DroNet: Learning to Fly by Driving

Antonio Loquercio*, Ana Isabel Maqueda †, Carlos R. del-Blanco †, and Davide Scaramuzza*

***Abstract*—Civilian drones are soon expected to be used in a wide variety of tasks, such as aerial surveillance, delivery, or monitoring of existing architectures. Nevertheless, their deployment in urban environments has so far been limited. Indeed, in unstructured and highly dynamic scenarios drones face numerous challenges to navigate autonomously in a feasible and safe way. To cope with the above challenges, this paper proposes DroNet, a convolutional neural network that can safely drive a drone through the streets of a city. Designed as a fast 8-layers residual network, DroNet produces, for each single input image, two outputs: a steering angle, to keep the drone navigating while avoiding obstacles, and a collision probability, to let the UAV recognize dangerous situations and promptly react to them. But how to collect enough data in an unstructured outdoor environment, such as a city? Clearly, having an expert pilot providing training trajectories is not an option given the large amount of data required and, above all, the risk that it involves for others vehicles or pedestrians moving in the streets. Therefore, we propose to train a UAV from data collected by cars and bicycles, which, already integrated into urban environments, would expose other cars and pedestrians to no danger. Although trained on city streets, from the viewpoint of urban vehicles, the navigation policy learned by DroNet is highly generalizable. Indeed, it allows a UAV to successfully fly at relative high altitudes, and even in indoor environments, such as parking lots and corridors.**

Fig. 1: **DroNet** is a convolutional neural network, whose purpose is to reliably drive an autonomous drone through the streets of a city. Trained with data collected by cars and bicycles, our system learns from them to follow basic traffic rules, *e.g*, do not go out of the road, and to safely avoid other pedestrians or obstacles. Surprisingly, the policy learned by DroNet is highly generalizable, and even allows to fly a drone on indoor corridors and parking lots.

Supplementary Material

For supplementary video see: https://youtu.be/qlSb8rpSYzM Code and dataset will be soon released!

I. Introduction

Safe and reliable outdoor navigation of autonomous systems, *e.g.* unmanned aerial vehicles (UAVs), is a challenging open problem in robotics. Being able to successfully navigate while avoiding obstacles is indeed crucial to unlock many robotics applications, *e.g.* surveillance, construction monitoring, delivery, and emergency response [1], [2], [3]. A robotic system facing the above tasks should simultaneously solve many challenges in perception, control, and localization. These become particularly harder when working in urban areas, as the one illustrated in Fig. 1, where the autonomous agent is not only expected to navigate while avoiding collisions, but also to safely interact with other agents present in the environment, such as pedestrians or cars.

The traditional approach to tackle this problem is a two step interleaved process consisting of (i) automatic localization in a given map (using GPS, visual and/or range sensors), and (ii) computation of control commands to allow the agent to avoid obstacles while achieving its goal [1], [4]. Even though advanced SLAM algorithms enable localization under a wide range of conditions [5], visual aliasing, dynamic scenes, and strong appearance changes can drive the perception system to unrecoverable errors. Moreover, keeping the perception and control blocks separated not only hinders any possibility of positive feedback between them, but also introduces the challenging problem of inferring control commands from 3D maps.

Recently, new approaches based on deep learning have offered a way to tightly couple perception and control, achieving impressive results in a large set of tasks [6], [7], [8]. Among them, methods based on reinforcement learning (RL) suffer from significantly high sample complexity, hindering their application to UAVs operating in safety-critical environments. In contrast, supervised-learning methods offer a more viable way to learn effective flying policies [6], [9], [10], but they still leave the issue of collecting enough expert trajectories to imitate. Additionally, as pointed out by [10], collision trajectories, usually avoided by expert human pilots, are actually necessary to let the robotic platform learn how to behave in dangerous situations.

Contributions

Clearly, a UAV successfully navigating through the streets, should be able to follow the roadway, as well as promptly react

*The authors are with the Robotics and Perception Group, Dep. of Informatics University of Zurich and Dep. of Neuroinformatics of the University of Zurich and ETH Zurich, Switzerland—http://rpg.ifi.uzh.ch.

†Universidad Politècnica de Madrid, Grupo de Tratamiento de Imágenes

to dangerous situations exactly as any other grounded vehicle would do. Therefore, we herein propose to use data collected from ground vehicles, already integrated in environments as the above-mentioned. Overall, this work makes the following contributions:

- We propose a residual convolutional architecture that, predicting a steering angle and a collision probability, can fly a quadrotor safely and smoothly in urban environments. To train it, we employ an outdoor dataset recorded from cars and bicycles.
- We collect a custom dataset of outdoor collision sequences, to let a UAV predict potentially dangerous situations.
- Trading off performance for processing time, we show that our design represents a good fit for navigation-related tasks. Indeed, it enables real-time processing of the video stream recorded by a UAV's camera.
- Through an extensive evaluation, we show that, even if trained from an outdoor city dataset, out approach generalizes to a large number of scenarios, including indoor corridors, and parking lots.

