Machine Learning and a Small Autonomous Aerial Vehicle

Part 1: Navigation and Supervised Deep Learning

Fabrice R. Noreils

1 Introduction

Machine Learning is changing the way we look at technology. Artificial Intelligence (AI) is leading the way in exploring new frontiers, particularly through progress in Machine Learning (ML) and Deep Learning.

Experts predict that many of the deep learning applications that are just around the corner will have a profound impact on our daily lives. In fact, some of these applications are already being put to use in certain industries, and others are poised to significantly reshape others in the near future.

For example, deep neural nets are already greatly utilized in Automatic Machine Translation, chatbots, Automatic Text Generation, Image recognition, Automatic Image Caption Generation, Voice Search & Voice Activated Assistants, Advertising, sentiment analysis, and self-driving cars, and they are sure to have a profound impact on industries like health-care through improvements in diagnosis through medical imaging, drug discovery, monitoring, and automatic treatment and recommendation.

As deep learning continues to mature, we can expect to see applications of deep learning in new domains almost every day.

This brings us to the topic of drones, specifically small drones (to be defined). There are lots of important initiatives happening in navigation, 3D depth extraction, image recognition, object and human tracking, and surveillance.

The aim of this paper is to provide an overview on how deep learning is applied in these domains, explore the existing industrial/commercial applications, and offer some reflections about a potential "paradigm shift" in robotics.

This paper is divided into four parts:

Part 1: Navigation and Supervised Deep Learning
Part 2: Navigation and (Deep) Reinforcement Learning
Part 3: Depth Estimation and Deep Learning
Part 4: Tracking and Surveillance Powered by Deep Learning

This paper is written with the assumption that the reader is at least somewhat familiar with Deep Learning concepts.

2 Indoor/Outdoor Navigation

Navigation refers to the ability of the drone to fly outdoors in the crowded environment of a city, a forest full of far-reaching branches, and indoors in warehouses, offices, or houses, all while avoiding the obstacles in its path—both static and dynamic.

Navigation is a crucial functionality because it unlocks many exciting civil applications, including surveillance, construction monitoring, delivery, and military applications.

2.1 The Current Robotics Paradigm

How does an autonomous system navigate and avoid obstacles? The current paradigm is robotics requires that you first need to know—as precisely as possible—your position with respect to the environment, which leads to the development of algorithms that are simultaneously modeling the environment and localizing the autonomous system with respect to the current known world, also known as “SLAM.” As sensors for perceiving the environment, such as. Lidar, cameras, and sensors to perceive robot movements (e.g., IMU) are noisy, a great amount of effort has been devoted to developing mathematical tools to handle uncertainties. Thus, SLAM is often divided into a front-end (which is in charge of updating the position of the autonomous system with respect to what is seen at a given time) and a back-end (which is in charge of maintaining a consistent global map with the objective to reduce uncertainties). Then, based on the current known map, a path is planned and then executed by issuing motor commands. All these different modules (SLAM front-end/back-end, path planning and execution) run at different time scales.

Most, if not all, of the autonomous robots are designed according to this paradigm, and it works quite well under certain conditions; however, it is also relatively slow since it is highly demanding in terms of CPU and memory, and it is very sensitive to dynamic environment and light conditions if the perception system is using cameras.

Pioneering work related to drone technology began to appear in publications around 2009, including:

- Grzonka et al. from University of Freiburg [1] demonstrated autonomous indoor navigation with a 2D Lidar.
- Shen et al. from the University of Pennsylvania Grasp Lab [2] demonstrated an autonomous indoor navigation on several floors with 2D Lidar + Camera. Shen started a Lab at HKUST in 2014 and has been doing groundbreaking work in the field. This work will be discussed in greater detail later on, in the section on Deep Neural Net and Depth extraction from a monocular camera.
- Barach et al. from MIT [3] demonstrated an autonomous indoor navigation MAV using a Lidar and a Camera. Barach and his teammate Bry founded Skydio in 2014 and released the R1 drone, an innovative piece of technology utilizing Deep Learning network based algorithms.

Today, a great deal of effort has been expended on monocular visual-inertial odometry (VIO) [4] and VIO based on a new type of camera referred to as event-camera [5] which look very promising.

The current Robotics Paradigm is relevant because the experiments done with Deep Learning and Reinforcement learning lead to the emergence of a new paradigm, which will be discussed in Part 2.

2.2 The Arrival of Deep Learning

With the rise of deep learning, researchers are starting to explore how navigation can be achieved with deep neural networks.

Machine learning systems can be classified according to the amount and type of supervision they receive during training. There are three main classifications: supervised learning, unsupervised learning, and reinforcement learning.

Most of the research papers on machine learning applied to drones involve supervised deep learning and reinforcement learnings.

