DoShiCo: a Domain Shift Challenge for Control

Klaas Kelchtermans* and Tinne Tuytelaars*

***Abstract*— Training deep neural control networks end-to-end for real-world applications typically requires big demonstration datasets in the real world or big sets consisting of a large variety of realistic and closely related 3D CAD models. These real or virtual data should, moreover, have very similar characteristics to the conditions expected at test time. These stringent requirements and the time consuming data collection processes that they entail, are probably the most important impediment that keeps deep neural policies from being deployed in real-world applications. Therefore, in this work we advocate an alternative approach, where instead of avoiding any domain shift by carefully selecting the training data, the goal is to learn a policy that can cope with it. To this end, we propose a new challenge: to train a model in very basic synthetic environments, far from realistic, in a way that it can fly in more realistic environments as well as take the control decisions on real-world data. We collected a benchmark dataset and implemented a baseline method, exploiting depth prediction as an auxiliary task to help overcome the domain shift. Even though the policy is trained in very basic environments, it can learn to fly in a very different realistic simulated environment. It is even capable to compete and in some cases outperform a policy trained in the more realistic environment when testing on real-world data.**

I. Introduction

Obstacle avoidance lies at the very heart of autonomous navigation, be it for self-driving cars, robots or drone applications. While many alternative solutions exist, methods relying solely on monocular cameras are particularly interesting, especially in the context of drones. Indeed, in contrast to active sensors, such as lasers or sonars, cameras consume limited battery power, are lightweight and have an almost infinite range, making them a first class sensor. Not surprisingly, more and more research focuses on the problem of predicting control actions directly from such camera input.

In particular, deep neural network policies with high dimensional inputs (RGB), be it based on standard convolutional neural networks (CNN) or on deep reinforcement learning (DRL), are gaining interest, thanks to the recent successes obtained in this direction (e.g. [1], [2], [3], [4]). Yet so far, models have always been evaluated in environments that are very similar to the situation in which they are trained. On the one hand, there is the initial work demonstrating the potential of deep control networks using computer games, i.e. in purely simulated environments (e.g. [4], [5]). On the other hand, methods have been proposed exploiting large scale real-world datasets, gathered from the exact same - or at least very similar - environments as the ones used at test time (e.g. [6], [7]). Here, we argue that, when the goal is to learn a policy, *the need for such close similarity between train and test environments is not sustainable from a practical point of view and probably forms the largest impediment that keeps deep neural policies from being deployed in real-world applications*.

Indeed, as an illustrative example, suppose you want to build a commercial system involving a drone navigating in a hospital for assistance or surveillance. It will be hard to get a realistic model of this hospital. Nor do you want to retrain a new network for each new hospital. Therefore, this would require a huge dataset of demonstration flights captured in a large variety of hospitals and under a wide range of conditions. Apart from the cost in terms of pilot manpower for collecting such a dataset, and the nuisance caused to patients and medical personnel that comes with it, there is the issue of a *state-space shift* [8]: due to the non-i.i.d.(independent and identically distributed) nature of online flights, a compound error of the policy will bring the drone into states not present in the demonstration dataset used for training.

As an alternative, it has been proposed to use a carefully selected set of realistic virtual environments instead. [9] shows how a deep neural network policy trained on a large variety of simulated corridors can fly through real corridors. The domain shift from the simulated source domain to the real-world target domain is overcome by introducing a very large variance in the simulated environments. In the hospital setting this would require a large set of 3D CAD models of hallways and rooms. The models would have to cover a large enough variety in order to work in the real world. In [10], a photorealistic dataset is collected from computer games. Unfortunately, not all real-world applications have an open photorealistic computer game as twin. Finally, although Izadinia et al. [11] demonstrate how 3d CAD models can be extracted from real images, collecting a dataset in this variety of CAD models seems an inefficient way to deal with the problem. Moreover it is often difficult to predict what variety is required to cover the possible test environments in real-world applications.

In summary, neither the real-world demonstration data nor the carefully selected synthetic data seem viable solutions in practice. Instead of putting effort in data collection for each and every specific case, each time trying to minimize the discrepancy between train and test conditions, it is worth exploring alternative strategies, that can effectively cope with the domain shift. This requires neural policies that can generalize to previously unseen conditions, perform well over a wide action range (hence exploiting visual cues such

*The authors are with KU Leuven, ESAT-PSI, Imec, Belgium. firstname.lastname@esat.kuleuven.be