II. Related work

A wide variety of techniques for drone navigation and obstacle avoidance can be found in the literature. At high level, these methods differ depending on the kind of sensory input and processing employed to control the flying platform.

An UAV operating outdoor is usually provided with GPS, range, and visual sensors to estimate the system state, infer the presence of obstacles, and perform path planning [1], [4]. Nevertheless, such works are still prone to fail in urban environments, where the presence of high rise buildings, and dynamic obstacles can result in significant undetected errors in the system state estimate. The prevalent approach in such scenarios is that of SLAM, where the robot simultaneously builds a map of the environment and self-localizes in it [5]. On the other hand, while an explicit 3D reconstruction of the environment can be good for global localization and navigation, it is not entirely clear how to infer from it control commands for a safe and reliable flight.

Recently, an increasing research effort has been spent into directly learning control policies from raw sensory data using Deep Neural Networks. These methodologies can be divided into two main categories: (i) methods based on reinforcement learning (RL) [7], [11], [12] and (ii) methods based on supervised learning [6], [13], [9], [10].

While RL-based algorithms have been successful in learning generalizing policies [7], [11], [8], they usually require a large amount of robot experience, limiting their applicability in real safety-critical systems. Supervised learning, instead, offers a more viable way to train control policies, but clearly depends upon the provided expert signal to imitate. This supervision may come from a human expert [6], hard-coded trajectories [10], or model predictive control [13]. However, when working in the streets of a city, it can be both tedious and dangerous to collect a large set of expert trajectories, or evaluate partially trained policies [6]. Additionally, the domain-shift between expert and agent might hinder generalization capabilities of supervised learning methods. Indeed, previous work in [9] trained a UAV from video collected by a mountain hiker, but did not show the learned policy to generalize to scenarios unseen at training time.

Another promising approach has been to use simulations to get training data for reinforcement or imitation learning tasks, while testing the learned policy in the real world [14], [15], [12]. Clearly, this approach suffers from the domain shift between simulation and reality, and it might require some real world data to actually generalize [12]. To our knowledge, current simulators still fail to model the large amount of variability present in an urban scenario, and therefore they are not fully acceptable for our task.

To overcome the above-mentioned limitations, we propose to train a neural network policy by imitating expert behaviour, only generated from wheeled manned vehicles. Our data collection proposal does not require any state estimate or even an expert drone pilot, while it exposes pedestrians, other vehicles, and the drone itself to no danger. Furthermore, we show our methodology to have very good generalization capabilities, even in scenarios very different from the training ones.

III. Methodology

Our learning approach aims at reactively predicting a steering angle and a probability of collision from the drone on-board forward-looking camera. These are later converted into control flying commands, that enable a UAV to safely navigate while avoiding obstacles.

Since we aim to reduce the bare image processing time, we advocate a single convolutional neural network (CNN) with a relative small size. The resulting network, which we call DroNet, is shown in Figure 2 (a). The architecture is partially shared by the two tasks to reduce the network's complexity and processing time, but is then separated into two branches at the very end. Indeed, steering prediction is a regression problem, while collision prediction is addressed as a binary classification problem. Due to their different nature and output range, we propose to separate the network's last fully-connected layer.

During the training procedure, we use only imagery recorded by manned vehicles. Steering angles are learned from images captured from a car, while probability of collision, from a bicycle.

A. Learning Approach

The part of the network that is shared by the two tasks consists of a ResNet-8 architecture followed by a dropout of 0.5 and a ReLU non-linearity. The residual blocks of the ResNet, proposed by He et al. [16] to increase model generalization, are shown in Figure 2 (b). Dotted lines represent skip connections, defined as 1×1 convolutional shortcuts to let the input and output of the residual blocks be added. We opted for this regularizing technique to control the model's overfitting, and thus to increase its generalization capacity to new unseen scenarios. After the last ReLU layer, tasks stop sharing parameters, and the architecture splits into two different fully-connected layers. The first one outputs the steering angle, and