We will start with supervised deep learning and focus on two papers in particular:

- DroNet [6]
- TrailNet [7]

It's vital to consider these two papers for three reasons: (1) the authors have tested their solution in a real environment, (2) these works are state-of-the-art in their respective field of application, and (3) the authors provide lot of insights on the methodology and the implementation on a real drone from end-to-end (specially [7]).

3 Navigation Powered by Supervised Deep Neural Networks

When a researcher is setting up an experiment with Deep Neural Networks (DNN), he or she will follow this general process:

- Define the parameters that will be trained or the parameters that will be used to control the drone.
- Get available dataset(s) or create and annotate your own dataset(s).
- Define the architecture of the DNN.
- Define the loss function and the optimizer.
- Train and test the DNN.
- Implement it and find a way to control the drone based on the DNN outputs.
- Go out in the field and conduct the experiment.

3.1 What are these Papers All About?

DroNet	**TrailNet**
DroNet is a deep neural network (DNN) that aims to reliably drive an autonomous drone through the streets of a city. Thus, it must be able to follow basic traffic rules, (e.g, do not go off the road, safely avoid other pedestrians or	TrailNet is a DNN that computes the orientation and lateral offset of an autonomous drone within the trail. This work is partly inspired by, and most closely related to, the trail DNN research of Giusti et al. [8].

obstacles). It is interesting to note that in the architecture of the system, the detection of obstacles is achieved by DroNet as well.	TrailNet focusses on keeping the drone on the trail while other modules take care of the detection of obstacles.

3.2 Parameters to Control the Drone

DroNet	**TrailNet**
There are only two parameters: one for predicting a steering angle and one for predicting a probability of collision	There are three parameters for the orientation (facing left, facing Center, facing right) plus three lateral offsets (shifted left, centered, shifted right). The author focused on the lateral shifts to keep the drone in the center of the trail.

3.3 The Delicate Question of the Datasets

To train a deep neural network, you need data—a great deal of data. Moreover, when you utilize supervised learning, you must annotate your data, at least for the training set. Annotation is a tedious, time-consuming, and expensive task, as it requires highly experienced and professional workspace to create large volumes of annotated data like pictures or images that can be used to train the models.

Besides needing a lot of data as a prerequisite for training a DNN, one of the many benefits of having datasets with a common reference point is that it allows you to benchmark algorithms.

Over time, the AI community has collected tons of annotated data in most of the domains where machine learning is applied (e.g., MS COCO for object detection, segmentation and captioning dataset; KITTY with a lot of video sequences, and many more).

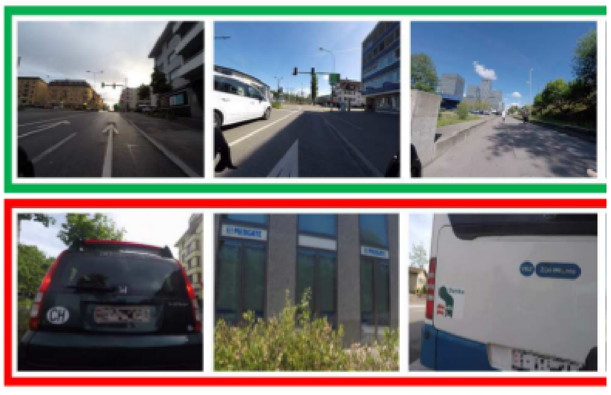
Nevertheless, in some cases, it's simply impossible to find all the data you need, and the situation requires some ingenuity and engineering skills to create your the necessary dataset and annotate it, as is the case for the two examples.

DroNet	**TrailNet**
To learn steering angles from images, the authors used one of the publicly available datasets from Udacity's project [9]. This dataset contains over 70,000 images taken while driving in a car ; these images were distributed over six experiments, five for training and one for testing. Every experiment stores time-stamped images from 3 cameras (left, central, right), IMU, GPS data, gear, brake, throttle, steering	To learn the orientation (left / center / right) parameters, the authors used the IDSIA Swiss Alps trail dataset available from [8]. This dataset includes footage of hikes recorded by 3 cameras (aimed left 30° , straight, and right 30°), with sufficient cross-seasonal and landscape variety. To learn the lateral offset parameters, the authors created their own dataset. They made a three-

angles, and speed.

The Dronet experiment utilized only the images from central camera (looking forward) along with the associated steering angles.