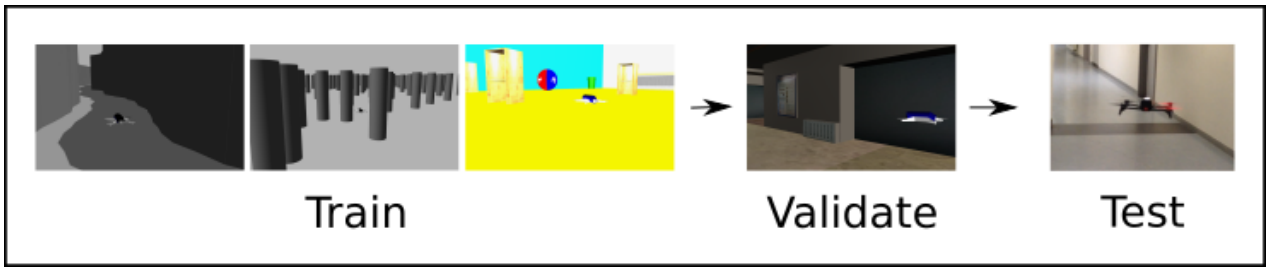

Fig. 1. DoShiCo Challenge: can a neural control network be trained on three sets of very basic simulated environments (train in "Canyon", "Forest" and "Sandbox"), so that it can fly in a more realistic environment (validate in "ESAT") as well as take control decisions on real-world data (test on "Almost-Collision dataset")?

as relative pose and depth) while at the same time being insensitive to irrelevant differences (such as color or texture), all learned in an end-to-end fashion.

Further, as a second observation, we notice that except for controlled settings like computer games, *progress in this field seems hampered by the lack of benchmarks*. The online nature of the problem, in combination with the above trend to always tune to a specific environment, makes it difficult to compare different methods and move forward as a field. This is aggravated by the naturally large variance in real-world experiments, due to external factors (different hardware, wind, delays, different environments) as well as internal ones (different initializations, different hyperparameters, etc.).

Building on the above observations, we make two main contributions. First, we propose a domain shift challenge for control systems (DoShiCo) that can serve as a benchmark for comparing different training strategies. Given three sets of very basic simulated environments, instances of which can be generated randomly during training, the goal is to come up with a model that can generalize from there to previously unseen more realistic conditions, as captured by our synthetic yet more realistic validation environment and, ultimately, the real world test data (see Figure 1). To avoid the issue with online control in a real world setting, a classification task using a dataset of 'almost collisions' is provided as a proxy for flying in the real world.

Second, we evaluate a couple of methods in this setting, building on a MobileNet [12] pretrained on ImageNet [13]. In an attempt to train the control end-to-end without loosing the robustness to varying imaging conditions and without overfitting to the basic simulated environments, we experiment with the use of an auxiliary task. In particular, we demonstrate how auxiliary depth prediction can reduce the impact of the domain shift and we show that our model succeeds in flying in the more realistic simulated validation world (ESAT) although it was trained solely on the mix of basic environments (Canyon, Forest and Sandbox). The policy trained could even perform a couple of real-world flights online, as can be seen in the supplementary material.

In order to use DoShiCo as a benchmark, we integrated the full setting of ROS, Gazebo, gpu-accelerated Tensorflow in a Docker image. Moreover, the docker image has a dummy xpra server installed so it can run remotely without the need of a graphic session. This significantly reduces the overhead required for on-policy benchmarking in 3D virtual environments. All the code, 3D environments and the best trained checkpoints of the model will be made publicly available in order to reproduce the results.

The remainder of this paper is organized as follows. First, in section II, we describe the most related work. Next, we give details on our models (Section III) and the details of DoShiCo (Section IV). In Section V, the experimental results are discussed. Section VI concludes the paper.

II. Background and Related Work

Learning to control

When it comes to autonomous navigation directly from high dimensional camera input, a first family of solutions relies on geometric techniques for simultaneously mapping and localizing the agent (SLAM). Besides the extra computational power, these algorithms often suffer from a lack of features to track [14]. In a similar spirit yet more robust, Gupta et al. [15] have recently demonstrated how a joint neural architecture, called a CMP (Cognitive Mapping and Planning), can effectively learn to map and plan jointly trained in an end-to-end fashion.

More related to our work, yet avoiding the end-to-end complexity and bypassing the domain shift issue, there are some works that rely on depth estimation as an intermediate step. Michels et al. [1] predict depth and then train a controller in simulation with reinforcement learning. In [16] depth is estimated from single images using a CNN, as in [17], and then used to avoid obstacles with a behavior arbitration algorithm [18].

On the other hand, Levine et al. [3] demonstrated that training end-to-end results in more stable and efficient learning. In fact, early work already demonstrated the promising path of collision avoidance control directly from RGB input. Pomerleau [19] successfully trained a single layer network end-to-end with apprenticeship learning for the task of road following. Lecun et al. [20] trained a 6-layered CNN to predict the steering angle of a small car based on stereo input from an offline dataset. More recent work by Ross et al. [8] trained an SVM (Support Vector Machine) iteratively with imitation learning in a forest. Giusti et al. [6] train a deep neural network end-to-end for following trails in a forest from a large offline dataset gathered manually.

Dealing with the virtual-real domain shift

All control algorithms previously mentioned are trained in an environment that is close to the test environment. However, as indicated above, for many real-world applications this is unfeasible. There have been some preliminary attempts, like [21], to cope for instance with different weather conditions. It is however often easier to train a control algorithm in simulation before testing it in the real-world. The latter entails a much bigger domain shift. In computer vision, several methods for domain adaptation have been proposed (see [22] for a recent survey), but mostly in rather artificial setups and, to the best of our knowledge, never really in the context of training neural control networks.