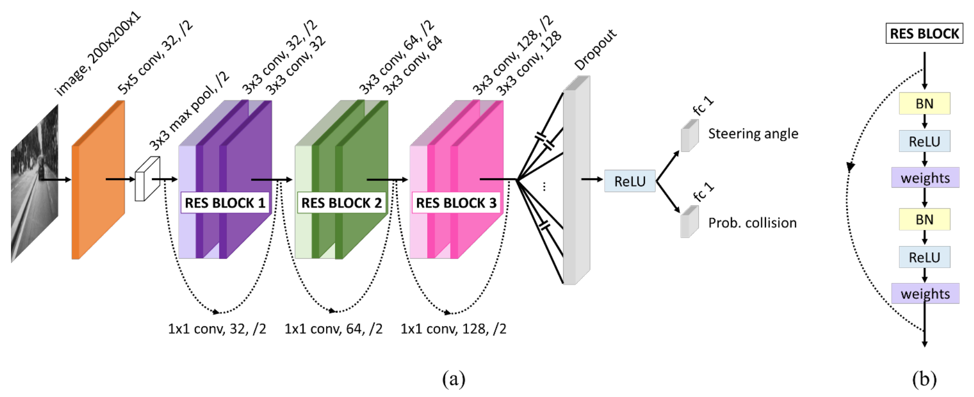


Fig. 2: (a) DroNet is a forked Convolutional Neural Network that predicts, from a single 200×200 frame in gray-scale, a steering angle and a collision probability. The shared part of the architecture consists of a ResNet-8 with 3 residual blocks (b), followed by dropout and ReLU non-linearity. After them, the network branches into two separated fully-connected layers, one to carry out steering prediction, and the other one to infer collision probability. In the notation above, we indicate for each convolution first the kernel's size, then the number of filters, and eventually the stride, if different from 1.

the second one a collision probability. Strictly speaking the latter is not a Bayesian probability, but an index quantifying the network uncertainty in prediction. Slightly abusing the notation, we still refer to it as "probability".

We use mean-squared error (MSE) and binary cross-entropy (BCE) to train the steering and collision predictions, respectively. Whereas the network architecture proves to be appropriate to minimize complexity and processing time, a naive joint optimization poses serious convergence problems due to the very different gradients' magnitude that each loss produces. Indeed, the output ranges and cost functions for MSE and binary cross-entropy are not the same. In particular, gradients computed for the steering component are initially higher, hence they can hinder the optimization of the collision prediction. We solve this issue by doing a sequential joint optimization. We first start solving the regression task by exclusively training on it for a few epochs, and then we start learning the binary classifier. To do that, we define a different weight for each loss, as detailed in Eq. (1). While the loss weight for steering prediction is always 1, the one corresponding to the binary classifier increases with the number of epochs. For our experiments, we set $decay = \frac{1}{10}$, and $epoch_0 = 10$.

$$L_{tot} = L_{MSE} + \max(0, 1 - \exp^{-decay(epoch - epoch_0)}) L_{BCE} \quad (1)$$

The Adam optimizer [17] is used with a starting learning rate of 0.001, and an exponential per-step decay equal to 10^{-5}. We also employ hard negative mining, in order to focus the optimization on those samples that are most difficult to learn. In particular, we select the k samples with the highest loss in each epoch, and compute the total loss according to Eq. (1). We define k so that it decreases over time.

B. Datasets

To learn steering angles from images, we use one of the publicly available datasets from Udacity's project [18]. This dataset contains over 70,000 images of car driving distributed over 6 experiments, 5 for training and 1 for testing. Every experiment stores time-stamped images from 3 cameras (left, central, right), IMU, GPS data, gear, brake, throttle, steering angles and speed. For our experiment, we only use images from the forward-looking camera (Figure 3 (a)) and their associated steering angles.

To our knowledge, there are no public datasets that associate images with collision probability according to the distance to the obstacles. Therefore, we collect our own collision data by mounting a GoPro camera on the handlebars of a bicycle. We drive along different areas of a city, trying to diversify the types of obstacles (vehicles, pedestrians, vegetation, under-construction sites) and the appearance of the environment (Figure 3 (b)). This way, the drone is able to generalize under different scenarios. We start recording when we are far away from an obstacle and stop when we are very close to it. In total, we collect around 32,000 images distributed over 137 sequences for a diverse set of obstacles. We manually annotate the sequences, so that frames far away from collision are labeled as 0 (no collision), and frames very close to the obstacle are labeled as 1 (collision), as can be seen in Fig. 3(b). Collision frames are the type of data that cannot be easily obtained by a drone, but are necessary to build a safe and robust system.