However, the researchers did not find any public datasets relevant to detecting collisions. Therefore, they created their own collision dataset. A GoPro camera was mounted onto the handlebars of a bicycle and the bicycle was ridden around many different areas of Zurich. Through this method, they collected around 32,000 images distributed over 137 sequences, which made for a diverse set of obstacles. Then come the tedious task, manually annotating the sequences, so that frames far away from collision are labeled as 0 (no collision, green in the picture below – from [6]), and frames very close to the obstacle are labeled as 1 (collision, red in the picture below – from [1]):

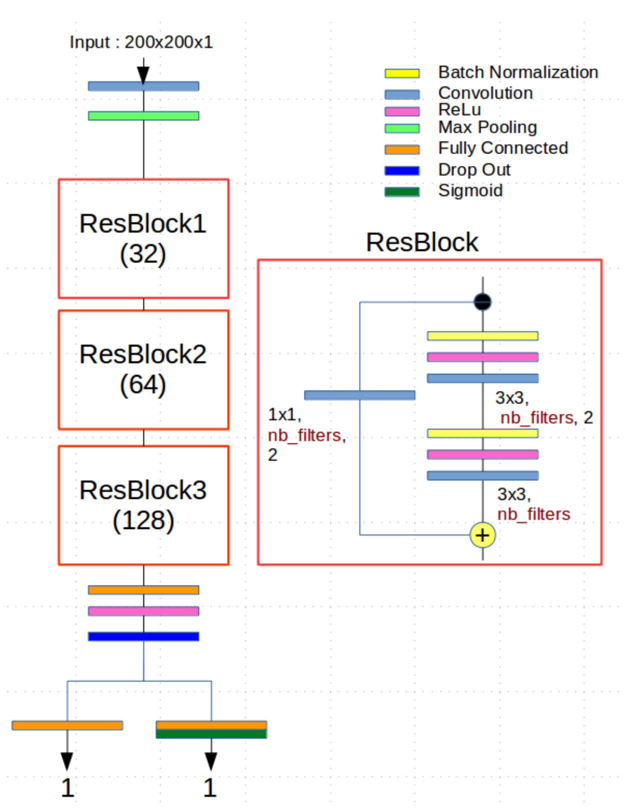
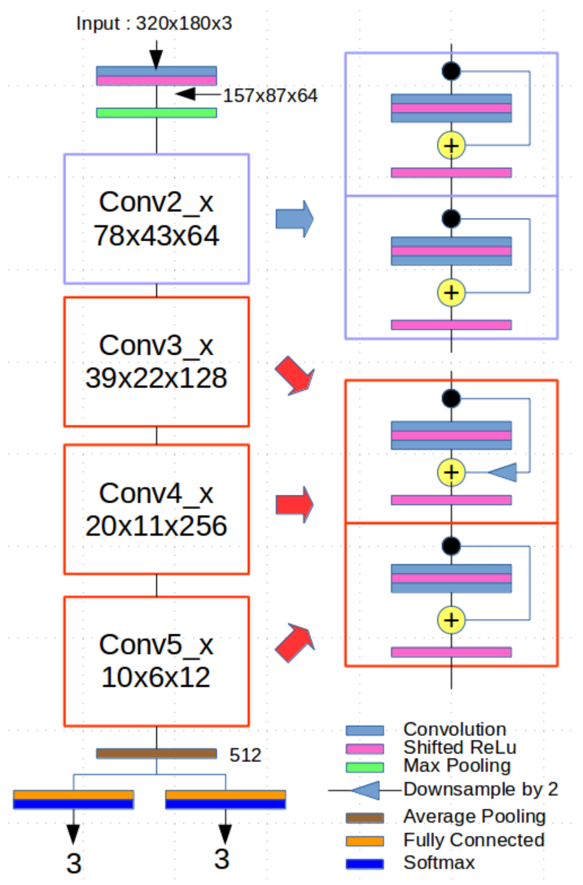


camera rig in which GoPro cameras were mounted on a one-meter bar (distance between adjacent camera was 0.5m). See the rig carried out by Nikolai Smolyanskiy below (from [7])

The rig was fixed on a Segway MiniPro, and data was gathered over several trails by recording video footage of all cameras simultaneously as the Segway was driven along the middle of the trail.

During training, the authors use a typical data augmentation pipeline: random horizontal flips; random contrast, brightness, saturation, sharpness, and jitter with random permutations of these transformations; random scale jitter; random rotations; and random crops.

3.4 DNN architecture

DroNet	TrailNet
 Input : 200x200x1 Batch Normalization Convolution ReLu Max Pooling Fully Connected Drop Out Sigmoid ResBlock1 (32) ResBlock2 (64) ResBlock3 (128) ResBlock 1x1, nb_filters, 2 3x3, nb_filters, 2 3x3, nb_filters 1 1	 Input : 320x180x3 157x87x64 Conv2_x 78x43x64 Conv3_x 39x22x128 Conv4_x 20x11x256 Conv5_x 10x6x12 512 3 3 Convolution Shifted ReLu Max Pooling Downsample by 2 Average Pooling Fully Connected Softmax
DroNet consists of a ResNet-8 [10] with 3 residual blocks, followed by dropout and ReLU non-linearity. Afterwards, the network branches into two separate but fully-connected layers, one to carry out steering prediction and the other to infer collision probability. For each convolution, the notation is the following: first the kernel's size, then the number of filters (provided as a parameter), and eventually the stride if it is different from 1.	The DNN network, referred to as s-ResNet-18, is the standard ResNet-18 [10] architecture, but without batch normalization and with ReLU replaced by shifted ReLU. The first convolution layer is 7×7, whereas all the others are 3×3. Some layers down sample using stride of 2; all others use stride of 1.

