions and those provided in the dataset. The dataset is randomly divided into training and test parts.

VII. Data Transformation via CGANs

The next key step is to find a suitable transformation between the data observed on the real platform and data observed in the simulator.

CycleGANs [13] were shown to be useful in the task of mutual mapping of two domains when only unpaired data are available. Specifically, Shrivastava *et al.* [10] used them to transform a simulated dataset to look real and then applied standard deep learning that expects real data at the inputs.

The mapping from simulated to real data is realized by generator G, while the opposite process is represented by generator G_s. The relation between the generators, their discriminators and input datasets is shown in Figure 4.

The input data with special structure ($20 \times 5\, 2D$ data possibly containing NaNs + 5 scalar constants), are preprocessed similar to Section VI. In generators and discriminators, the input DEM is transformed into a $20 \times 5 \times 2$ tensor where the first channel

contains the DEM with *NaNs* substituted with 0 s and the second channel contains a mask with -1 s at *NaN* cells in the DEM, and 1 s otherwise see Figures 5 and 6 for details.

The scalar inputs (robot orientation and flipper angles) skip these first convolution layers and enter the network later as inputs to a fully connected layer. At the output, the DEM and the scalar values are again separated. This allows the network to work as a standard image-to-image CycleGAN, but also allows it to use the scalar information.

The internal structure of the generators and discriminators contains several convolution layers that use the Leaky ReLU activation function, and a final fully-connected layer.

Our pipeline suggests that the generators should be initialized to identity, which is not generally possible with neural networks containing non-linear activation functions. However, implementing a skip-connection of the input data directly to the fully-connected layer allows this initialization. Identity should be a good initial guess for the generator, because we do not want it to change the data too much.

Both discriminators use the pure GAN loss formulation (see Section III).

Loss function of both generators is defined by their corresponding discriminator (D for generator G; D_s for generator G_s):

$$\mathcal{L}_G(\mathbf{x}) = +\lambda \cdot \sum(\log(D(\mathbf{x})) + \lambda_p \cdot \sum_i ||\mathbf{x}_i - G(\mathbf{x}_i)||$$

We penalize distance of the generated output from the inputs (pixel-wise), as it was shown to stabilize the learning [10]. One additional component of $\mathcal{L}_{G_s}$ can be added that penalizes any NaN values in the output, since we know there are no NaNs in the simulator DEMs.

The cycle loss $\mathcal{L}_c(G, D, \mathbf{x}_r, G_s, D_s, \mathbf{x}_s)$ is defined as

$$\mathcal{L}_D(G(G_s(\mathbf{x}_r))) + \mathcal{L}_{D_s}(G_s(G(\mathbf{x}_s)))$$

Training of the network is done by repeated optimization of all generator and discriminator losses, where $\lambda_c \cdot \mathcal{L}_{cycle}$ is added to the loss of both generators. The training is done on simulated data from τ_s^k and real data from τ_r.

It is usually difficult to tell when to stop GAN training. Although it is not required for the training itself, we constructed a small validation dataset consisting of pairs of data from the simulator and their closest counterparts encountered in the real data. If it is possible to collect more such correspondences, a part of them can be added to the learning process via L_2 loss on these samples. In our tests, adding the correspondences further helped training the GAN, but care must be taken to not overfit the network to the correspondences.

VIII. Experiments

Experimental evaluation of the learned policies is an essential part of the learning loop. After several iterations of the learning, planning and generator optimization, verification in the real world is to be performed.

For the task of terrain traversal with a tracked robot, we designed a real test scenario consisting of flat ground, a pallet and a staircase, which are typical obstacles the robot can encounter. The staircase is subdivided to 6 sections with different characteristics – approach to stairs, on stairs, leaving stairs, and the stairs can go either upwards or downwards. The staircase is traversed with constant forward speed 0.3 m/s three times and the pallet 10 times, resulting in execution of 13 trajectories. Every trajectory is assigned one of three success levels – *good* in case the trajectory was without problems, the robot passed and did not endanger itself; *unclear* if there were minor problems during the execution, but the robot traversed the whole required length (e.g., behavior close to unsafe, the operator had to reduce the otherwise constant travel speed, and so on); finally *fail* level is assigned to trajectories that the robot could not finish or executed an unsafe action. These levels carry numerical value (*good* $=$ 1.0, *unclear* $=$ 0.5, *fail* $=$ 0.0) and policy performance is an average of these values over all executions.