C. Drone Control

The outputs of DroNet are used to command the UAV to move on a plane with forward velocity v_k and steering angle θ_k. More specifically, we use the probability of collision p_t provided by the network to modulate the forward velocity: the vehicle is commanded to go at maximal speed V_{max} when the probability of collision is null, and to stop whenever it is close to 1. We use a low-pass filtered version of the modulated forward velocity v_k to provide the controller with smooth,

(a) Udacity dataset

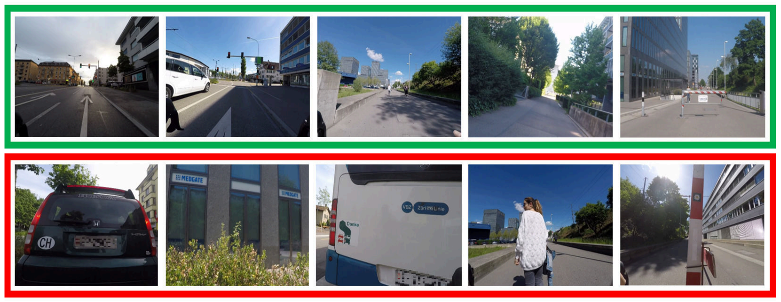

(b) Collected dataset

Fig. 3: (a) Udacity images used to learn steering angles. (b) Collected images to learn probability of collision. The green box contains no-collision frames, and the red one, collision frames.

continuous inputs ($0 \leq \alpha \leq 1$):

$$v_k = (1-\alpha)v_{k-1} + \alpha(1-p_t)V_{max}, \tag{2}$$

Similarly, we map the predicted scaled steering s_k into a rotation around the body z-axis (yaw angle, θ), corresponding to the axis orthogonal to the propellers' plane. Concretely, we convert s_k from a $[-1,1]$ range into a desired yaw angle θ_k in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$ and low-pass filter it:

$$\theta_k = (1-\beta)\theta_{k-1} + \beta \; \frac{\pi}{2} \; s_k \tag{3}$$

In all our experiments, we set $\alpha = 0.7$ and $\beta = 0.5$, while V_{max} was changed according to the testing environment. The above constants have been selected empirically trading off smoothness for reactiveness of the drone's flight.

IV. Experimental Results

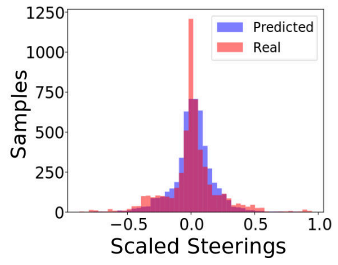
In this section, we show quantitative and qualitative results of our proposed methodology. First, we evaluate the accuracy of DroNet with a set of performance metrics; then, we discuss its control capabilities, comparing it again a set of navigation baselines.

A. Hardware Specification

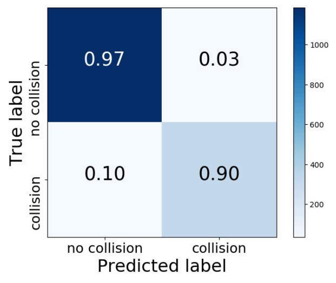
We performed our experiments on a Parrot Bebop 2.0 drone. Designed as an outdoor hobby platform, it has a basic and rather inaccurate, visual odometry system that allows the user to provide only high-level commands, such as body-frame velocities, to control the platform. Velocity commands are produced by our network running on an Intel Core i7 2.6 GHz CPU that receives images at 30 Hz from the drone through Wi-Fi.

B. Regression and Classification Results

We first evaluate the regression performance of our model employing the testing sequence from the Udacity dataset [18]. To quantify the performance on steering prediction, we use two metrics: root-mean-squared error (RMSE) and explained variance ratio (EVA)[1]. To asses the performance on collision



(a)



(b)

Fig. 4: **Model performance:** (a) Probability Density Function (PDF) of actual vs. predicted steerings of the Udacity dataset testing sequence. (b) Confusion matrix on the collision classification evaluated on testing images of the collected dataset.

prediction, we use average classification accuracy and the F-1 score[2].

Table I compares DroNet against a set of other architectures from the literature [16], [9]. Additionally, we use as weak baselines a constant estimator, that always predicts 0 as steering angle and "no collision", and a random one. From these results we can observe that our design, even though 80 times smaller than the best architecture, maintains a considerable prediction performance, while achieving real-time operation (20 frames per second). For this reason, our design succeeds at finding a good trade-off between performance and processing time, as shown in Table I and Fig. 4. Indeed, to enable a drone to promptly react to unexpected events or dangerous situations, it is necessary to reduce the network's latency as much as possible.