3.5 Loss Function and the Optimizer

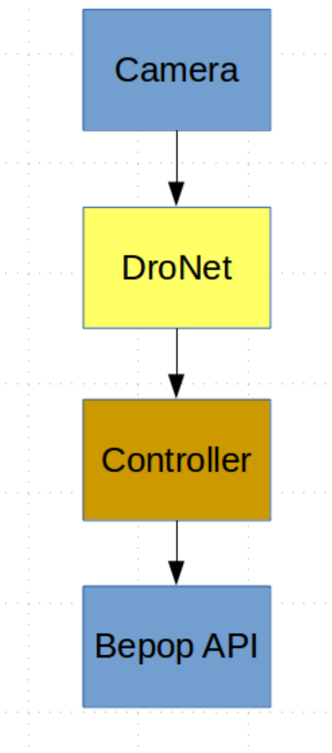
Defining the loss function is not a trivial matter in either case because there are two heads out of the DNN.

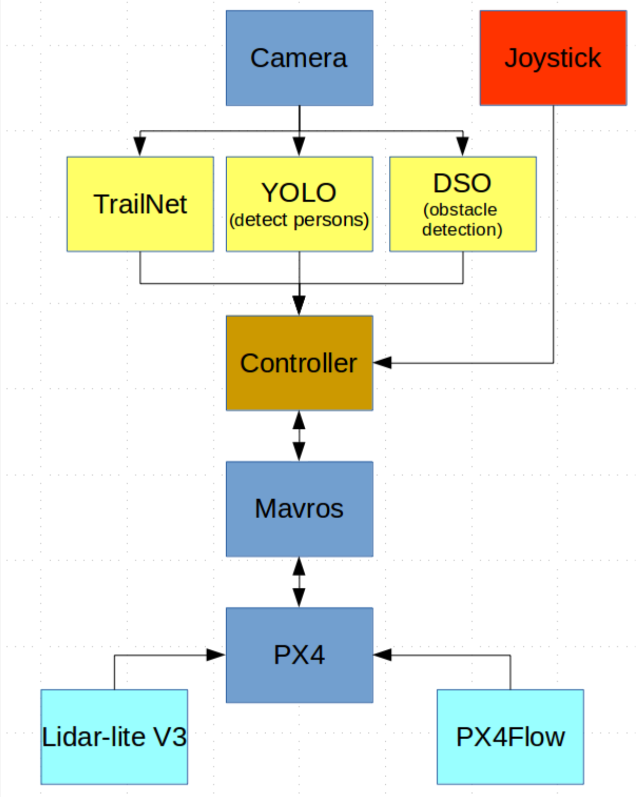
- For Dronet, one head is a binary classification (obstacle detection) and the other is a regression (steering angle).
- For TrailNet, the two heads ended up as a classification problem.

Let's take a look at how the authors define a global loss function which is handling all the parameters.

DroNet	**TrailNet**
The authors use mean-squared error (MSE) to train the steering angle and the binary cross-entropy (BCE) to train the "collision probability." The problem is how to create a join loss function knowing that there is by difference of gradients' magnitudes in the classification (obstacle) and regression (steering) task at the **initial stages of training**. Indeed, the gradients from the regression task are initially much larger, since the MSE gradients' norms are proportional to the absolute steering error. The idea is to give more and more weight to the classification loss in later stages of training, and once losses' magnitudes are comparable, the optimizer will try to find a good solution for both at the same time. The training is done jointly, but the weights for the two losses depend on time. At the beginning more importance is placed on the steering, then attention is turned toward training the collision part as well. This is a form of curriculum learning that has provided very good results. The global loss function is defined as: $L_{tot} = L_{MSE} + max\left(0, 1 - \exp^{-decay.\left(epoch - epoch_0\right)}\right) L_{BCE}$ with decay = 0.1 and epoch$_0$ = 10. Of course, the training set must be organized accordingly. The Adam optimizer is used with a starting learning rate of 0.001 and an exponential per-step decay equal to 0.00001.	The authors treat the problem as one of classification with soft labels, and they defined the loss function as follows: $L = -\sum_i p_i \ln(y_i) - \lambda_1 \left(-\sum_i y_i \ln(y_i)\right) + \lambda_2 \phi(y)$ The first term of the loss function is referred to as the "gross entropy term," the second is the "entropy reward term," and the last one is the "side swap penalty." p_i is the smoothed ground truth label, y_i is the network prediction, $i \in \{L, C, R\}$, y is a 3-element vector containing all y_i , λ_1 , λ_2 are scalar weights, and the side swap penalty penalizes gross mistakes (e.g., swapping left and right) and is only used for the lateral offset head. As you can see, the loss function takes only three parameters into account! In fact, at the training stage, the network is trained with the IDSIA Swiss Alps trail dataset to classify the orientation view. Once it is done, the s-ResNet-18 and the head that classifies the orientation view are frozen. Next, the "lateral view" dataset is used to train the second head that classifies the lateral positions. The same loss function is used in both cases, just note that the side swap penalty is set to 0 for the orientation view.