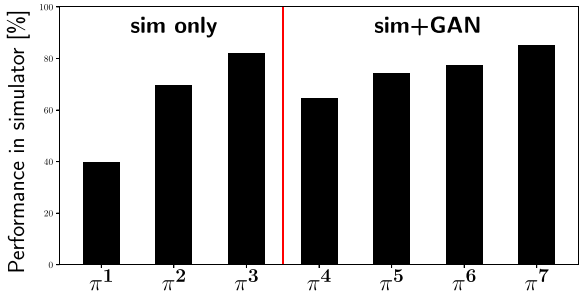

Fig. 7. **Average policy performance in simulated worlds.** Performance of 100% means traversing all test worlds in a safe manner.

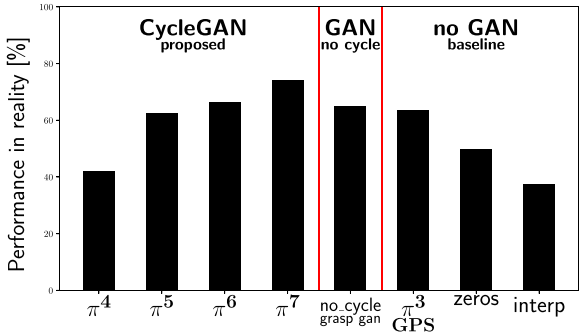

Fig. 8. **Average policy performance in real world.**

Similar obstacles were modeled in the simulator and a set of 8 test worlds was created. The metrics for simulation is proportion of *good* trajectories among all executed. Here *good* means traversing the required length of the trajectory with constant speed 0.3 m/s without executing any unsafe actions (as described in Section IV).

Results of the learning process are summarized in Figures 7 and 8. First, 3 iterations (policies π^1–π^3) were using only the simulator without GAN for adjusting perception. Further simulator-only iterations showed little performance improvement, so we assume the process converged at π^3. Policy π^3 is similar to what Guided Policy Search [11] with Adaptive Guiding Samples would find, so we also call it GPS (we use a different guiding sample generator – RRT instead of DDP, and the RRT planner automatically generates adaptive samples by prioritizing actions similar to the policy decisions).

TABLE I
PATH-PLANNING PERFORMANCE

It.	Guided	GAN	# actions	Δt	Visited nodes	Avg. CPU time
1	×	×	9	1000 ms	116 ± 60	8 min
2	✓	×	9	1000 ms	103 ± 55	5 min
3	✓	×	9	1000 ms	102 ± 54	5 min
4	✓	×	**49**	1000 ms	138 ± 82	15 min
5	✓	**✓**	49	1000 ms	238 ± 13	17 min
6	✓	✓	49	**200 ms**	1239 ± 811	40 min
7	✓	✓	49	200 ms	924 ± 497	35 min
-	×	✓	9	200 ms	-	$\gg$60 min

CPU-core–time and number of visited nodes needed to sample one trajectory by the path planner. Δt is time resolution (i.e.,with $\Delta t = 200$ a trajectory of some defined metric length needs $5\times$ more nodes than with $\Delta t = 1000$). Bold values highlight changes between iterations.

Unfortunately, real-world trajectories cannot be used as guiding samples in GPS, because the simulated and real domains differ too much for the learning to converge.

Testing in real world started in the fourth iteration. Two of the best policies found in simulator were tested in real world and the better one became π^{k+1}.

We cut off the whole pipeline once the policy achieved good performance in the real world (after 7 iterations). That accounts for ca 15 minutes of driving with the real robot to collect the initial τ_r, then 4×13 trajectories for real-world policy verification, which is about 20 minutes. No more real-world execution was needed.*

To see the benefits of our pipeline, we tested running π^3 aka GPS (the best simulator-only policy) directly on the real robot. The performance was, as expected, poor. We also trained two baseline policies ($zeros$ and $interp$) which either zero-out or bi-linearly interpolate the missing values (NaNs) in real data. These policies can have a simpler structure (the second channel for NaNs is removed). They were trained on the same trajectories π^4 was trained on. None of these policies managed to outperform the proposed pipeline. Last, we also tested the importance of the cycle loss in GANs. Policy no_cycle was trained on a dataset transformed by a GAN that was trained without the cycle loss, similar to GraspGAN [12], so we also call it $grasp\_gan$. Validation error of the GAN (as mentioned at the end of Section VII) was about 12% higher than with cycle loss, and performance of the no_cycle policy did also not beat the proposed pipeline.