C. Quantitative Results on DroNet's Control Capabilities

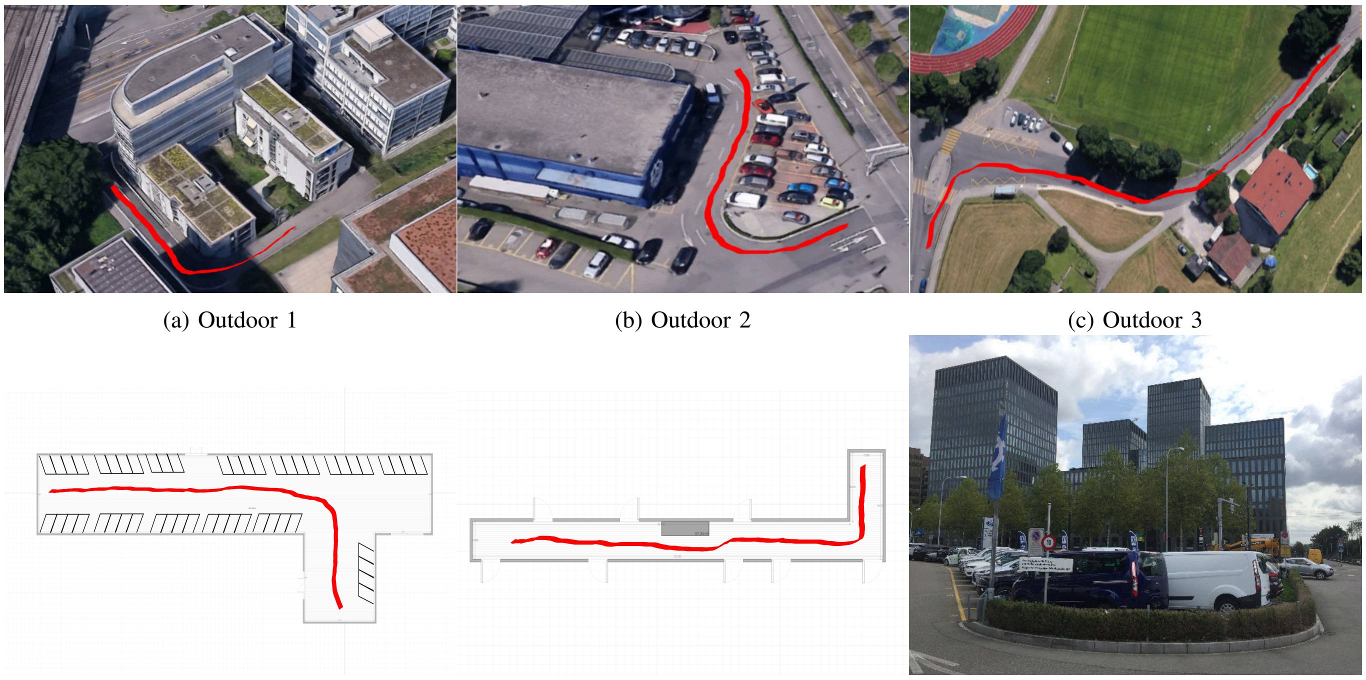
We tested our DroNet system by autonomously navigating in a number of different urban trails, including straight paths and sharp curves. Moreover, to test the generalization capabilities of the learned policy, we also performed experiments in indoor environments. An illustration of the testing environments can be found in Fig. 5 and Fig. 6. We compare our approach against two baselines:

[1]Explained Variance is a metric used to quantify the quality of a regressor, and is defined as $EVA = \frac{\text{Var}[y_{true} - y_{pred}]}{\text{Var}[y_{true}]}$

[2]F-1 score is a metric used to quantify the quality of a classifier. It is defined as F-1$= 2\frac{precision \times recall}{precision + recall}$

Model	EVA	RMSE	Avg. accuracy	F-1 score	Num. Layers	Num. parameters	Processing time [fps]
Random baseline	-1.0 $\pm$ 0.022	0.3 $\pm$ 0.001	50.0$\pm$0.1%	0.3$\pm$0.01	-	-	-
Constant baseline	0	0.2129	75.6%	0.00	-	-	-
Giusti et al. [9]	0.672	0.125	91.2%	0.823	6	5.8×10^4	23
ResNet-50 [16]	0.795	0.097	96.6%	0.92	50	2.6×10^7	7
DroNet (Ours)	0.737	0.109	95.4%	0.901	8	3.2×10^5	20

TABLE I: **Quantitative results on regression and classification task: EVA** and **RMSE** are computed on the steering regression task, while **Avg. accuracy** and **F-1 score** are evaluated on the collision prediction task. Our model compares favorably against the considered baselines. Despite being relatively lightweight in terms of number of parameters, DroNet maintains a very good performance on both tasks. We additionally report the on-line processing time in frames per second (fps), achieved when receiving images at 30 Hz from the UAV.