The work coming closest to a solution to this problem is the CAD2RL setup [9]. Here the deep neural policy is trained with reinforcement learning in a large variety of realistic, though not photorealistic, 3D CAD models of hallways. Due to the large visual variance and realistic models in simulation the policy can generalize to real hallways. However, using a large variety of realistic 3D models to close the gap between simulation and the real-world, seems an inefficient way to deal with the issue. Besides, it is often impossible to obtain such a large variety of realistic 3D models for a specific real-world application. In our work, on the other hand, the deep networks are trained off-policy from an offline dataset gathered in very basic simulated environments. The advantage of training offline is that it is much faster due to the sample efficiency.

Finally, the idea of using auxiliary tasks stems from Multi-Task Learning (MTL). In computer vision MTL has been demonstrated to improve the performance of one task by sharing the network with other tasks – for instance object detection together with classification and segmentation [23]. Mirowski et al. [24] demonstrate the use of depth prediction as an auxiliary task in order to learn a deep neural agent to navigate and localize itself in a simulated maze. The use of auxiliary tasks helps the extracted features to focus on the information that is relevant for the task which makes the learning less prone to fitting toward irrelevant features only visible in the training environments. We see in our baseline methods that the use of depth prediction has indeed a positive influence on the training.

III. Model

Here, we describe the architectures of the models used in our experiments. Then we discuss how they are regularized, with the aim to minimize the drop in performance when switching to new environments. Finally we explain how the data is collected automatically with a behavior arbitration algorithm in order to be able to train with supervised learning.

Architectures: NAUX & AUXD

The architecture of the baseline model is shown in Figure 2. The base network is called NAUX (for "No Auxiliary task") and contains a feature extracting part (yellow) and two fully connected layers for control (green). In order to give the neural network a sense of time, the network takes three consecutive frames as input. Each frame is fed to a feature extracting part with shared weights, with the architecture of mobilenet-0.25 [12]. The weights of the mobilenet-0.25 are initialized from a model pretrained on Imagenet [13]. The features are concatenated (black circle) and fed to the control prediction part (green). The control part has a fully connected hidden layer with 50 nodes with ReLu activation and an output layer with no activation function. The control output is a continuous value of the angular velocity in yaw. The control is trained in a supervised fashion using the mean squared error as loss. The targets are provided by a behavior arbitration algorithm that uses the ground truth depth in simulation as explained bellow.

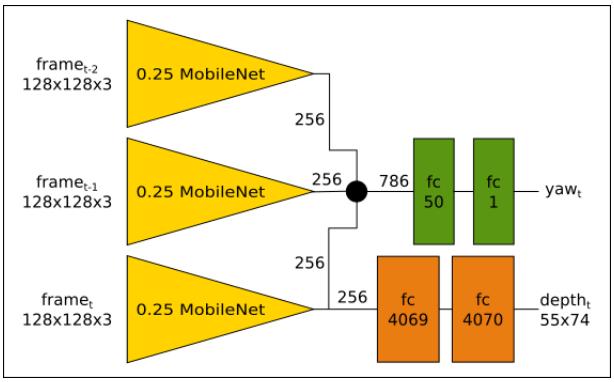

Fig. 2. The architecture for training the policy. The figure is best seen in color. The yellow parts are 0.25-MobileNets that share weights over three consecutive frames. The circle represents the concatenation of the extracted features. The green part is the control prediction that consists of two fully connected layers. The orange part represents the auxiliary depth prediction layers.

The network with auxiliary depth prediction is further referred to as AUXD. It is build on top of NAUX. The extracted features of the last frame are fed to two fully connected layers that predict a depth frame of 55x74, based on the work of Eigen et al. [17] (see Figure 2).

Regularization and stabilization

Deep neural networks are prone to fitting the extracted features towards features only relevant to the training data. This is to be avoided in our setting, as it will lead to lower performance in the validation environment as well as on the test dataset. Luckily, several robust ways to avoid such overfitting have been proposed.

The first type of regularization we use, is starting off from a *pretrained network*. The SLIM library [25] provides the weights of different models trained on Imagenet. Using these weights to initialize the network ensures that the initial extracted features are very generic and well suited for real images. In order to keep those generic Imagenet features, the gradients of the feature extracting weights are applied with a learning rate that is 10^{-3} smaller than the fully connected prediction layers.

The second modification for stabilization and regularization, is training offline on batches stochastically sampled

from a demonstration dataset. In reinforcement learning it is common to train online from the current frame or a small replay buffer [4]. But the small buffer introduces a bias towards the last experience. This bias makes it less stable to train a network and results in extra variance during training. Because of this, we train our models offline from a demonstration dataset.

Apart from the above, other regularization methods common in deep learning are applied: 0.5 dropout [26], mini-batches of size 32, and optimizing with Adadelta [27].