3.6 System Architecture – Implementing the Whole Thing on the Drone

DroNet	TrailNet
the DroNet architecture is pretty straightforward;it sends the steering angle and the probability of the collision to a controller, which is computing the linear and angular velocities of the drone. Dronet has been developed with Keras framework, the controller on C++, and both Dronet and the controller are running on ROS. The authors use a Bepop from Parrot.  	Regarding the architecture of the system, the authors describeall the different components in details. These include: • The DNN ; • Yolo [11] for object/person detection ; • DSO-monocular SLAM for obstacle detections and avoidance [12]; • The controller, which is the module in charge of prioritizing the command that will be sent to the flight control unit, namely the PX4, via Mavros; • PX4flow [13], which computes the optical flow and deduces the x and y position of the drone; • Lidar-lite V3 in charge of measuring the altitude (z); • A joystick acting like a safeguard allowing an operator to take over the control of the drone in case it goes in the wrong direction or comes too close to obstacles. TrailNet has been developed with Caffe and the controller in C++. All these modules are running in real-time, at different rates, on an NVIDIA Jetson TX1. Together, these systems use 80% of the CPU (4 cores) and 100% of the GPU, when running DSO at 30 Hz, TrailNet at 30 Hz, and YOLO at 1 Hz. It is important to note that DSO and the DNNs are complementary to each other, since the former uses the CPU, while the latter use the GPU.

	This architecture will sound familiar to researchers working with ROS and PX4. The authors made their own drone from scratch.  Camera Joystick TrailNet YOLO (detect persons) DSO (obstacle detection) Controller Mavros PX4 Lidar-lite V3 PX4Flow

3.7 Time to Control the Drone

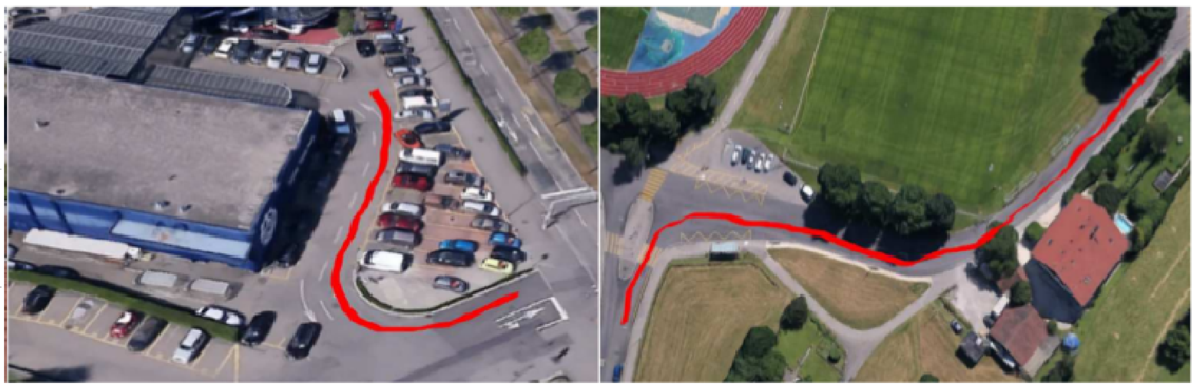
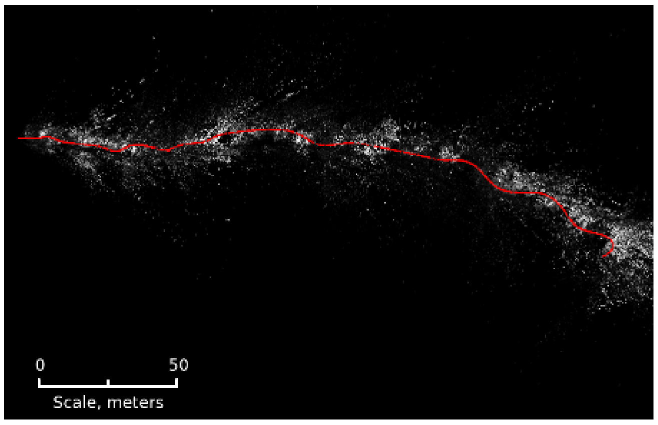
The DNN outputs values for the different parameters that have to be translated in commands (linear/angular velocities, waypoint or more complex trajectories such as splines or Bezier curves) to move the drone.