To train final policy π^7 from scratch, we needed 800 CPU-core–hours (of which 90% is spent on performance verification, which could be lowered) and 50 GPU-hours (highly depends on structures of the policy and GANs).

We also experimentally verified that guiding decreases path-planning time or allows to plan paths in larger action spaces or with longer planning horizon. A summary of computation times is shown in Table I. We also tried unguided planning with 200 ms resolution, but no path was found in one hour.

*See the attached video with policy tests, or http://cmp.felk.cvut.cz/∼peckama2/policy_transfer/ for more information and FullHD video.

IX. Conclusion and Future Work

We have proposed and experimentally evaluated the new self-contained learning–planning–transfer loop, which employs a simulator of robot–terrain interactions. The proposed method simultaneously learned the policy in simulation and transferred it to the real robot. The transfer was achieved by a generative model which corrected imprecisely simulated perception. The experimental evaluation showed that iterations of the learning–planning–transfer loop improve performance of the policy on the real robot. We also showed that it is possible to further refine the action space of guiding policies without compromising computational tractability.

Our ongoing research will focus on possibilities of making the CycleGAN learning policy-aware, so that the generators are trained with policy performance in mind.

References

[1] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2006, pp. 5369–5375.

[2] J. Kober, "Learning motor skills: From algorithms to robot experiments," in, *Information Technology*. New York, NY, USA: Springer, 2014.

[3] M. Pecka, K. Zimmermann, and T. Svoboda, "Fast simulation of vehicles with non-deformable tracks," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2017, pp. 6414–6419.

[4] S. Oswald, A. Hornung, and M. Bennewitz, "Learning reliable and efficient navigation with a humanoid," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2010, pp. 2375–2380.

[5] P. Christiano *et al.*, "Transfer from simulation to real world through learning deep inverse dynamics model," arXiv:1610.03518, Oct. 2016.

[6] B. Nemec, M. Zorko, and L. Zlajpah, "Learning of a ball-in-a-cup playing robot," in *Proc. 19th Int. Workshop Robot. Alpe-Adria-Danube Region Robot.*, 2010, pp. 297–301.

[7] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," in *Proc. IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 2, pp. 408–423, Feb. 2015.

[8] R. Tedrake, "LQR-trees: Feedback motion planning on sparse randomized trees," in *Proc. Robot., Sci. Syst.*, 2010.

[9] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[10] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," *Comput. Vision and Pattern Recognition*, 2017.

[11] S. Levine and V. Koltun, "Guided policy search," in *Proc. 30th Int. Conf. Mach. Learn.*, vol. 28, no. 3, 2013, pp. 1–9.

[12] K. Bousmalis *et al.*, "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping," in *Proc. Int. Conf. on Robot. and Autom.*, Sep. 2018.

[13] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. Int. Conf. on Comput. Vision*, 2017.

[14] I. Kruijff *et al.*, "Designing, developing, and deploying systems to support human-robot teams in disaster response," *Adv. Robot.*, vol. 28, no. 23, pp. 1547–1570, 2014.

[15] R. Dube, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "SegMatch: Segment based place recognition in 3D point clouds," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 5266–5272.

[16] J. Simanek, M. Reinstein, and V. Kubelka, "Evaluation of the EKF-based estimation architectures for data fusion in mobile robots," *IEEE/ASME Trans. Mechatronics*, vol. 20, no. 2, pp. 985–990, Apr. 2015.

[17] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints," in *Proc. Int. Conf. on Robot. and Autom.*, 2013.

[18] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 4569–4574.

[19] V. Vonasek, J. Faigl, T. Krajnik, and L. Preucil, "RRT-Path: A guided rapidly exploring random tree," in *Robot Motion and Control*, Berlin, Germany: Springer, 2009, pp. 307–316.

[20] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, pp. 846–894, 2011.