Data collection with Behavior Arbitration

The data is collected by flying with a simulated drone in ROS-Gazebo.A controller based on behavior arbitration navigates the drone through a CAD model in simulation. The controller utilizes true depth images taken from a simulated kinect in order to steer the drones yaw so it avoids obstacles in the horizontal plane [18]. The controller calculates a larger angular velocity according to how close and how much an object appears in the middle of the depth image.

$$\dot{\phi} = \sum_i [(\psi - \phi_i) \exp(-c_{obst} - d_i) \exp(-(\psi - \phi_i)^2/(2\sigma^2))]$$

with d_i the depth values corresponding to the angles ϕ_i over an angular range σ and a gain c_{obst}. The latter two parameters needed to be tweaked for each simulated environment.

The dataset contains RGB and depth frames at 20 fps annotated with the demonstrated control. In each type of training worlds (Canyon, Forest and Sandbox), data is collected over 100 runs. The training worlds are freshly generated for each run. The algorithm flies at different speeds in order to cover a range of required actions.

An offline validation set is also made in the ESAT corridor in order to optimize the offline training without overfitting.

IV. The DoShiCo Challenge

The DoShiCo challenge allows training on- or off-policy in a mix of 3 types of environments: Canyon, Forest and Sandbox. The performance of the policy should be validated online in the more realistic ESAT environment. Finally the performance is tested on a classification task on real-world data from the almost-collision dataset.

A Mix of Basic Training Environments: Canyon, Forest and Sandbox

Snapshots of the environments are shown in Figure 1. Top-down views can be seen in Figure 3.

One training environment (Canyon) is a canyon that bends randomly, inspired by [28]. The goal is to fly more than 45m through the canyon. From the same work, we copied the idea of a forest with cylinders placed on random spots (Forest) in which the goal is to cross 45m. The third environment is called the Sandbox. It is a box, the size of a big room with walls in varying colors and a number of different objects spread around. We picked 13 basic objects with different shapes found on the Gazebo model server [29]. In the sandbox the agent needs to get further than 7m from the starting position. In each environment the agent is spawned at location $(0, 0)$ and needs to travel without getting any closer than 60cm to an object. The environments are made on the fly during data acquisition or on-policy training.

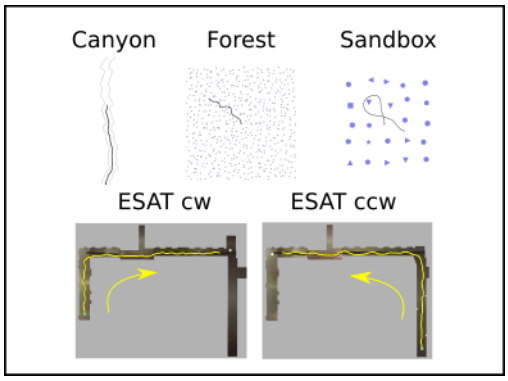

Fig. 3. Top-down view of the training (top) and validation (bottom) environments with an example trajectory.

The three training environments all help to learn a different type of behavior. The Canyon learns the network to focus on perspective lines relevant for wall and corridor following. In the Forest the neural control can learn to avoid moving vertical lines relevant for general obstacle avoidance. The Sandbox ensures a healthy invariance towards unknown shapes and visual features.

More realistic validation environment: ESAT

The network is validated online on the ESAT validation environment. We build a lookalike model of our corridor at the department ESAT. It is important to test online to see the influence of the state-space shift. The policy flies through the ESAT corridor in 2 directions in order to avoid a bias towards one direction. Successful example flights are depicted in the top-down view of Figure 3. Attentive readers might see that the flight clockwise is shorter than in the other direction. Clockwise the required distance for a success is 57m while counterclockwise this is 65m.

Real-world test data: Almost-Collision dataset

Collision avoidance on a real drone performed by different policies is hard to compare correctly. Real-world experiments are influenced by many external factors such as battery state, propeller state, on board electronics, etc.. A drone might crash in one test deteriorating all consecutive flights of different policies.

Using an imitation loss from a demonstration flight by a pilot, on the other hand, is biased towards the specific flying behavior of the pilot which is unfair for comparing one network to another. Different policies might prefer flying more on the left side or the right side of a corridor while both avoiding collision successfully.

In order to compare collision avoidance in a fair and quantitative way on real-world data, we made a small dataset containing images of situations in which only one control is suitable: straight, left or right yaw-turn. We make sure that collision is very nearby in all trajectories, without actually

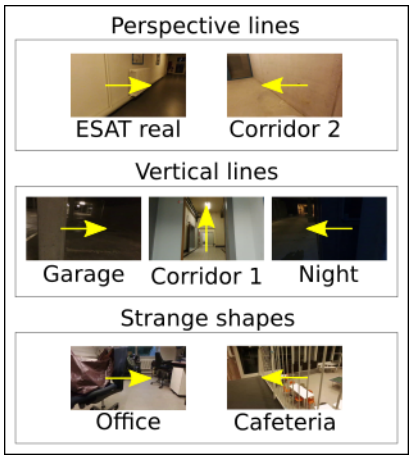

Fig. 4. Snapshots from the seven different locations of the Almost-Collision dataset. The yellow arrow indicates the target direction. Different trajectories have different visual cues to indicate the approaching collision.