DroNet	TrailNet
There are two distinct parameters: A “collision probability” p_t that will be used to compute a linear forward velocity. The authors propose a low pass filter: $v_k = v_{k-1} \times (1-\alpha) + V_{max} \times (1-p_t)$ $0 \leq \alpha \leq 1$ A steering parameter s_k that will be used to	TrailNet issues six parameters: - three for the view orientation $(y_L^{vo}, y_c^{vo}, y_R^{vo})$ - three for the lateral offset $(y_L^{lo}, y_C^{lo}, y_R^{lo})$ The authors compute a steering angle α which is the weighted sum of differences between the right and left predictions of the view orientation (vo) and lateral offset (lo): $\alpha = \beta_1 \times (y_R^{vo} - y_L^{vo}) + \beta_2 \times (y_R^{lo} - y_L^{lo})$

compute the desired yaw angular velocity θ_k angular value with a low pass filter: $\theta_k = \theta_{k-1} \times (1-\beta) + \frac{\pi}{2} \times \beta \times s_k$ The linear and angular velocities are then sent via the Bepop API to the low-level controller.	Once the turn angle is known, a new waypoint located some distance in front of the drone and at a turn angle relative to the drone's body frame is computed. As the distance to the waypoint affects the vehicle's speed, the orientation of the drone is set to face the new waypoint direction. The waypoint is converted to the inertial frame and sent (with a fixed distance to the ground) to the MAVROS module for PX4 flight plan execution.

3.8 Let's Go to the Field!

Results are impressive; they show that supervised deep learning can be applied successfully to some very specific tasks.

DroNet
 (a) Outdoor 2 (b) Outdoor 3 DroNet has been tested on different outdoor environments. The pictures above are two examples. The first one (a) is a sharp 160 degree curve followed by a 30 m straight path, for a total length of 68 meters. The second one (b) is a series of two curves, each approximately 60 degrees with straight paths in between, for a total path length of 245 meters.
TrailNet


The picture above is a top-down view of a 250-meter drone trajectory (red) through the forest with 3D map overlaid (gray dots), generated by DSO SLAM. The forest trail was composed of several 250 m zigzags with six turns. The trail was around 1.5 m wide. TrailNet successfully managed a 1 km zigzag forest trail over hilly terrain, as shown in this **video.**

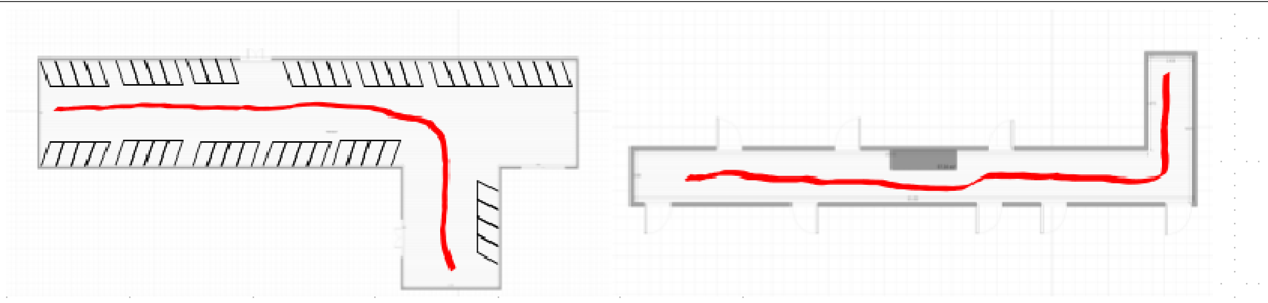
It's unlikely that the DNN architectures presented in the papers were the first bet of the authors. They probably tried several different DNN architectures, tried different loss functions, ran the training set, noticed that the results were more or less convincing, and iterated until they came up with what were presented in the research articles. Next, they likely worked on the hardware, transferred the code on an embedded computer, and finally "passed the field test." Respect for the authors of these two experiments for their achievements.