(a) Outdoor 1 (b) Outdoor 2 (c) Outdoor 3

(d) Indoor Parking Lot (e) Indoor Corridor (f) Closer view of Outdoor 2

Fig. 5: **Testing environments:** (a) *Outdoor 1* is a 90° curve with a dead end. This scenario is also tested with the drone flying at high altitude (5 m), as shown in Fig. 6. (b) *Outdoor 2* is a sharp 160° curve followed by a 30 m straight path. A closer view of this environment can be seen in (f). (c) *Outdoor 3* is a series of 2 curves, each of approximately 60°, with straight paths in between. Moreover, we also tested DroNet on scenarios visually different from the training ones, such as (d) an indoor parking lot, and (e) an indoor corridor.

(a) *Straight line policy*: trivial baseline consisting in following a straight path in open-loop. This baseline is expected to be very weak, given that we always tested in environments with curves.

(b) *Minimize probability of collision policy*: strong baseline consisting in going toward the direction minimizing the collision probability. For this approach, we implemented the algorithm proposed in [10], shown by the authors to have very good control capabilities in indoor environments. We employ the same architecture as in DroNet, along with our collected dataset, to estimate the collision probability.

We use as metric the average distance travelling before stopping or colliding. Results from Table II indicate that DroNet is able to drive a UAV the longest on almost all the selected testing scenarios. The main strengths of the policy learned by DroNet are twofold: (i) the platform smoothly follows the road lane, while avoiding static obstacles; (ii) the drone is never driven into a collision, even in presence of dynamic obstacles, *e.g.*, pedestrians, or bicycles, occasionally occluding its path. Another interesting feature of our method is that, in open spaces and at intersections, DroNet usually drives the vehicle to a random direction. In contrast, the baseline policy of minimizing the probability of collision was very often confused by intersections and open spaces, and resulted in a shaky uncontrolled behaviour. This explains the usually large gaps in performance between our selected methodology and the considered baselines.

Interestingly, the policy learned by DroNet generalizes well to scenarios visually different from the training ones, as shown in Table II. First, we noticed only a very little drop in performance when the vehicle was flying at relatively high altitude (5 m). Even though the drone's viewpoint was in this case different from a grounded vehicle's one (usually at 1.5 m), the curve could be successfully completed as long as the path was in the field of view of the camera. More surprisingly was the generalization of our method to

	Urban Environment			Generalization Environments		
Policy	Outdoor 1	Outdoor 2	Outdoor 3	High Altitude Outdoor 1	Corridor	Garage
Straight	23 m	20 m	28 m	23 m	5 m	18 m
Ghandi et al. [10]	38 m	42 m	75 m	18 m	**31 m**	23 m
DroNet (Ours)	**52 m**	**68 m**	**245 m**	**45 m**	27 m	**50 m**

TABLE II: **Average travelled distance before stopping**: We show here navigation results using three different policies on a several environments. Recall that [10] uses only collision probabilities, while DroNet uses also predicted steering angles, too. *High Altitude Outdoor* 1 consists of the same path as *Outdoor 1*, but flying at 5 m altitude, as shown in Fig. 6

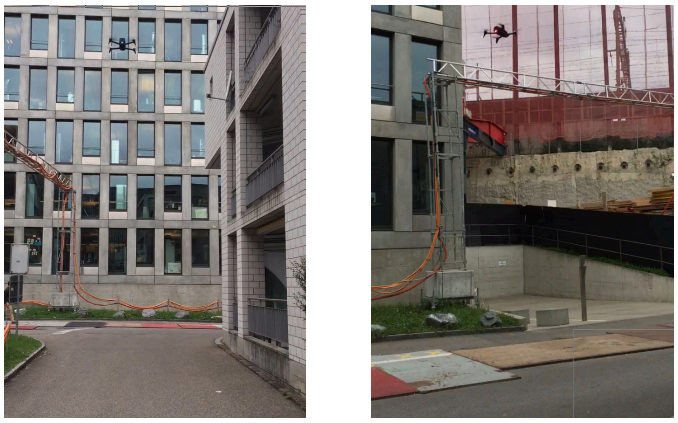


Fig. 6: **High altitude Outdoor 1:** To test the ability of DroNet to generalize at high altitude, we made the drone fly at 5 m altitude in the testing environment *Outdoor* 1. Table II indicates that our policy is able to cope with the large difference between the viewpoint of a camera mounted on a car (1.5 m) and the one of the UAV.

indoor environments, such as a corridor and a parking lot. In these scenarios, the drone was still able to avoid static obstacles, follow paths, and stop in case of dynamic obstacles occluding its way. Nonetheless, we experienced some domain-shift problems. In indoor environments, we experienced some drifts at intersections, that were sometimes too narrow to be smoothly performed by our algorithm; By contrast, as we expected, the baseline policy of [10], specifically designed to work in narrow indoor spaces, outperformed our method. Still, we believe that it is very surprising that a UAV trained on streets can actually perform well even in indoor corridors.