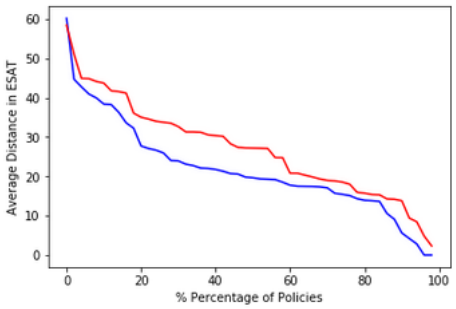

Fig. 5. The variance of the on-policy performance as distance [m] traveled in the ESAT validation environment over the percentage of the population of policies. The red and blue lines corresponds to the NAUX and AUXD architectures.

crashing. This resulted in the convenient name of Almost-Collision dataset.

We recorded data on seven different locations that differ a lot in visible features. Snapshots are shown in Figure 4. The trajectories are tagged with different visible cues. These cues are a type of feature specific to this type of collision: perspective lines, vertical lines and strange shapes. The cues are learned in the corresponding simulated environments: Canyon, Forest and Sandbox.

The trajectories are around 3 to 5 seconds at 20fps. They are labeled with the control required to avoid collision. Over the seven locations a total of 25 trajectories are collected with an equal amount of left and right target controls with the exception of one trajectory with straight as target control. The total size is around 1600 frames. For classification, the predicted angular velocity in yaw is discretized with thresholds ± 0.3 for left, straight and right.

V. Experimental Results

The description of our experiments are grouped in four paragraphs. First we demonstrate the impact of adding depth prediction as an auxiliary task by comparing the performance online in the ESAT environment and offline on the Almost-Collision dataset. In the second paragraph we explore the use of a more realistic validation environment in order to select good policies. The third paragraph handles the trade-off between training on a smaller more realistic environment in comparison to our mix of basic environments. In the final paragraph we give some qualitative results of the online real-world flights of policies trained offline in the mix of basic environments.

NAUX versus AUXD: The influence of depth prediction as an auxiliary task.

The experiments with our baseline methods are exposed to a large amount of variance making it hard to compare different setups. Though the variance seems to be a common problem in the field of training deep neural policies, it is not always openly reported. Jaderberg et al. [30] compare performances of the top 3 policies picked from 50 policies trained with different hyperparameters. Plotting the performance expressed as a percentage of the population of policies reaching this performance, demonstrates the variety over all policies trained. Inspired by this work, we plot a similar graph in Figure 5. The performance is measured as the average collision-free distance traveled by the policy online over 10 runs in the more realistic ESAT validation environment. Please note that the evaluation on the more realistic ESAT environment entails a large domain shift. This shift augments the variance over the different trained policies.

The blue line is the ranked performance of the NAUX networks trained without auxiliary depth. The red line defines the AUXD networks trained with depth prediction. It is clearly visible how the use of auxiliary depth improves the general online performance on the validation environment.

Table I shows the performance averaged over the top 5, top 3 and best policies. The policies are selected based on their average distance traveled online in the ESAT validation environment. The first three rows are the online performances of both NAUX and AUXD policies tested in new generated environments similar to the training environment. The performances are expressed as average collision-free distance and average number of successes out of 10 runs.

Evaluating the performance on environments similar to the training environments is the common practice in training deep neural control. It is clear how, both with and without auxiliary depth, the policies can already succeed a large number of times. In other words, it has learned to succeed at avoiding collisions in the basic environments. Training on longer and more data with possibly intermediate on-policy iterations or larger networks could improve these numbers further but that is not the goal of this work.

The last row of the table shows the average performance of the best policies online in the ESAT environment. It is remarkable that the top networks succeed at performing this task. The ESAT validation environment is not only visually very different, also the trajectory is very different. The distance is much longer and the turns of 90 degrees are not present in the Canyon training environments. Moreover

TABLE I

ONLINE PERFORMANCE IN SIMULATION

	Average distance [m]						Average successes out of 10					
	TOP 5		TOP 3		TOP 1		TOP 5		TOP 3		TOP 1	
	NAUX	AUXD	NAUX	AUXD	NAUX	AUXD	NAUX	AUXD	NAUX	AUXD	NAUX	AUXD
Canyon	43.96	38.41	42.33	43.08	38.05	41.79	9.2	7.4	8.67	8.33	8	7
Forest	45.99	50.24	42.90	48.48	48.67	51.35	5.6	5.4	5.33	4.67	6	4.66
Sandbox	7.03	8.62	6.98	9.11	9.22	8.09	5.6	5.6	5.67	5.33	7	5
ESAT	47.03	57.63	50.08	61.66	60.25	71.69	1.2	4.6	2	5.33	5	8

TABLE II

ACCURACIES ON THE ALMOST-COLLISION DATASET [%]