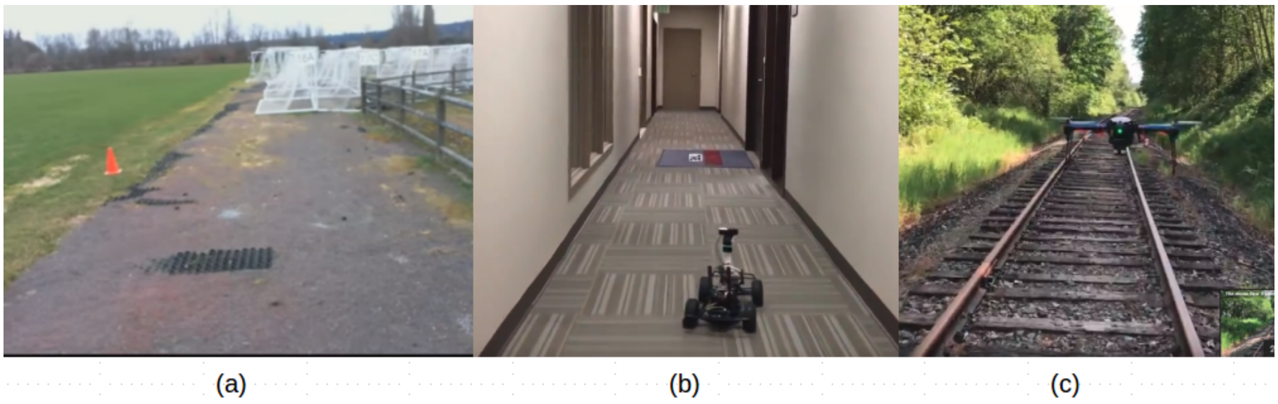
3.9 The Magic of DNN: Generalization

To quote Judea Peal in a recent interview related to his new book [14]:

"As much as I look into what's being done with deep learning, I see they're all stuck there on the level of associations. Curve fitting. That sounds like sacrilege, to say that all the impressive achievements of deep learning amount to just fitting a curve to data. From the point of view of the mathematical hierarchy, no matter how skillfully you manipulate the data and what you read into the data when you manipulate it, it's still a curve-fitting exercise, albeit complex and nontrivial."

As long as the new environment contains "features" that have been encoded by the CNN during the training phase, the probability that generalization occurs is quite high.

DroNet
 (a) Indoor Parking Lot (b) Indoor Parking Lot Without tuning, DroNet allows a drone to fly in indoor parking lots (a) and corridors (b).
TrailNet

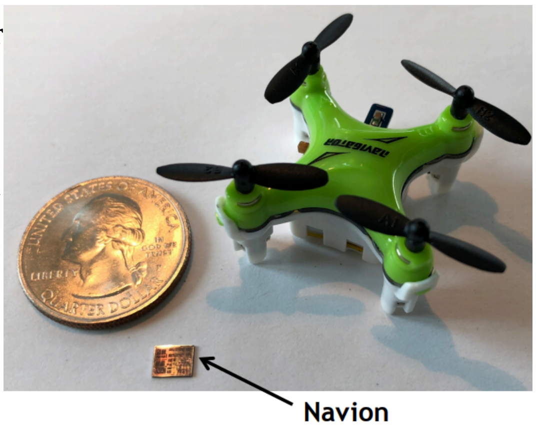


TrailNet was able to follow (a) an open road, (b) hallways (when mounted on a wheeled car), or (c) railroad (with some fine tuning) – you can see more examples in this **video**.

3.10 Algorithm and Hardware Co-Design Approach

The title of this section is taken from an article written by researchers from MIT [15] who implemented a Visual-Inertial Odometry (VIO) on a chip that allows a pico UAV of 10gr to fly autonomously in a GPS denied environment.

At 10 gr, you imagine the constraints in terms of processing power and energy consumption!

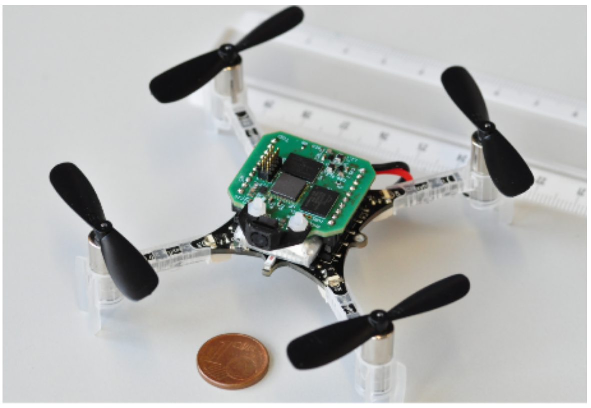


The authors claim that scaling down VIO to miniaturized platforms—without sacrificing performance—requires a paradigm shift in the design of perception algorithms, and they advocate a co-design approach in which algorithmic and hardware design choices are tightly coupled.

They proposed to call this paradigm NAVION, and they created a dedicated web page for it [16].

In September 2018, the MIT Team announced that they have produced a chip 4x5mm, 2mW consumption which runs a full VIO pipeline (see photo from [17]).