D. Qualitative Results and Discussion

In Fig. 8 and, more extensively, in the supplementary video, it is possible to observe the behaviour of DroNet in some of the considered testing environments. Differently from previous work [9], our approach always produced safe and smooth flight. In particular, the drone always promptly reacted to dangerous situations, *e.g.* sudden occlusions by bikers or pedestrians in front of it.

To better understand our flying policy, we employed the technique outlined in [19]. Fig. 7 shows which part of an image are the more important for DroNet to generate a steering decision. Intuitively, the network concentrates mainly on the "lines-like" patterns present in a frame, that roughly indicate the steering direction. This explains explains why DroNet generalizes so well to many different indoors and outdoors scenes. Indeed, all of them contain distinctive, "line-like" features, *e.g.*, road lanes or wall edges, which show the direction where the UAV is supposed to go, independently of the scene background.

Additionally, the importance of our proposed methodology is supported by the difficulties encountered while carrying out outdoor experiments. In particular, we had to interrupt many times our tests due to the large number of people that an autonomous flying drone inevitability attracts. Likewise, we had to frequently stop our experiments due to the presence of many other drivers, who were not very happy to "share" the street with our drone. Therefore, if we want a drone to learn to fly in a city, it is crucial to take advantage of cars, bicycles, or other manned vehicles that, already integrated in the urban streets, allow to collect enough valid training data.

V. Conclusion

In this paper, we proposed DroNet, a convolutional neural network that can safely drive a drone in the streets of a city. Since collecting data with an UAV in such an uncontrolled environment is a very tedious and dangerous task, our model learns how to navigate from cars and bicycles, who already know the traffic rules. Designed to trade off performance for processing time, DroNet predicts simultaneously the collision probability and the desired steering angle, enabling a UAV to promptly react to unforeseen events and obstacles. Through extensive evaluations, not only we showed, on one side, that a drone can learn to fly by imitating manned vehicles. On the other, we proved that it can very well generalize to a wide variety of scenarios. Our proposed methodology does not aim at replacing supervised learning algorithms for training UAVs, such as DAGGER [6], but can actually be complementary to them. Indeed, it could be used as very good initialization for either reinforcement or imitation learning methods to achieve some particular navigation-related tasks, *e.g.*, search and rescue. For this reason, we will release all our training code and datasets to share our findings with the robotics community.

References

[1] S. Scherer, J. Rehder, S. Achar, H. Cover, A. Chambers, S. Nuske, and S. Singh, "River mapping from a flying robot: state estimation, river detection, and obstacle mapping," *Autonomous Robots*, vol. 33, no. 1-2, pp. 189–214, 2012.

[2] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida *et al.*, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832–841, 2012.

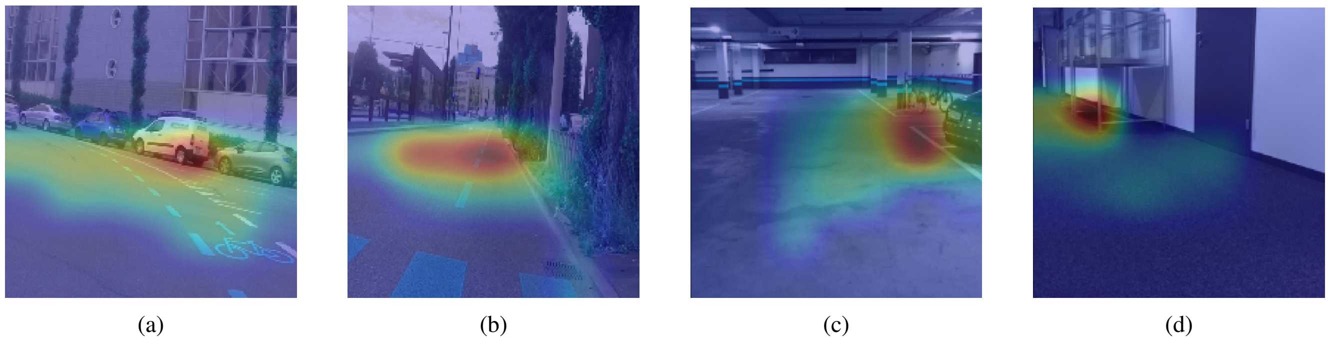


Fig. 7: **Activation maps:** Spatial support regions for steering regression in city streets, on (a) a left curve and (b) a straight path. Moreover we show activations on (c) an indoor parking lot, and (d) an indoor corridor. We can observe that the networks concentrates its attention on “lines-like” patterns, that approximately indicate the steering direction.