	TOP 5		TOP 3		TOP 1	
	NAUX	AUXD	NAUX	AUXD	NAUX	AUXD
ESAT real	35	**64**	28	**80**	27	**73**
Corridor 1	40	**57**	38	**61**	40	**60**
Corridor 2	**53**	39	**51**	45	**49**	21
Office	76	**99**	78	**100**	100	100
Cafeteria	29	25	30	26	**42**	34
Garage	50	**62**	45	**56**	46	**58**
Night	**77**	63	72	71	**76**	59
Avg. Loc.	51	**58**	49	**63**	54	58
Strange	46	**54**	45	**57**	52	**60**
Perspective	46	49	43	**53**	46	44
Vertical	70	72	68	**76**	66	**73**
Avg. Cue	54	58	52	**62**	55	59

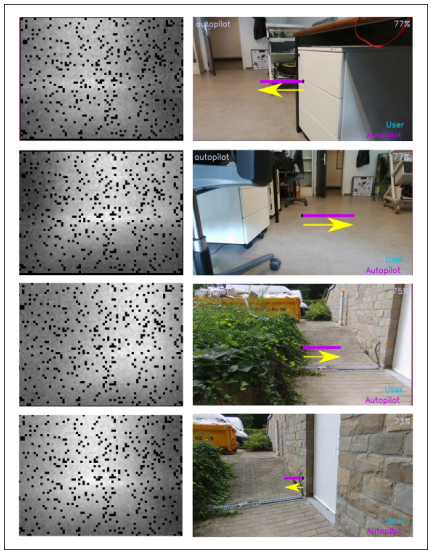

Fig. 6. Auxiliary depth prediction on real-world images.

the network is trained off-policy from a dataset so has never actually flown before.

As Figure 5 implied, the auxiliary depth has a consistent positive impact on the validation performance. This positive impact is less present in environments similar to the training environment which confirms the believe that the auxiliary task helps to regularize over a domain shift.

The step to the real world comes with an extra domain shift. The best policies are quantitatively tested on the Almost-Collision dataset. The results are shown in table II. The top rows show the results per location and with an overall average taken with equal weight for each location. The bottom rows show the results per type of visual cue with an overall average taken with equal weight for each visual cue. The best accuracy between NAUX and AUXD is put in bold if the difference is significant (greater than 5%). The improvement is less distinct with auxiliary depth on real world data. Although the average top 5 and top 3 performances indicate a clear positive trend towards the use of auxiliary depth.

The auxiliary depth prediction performs still reasonably well, qualitatively speaking, on real-world images as depicted in Figure 6. Dark corresponds to close and white corresponds to far. The black dots visible in the depth prediction are due to the short period of training. Depth prediction is only an auxiliary task, since the output is not used by the policy. The training is finished once the policy has converged which happened before the depth prediction was fully optimized. This results in weights in the output layer that are still saturated at zero. Thanks to the depth prediction, we are able to get already an impression of how well the policy might perform in a new environment. Although there are no plants or doorways in the training environments, the depth already gives an indication that the plants are close and the furthest point in the cluttered office is around the door in the back.

The use of simulated ESAT as a Validation environment

The basic simulated training environments are very generic. In order to test how much a policy suffers from switching from the basic training environments to a new environment, it is necessary to validate the policy. If the best policies are validated on environments similar to the training environment, we cannot be sure that the policy performs well after a domain shift.

Table III shows a comparison of policies selected by their online performance on a mix of new generated environments (Canyon, Forest and Sandbox) and policies selected by their online performance in the ESAT environment.

It is clear that networks picked by the best performance in environments similar to the training environment do not necessarily perform well in the ESAT environment due to the domain shift. For instance the average distance of the top 5 networks validated on the Mix validation environment is 39.57, much less than the 57.63 of the top 5 nets validated on the ESAT environment.

In other words, it is required to have a good validation

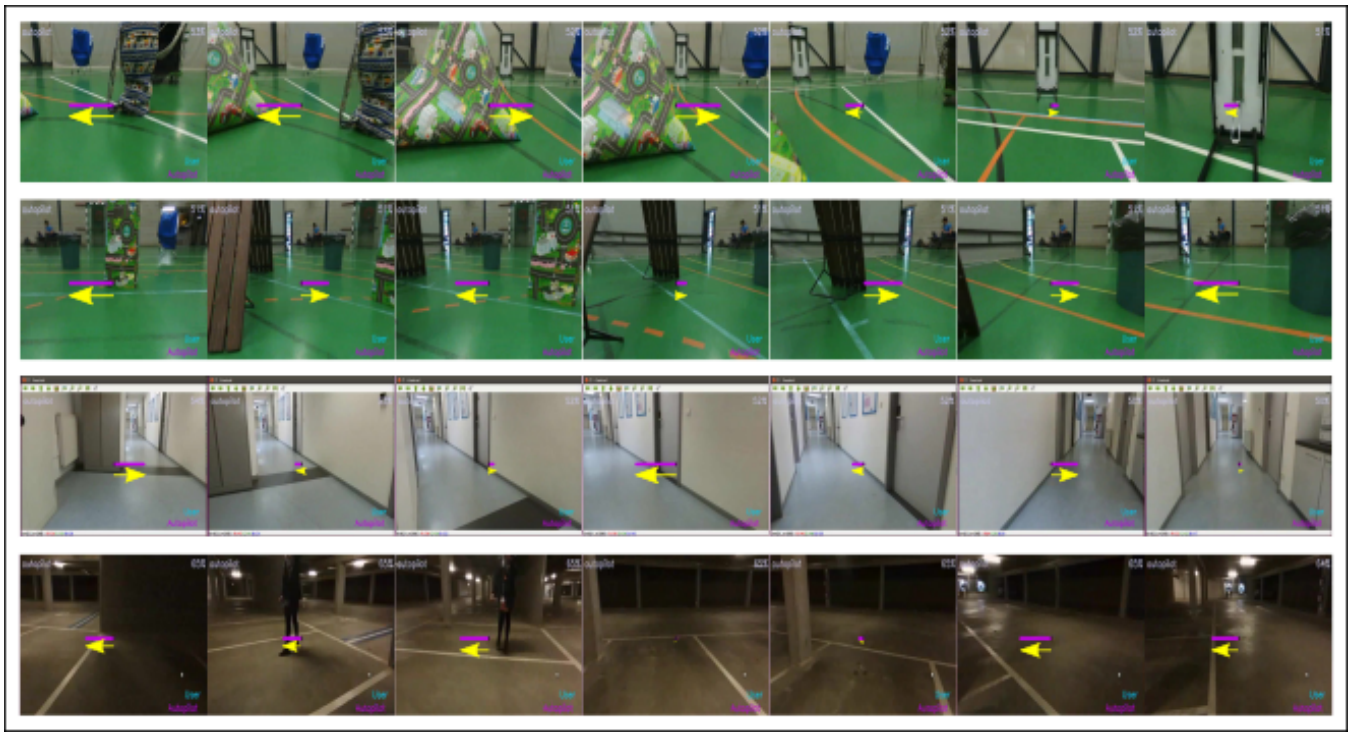

Fig. 7. Small trajectories flown online in the real-world by a policy trained on basic simulated environments. The yellow arrow is added for visualization of the predicted control (originally in purple).

TABLE III
AVERAGE ONLINE PERFORMANCE OF TOP 5 SELECTED ON MIX (CANYON, FOREST, SANDBOX) AND ON ESAT ENVIRONMENT

	Average distance [m]		Successes out of 10	
	MIX	ESAT	MIX	ESAT
Canyon	43.37	38.41	9.00	7.40
Forest	60.08	50.24	8.40	5.40
Sandbox	9.4	8.62	5.60	5.60
ESAT	39.57	57.63	1.50	4.60

setting to see how well the policy can handle a domain shift. For real-world applications it is best that the validation setting comes as close as possible to the real-world setting. Note that the requirement of only one good look-alike of the real-world setting is a lot more feasible than a large randomly generated collection of real-world environments.

Training only in the more realistic ESAT environment

Instead of training on this variety of basic simulated environments, one could also train on the more realistic environment. We compare the performance of the top 5 networks trained on the basic mixed models with the performance of the model trained on the more realistic ESAT environment. By training on the more realistic environment, the domain shift is reduced and features could be extracted that are most relevant for this kind of environment. Table IV indicates that this is not always a good idea. The performance in the realistic environment, in this case ESAT real, is of course much better. However the gain you can expect by reducing the domain shift will be lost once the policy is required to generalize to new environments. As visible for Corridor 1, the policy trained on only basic environments can outperform the policy trained on one more realistic environment. For the moment our baseline is able to compete with a policy trained on more realistic data.

However, when the proposed DoChiCo challenge is resolved and we succeed at training policies in basic environments while still performing well on more realistic environment, one can expect that the policy trained in a large variety of basic environments will outperform the policy trained solely on simulated lookalikes. It is this domain shift that keeps the deep neural control network from using the knowledge obtained in simulation at application time.

Qualitative results on real-world fligths

Flying a drone indoors in the real-world can be delicate. The turbulences caused by the rotors make it hard for the drone to stabilize. We found that experiments mostly failed due to external factors like turbulences, drift, blurred images or bad wifi connection. This made it hard to compare different online experiments. Rather than working with average crash per distance or average pilot intervention over time, we decided to restrict quantitative results to the offline dataset.

Still we want to show what the policy is already capable off. We use the AUXD network that performed best on the ESAT validation environment. Figure 7 depicts four trajectories. The two upper trajectories are around 5 seconds. In the first one the policy successfully avoids a djembe and a toy mat but mistakes the white table as free space. In the

TABLE IV

ACCURACIES ON THE ALMOST-COLLISION DATASET OF THE TOP 5 POLICIES TRAINED ON MIX AND ESAT ENVIRONMENT. [%]

	MIX	ESAT
ESAT real	64	**90**
Corridor 1	**67**	55
Corridor 2	**35**	30
Office	**52**	33
Cafeteria	25	**39**
Garage	**61**	47
Night	63	**71**
Avg. Loc.	52	52
Strange	44	44
Perspective	54	55
Vertical	72	68
Avg. Cue.	57	56

second trajectory a toy mat, wooden bench and dustbin are avoided. Due to the drift however, one propeller gets hit by the bench. The series of images shown of the two lower trajectories are around 10 seconds. The third row shows the drone flying over 30m in a narrow corridor of 2m width successfully avoiding a closet. In the garage the drone could fly for long periods of time, circling around a pillar.