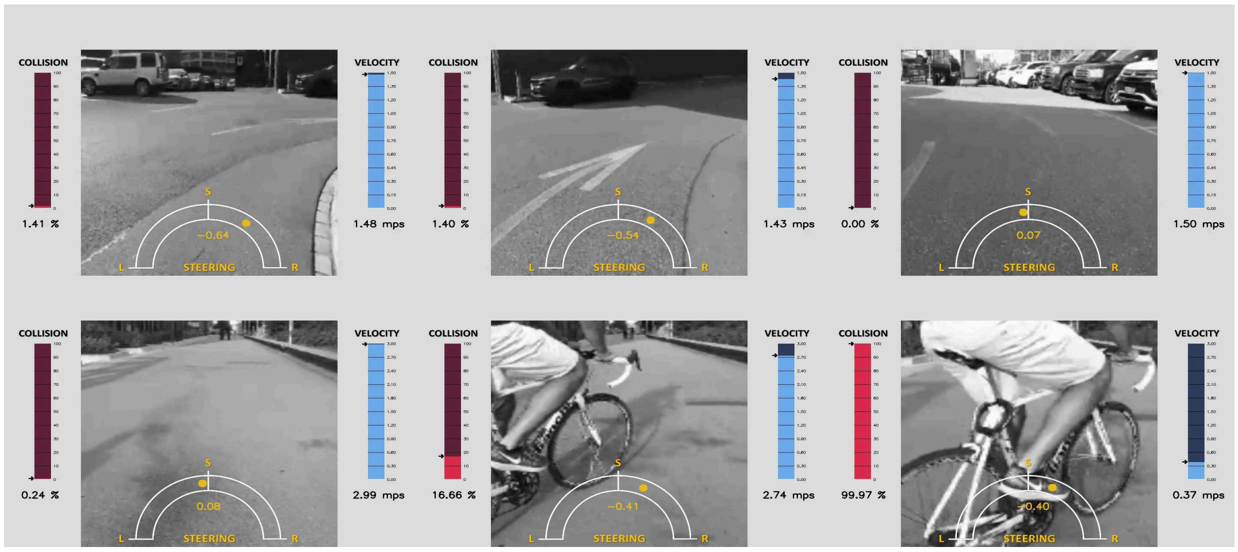


Fig. 8: **DroNet predictions:** The above figures show predicted steering and probability of collision evaluated over several experiments. Despite the diverse scenarios and obstacles types, DroNet predictions always follow common sense and enable safe and reliable navigation.

[3] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, “Autonomous, vision-based flight and live dense 3D mapping with a quadrotor MAV,” *J. Field Robot.*, vol. 33, no. 4, pp. 431–450, 2016.

[4] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, “Multi-sensor fusion for robust autonomous flight in indoor and outdoor environments with a rotorcraft MAV,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4974–4981.

[5] S. Lynen, T. Sattler, M. Bosse, J. A. Hesch, M. Pollefeys, and R. Siegwart, “Get out of my lab: Large-scale, real-time visual-inertial localization.”

[6] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, “Learning monocular reactive UAV control in cluttered natural environments,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2013, pp. 1765–1772.

[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1889–1897.

[9] A. Giusti, J. Guzzi, D. C. Cirean, F. L. He, J. P. Rodrguez, 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, no. 2, pp. 661–667, July 2016.

[10] D. Gandhi, L. Pinto, and A. Gupta, “Learning to fly by crashing,” *arXiv*, vol. abs/1704.05588, 2017.

[11] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, “Rl2: Fast reinforcement learning via slow reinforcement learning,” *arXiv preprint arXiv:1611.02779*, 2016.

[12] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” *arXiv preprint arXiv:1609.05143*, 2016.

[13] G. Kahn, T. Zhang, S. Levine, and P. Abbeel, “Plato: Policy learning using adaptive trajectory optimization,” *arXiv preprint arXiv:1603.00622*, 2016.

[14] F. Sadeghi and S. Levine, “(cad)2rl: Real single-image flight without a single real image,” 2017.

[15] M. Mancini, G. Costante, P. Valigi, T. A. Ciarfuglia, J. Delmerico, and D. Scaramuzza, “Towards domain independence for learning-based monocular depth estimation,” *IEEE Robot. Autom. Lett.*, 2017.

[16] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *arXiv*, vol. abs/1603.05027, 2016.

[17] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *International Conference on Learning Representations*, 2015.

[18] Udacity, “An Open Source Self-Driving Car,” https://www.udacity.com/self-driving-car, 2016.

[19] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, “Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization,” *arXiv preprint arXiv:1610.02391*, 2016.