VI. CONCLUSION

The ability of neural networks to be trained in simulation and still perform robustly in the real world is a major challenge for applying deep neural control in real-world applications. A lack of benchmarks makes it hard to compare different methods, resulting in a slower progress of this research field. By proposing the DoShiCo challenge as benchmark that is made easily accessible online, we want to boost this research field. The challenge represents a dummy domain shift from basic simulated environments to a more realistic validation environment. For testing on real-world data, an Almost-Collision dataset is provided that represents the full domain shift from simulation to the real world.

As a baseline we propose a method that successfully uses auxiliary depth prediction. Our model succeeds at taking the dummy domain shift from the basic mixed environments to the ESAT validation environment. It is even capable of competing with a model trained in a more realistic environment when tested on the Almost-Collision dataset. The policy trained in very basic environments also succeeds at making short flights in the real world.

REFERENCES

[1] J. Michels, A. Saxena, and A. Y. Ng, "High Speed Obstacle Avoidance using Monocular Vision and Reinforcement Learning," *ICML*, no. 22, 2005.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An Application of Reinforcement Learning to Aerobatic Helicopter Flight," *NIPS*, 2006.

[3] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *Journal of Machine Learning Research*, vol. 17, pp. 1–40, 2016.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *NIPS Deep Learning Workshop*, 2013.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *ICML*, 2 2016.

[6] A. Giusti, J. Guzzi, D. C. Ciresan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots," *IEEE Robotics and Automation Letters*, vol. 1, pp. 661–667, 7 2016.

[7] D. Gandhi, L. Pinto, and A. Gupta, "Learning to Fly by Crashing," *Arxiv 1704.05588*.

[8] S. Ross, N. Melik-Barkhudarov, S. K. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and H. Martial, "Learning Monocular Reactive UAV Control in Cluttered Natural Environments," *ICRA*, pp. 1765–1772, 2013.

[9] F. Sadeghi and S. Levine, "(CAD)ˆ2RL: Real Single-Image Flight without a Single Real Image," *Robotics: Science and Systems(RSS)*, 2017.

[10] S. R. Richter, V. Vineet, S. Roth, V. Koltun, and T. Darmstadt, "Playing for Data: Ground Truth from Computer Games," *ECCV*, no. 14, 2016.

[11] H. Izadinia, Q. Shan, and S. M. Seitz, "IM2CAD," *CVPR*, 2017.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *Arxiv 1704.04861*.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, pp. 211–252, 12 2015.

[14] J. Engel, V. Koltun, and D. Cremers, "Direct Sparse Odometry," *arXiv:1607.02565*, 2016.

[15] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive Mapping and Planning for Visual Navigation," *CVPR*, 2017.

[16] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars, and L. Van Eycken, "CNN-based single image obstacle avoidance on a quadrotor," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017.

[17] D. Eigen, C. Puhrsch, and R. Fergus, "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network," 6 2014.

[18] P. Althaus and H. I. Christensen, "Behaviour Coordination for Navigation in Office Environments," *Intelligent Robots and Systems IEEE/RSJ*, 2002.

[19] D. A. Pomerleau, "Rapidly Adapting Artificial Neural Networks for Autonomous Navigation," *NIPS*, 1991.

[20] Y. Lecun, U. Muller, J. Ben, and E. Cosatto, "Off-Road Obstacle Avoidance through End-to-End Learning," *NIPS*, 2005.

[21] S. Daftry, J. A. Bagnell, and M. Hebert, "Learning Transferable Policies for Monocular Reactive MAV Control," *International Symposium on Experimental Robotics (ISER)*, 8 2016.

[22] G. Csurka, "Domain Adaptation for Visual Applications: A Comprehensive Survey," in *Domain Adaptation in Computer Vision Applications* (Gabriela Csurka, ed.), Springer.

[23] R. Caruana, T. Mitchell, H. Simon, and D. Pomerleau, "Multitask Learning Rich Caruana 23 September1997," *CMU-CS-97-203. School of Computer Science. Carnegie Mellon University. Pittsburgh, PA 15213*, 1997.

[24] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, and D. London, "Learning To Navigate In Complex Environments," *ICLR*, 2017.

[25] S. Guadarrama and N. Silberman, "TensorFlow-Slim," 2017.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[27] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *arXiv:1212.5701 [cs.LG]*, 12 2012.

[28] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, "PLATO: Policy Learning using Adaptive Trajectory Optimization," *ICRA*, 2017.

[29] Open Source Robotics Foundation, "Gazebo."

[30] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, K. Kavukcuoglu, and D. London, "Reinforcement Learning with Unsupervised Auxiliary Tasks," *arxiv 1611.05397*.