The authors of the DroNet article wanted to go one step further and implement DroNet in a miniature drone, namely the crazyFly [18], and they had to follow the same path as the MIT team. They did not conceive a brand new ship on their own but chose GAP8, a commercial embedded RISC-V processor derived from the **PULP open source project**. At its heart, GAP8 is composed by an advanced RISC-V microcontroller unit coupled with a programmable eight-core accelerator for digital signal processing and embedded deep inference; however, they did rethink how to perform the convolution calculations, play with the registers, how to parallelized the computation on the eight



cores, and how to optimize the code with very limited memory resources (512KB internal + 128Mb external but with a bandwidth of 333Mb/s).

Finally, they managed to run DroNet up to 12 frames per second. The drone technology community is likely eagerly awaiting the upcoming video to see how it flies.

4 Preliminary Conclusion

These experiments act as proof of concept that a drone can navigate safely in an indoor/outdoor environment just by using camera inputs.

We are still in the early stages of research in this area, but it looks very promising. One vital question which has been raised is this: how far can we go in applying machine learning to drones? Some researchers in reinforcement learning say that we can do all or "end-to-end training." According to these researchers, instead of having a machine learning engine that issues parameters translated into high-level commands for the low-level controllers, you can provide thrust values directly to control the motors. So far, "end-to-end training" is not working well, and we will see why in Part 2.

It seems that the best compromise consists in applying machine learning where it performs best and keeping the controller in charge of the stability of the drone. Following this principle, the Robotics and Perception Groups ETH Zurich were able to develop a vision-based drone racing in dynamic environments [19], and they even won the IROS 2018 Autonomous Drone Race Competition [20]

To make one final remark before closing, it is interesting to note that there is no need to accurately compute the position of the drone nor a precise map of the environment in the experiments described.

To continue exploring this topic, navigation and reinforcement learning will be covered in Part 2.

5 Bibliography

[1] Grzonka, S., Grisetti, G., & Burgard, W. (2009). Towards a navigation system for autonomous indoor flying. *2009 IEEE International Conference on Robotics and Automation*, 2878-2883.

[2] Shen, S., Michael, N., & Kumar, V. (2011). Autonomous multi-floor indoor navigation with a computationally constrained MAV. *2011 IEEE International Conference on Robotics and Automation*, 20-25.

[3] Bachrach, A., He, R., & Roy, N. (2009). Autonomous Flight in Unstructured and Unknown Indoor Environments.

[4] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen (2017). Autonomous aerial navigation using monocular visual-inertial fusion. J. Field Robot., vol. 00, pp. 1–29.

[5] Vidal, A.R., Rebecq, H., Horstschaefer, T., & Scaramuzza, D. (2018). Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High-Speed Scenarios. *IEEE Robotics and Automation Letters, 3*, 994-1001.

[6] Loquercio, A., Maqueda, A.I., del-Blanco, C.R., & Scaramuzza, D. (2018). DroNet: Learning to Fly by Driving. *IEEE Robotics and Automation Letters, 3*, 1088-1095. **Code and dataset**, **video**

[7] Smolyanskiy, N., Kamenev, A., Smith, J., & Birchfield, S.T. (2017). Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4241-4247. **Code**, **Video**.

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

[9] Udacity, "An Open Source Self-Driving Car," https://www.udacity.com/

self-driving-car, 2016.

[10] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778.

[11] Redmon, J., Divvala, S.K., Girshick, R.B., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788.

[12] Engel, J., Koltun, V., & Cremers, D. (2018). Direct Sparse Odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 40*, 611-625.

[13] Honegger, D., Meier, L., Tanskanen, P., & Pollefeys, M. (2013). An open source and open hardware embedded metric optical flow CMOS camera for indoor and outdoor applications. *2013 IEEE International Conference on Robotics and Automation*, 1736-1741.

[14] https://www.quantamagazine.org/to-build-truly-intelligent-machines-teach-them-cause-and-effect-20180515/

[15] Zhang, Z., Suleiman, A., Carlone, L., Sze, V., & Karaman, S. (2017). Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach. *Robotics: Science and Systems*.

[16] http://navion.mit.edu/

[17] Suleiman, A., Zhang, Z,. Carlone, L,. Karaman, S,., & Sze, V. (2018). Navion: An Energy-Efficient Visual-Inertial Odometry Accelerator for Micro Robotics and Beyond, Hot Chips-30.

[18] Palossi, D., Loquercio, A., Conti, F., Flamand, E., Scaramuzza, D., & Benini, L. (2018). Ultra Low Power Deep-Learning-powered Autonomous Nano Drones. *CoRR, arXiv :1805.01831*.

[19] Kaufmann, E., Loquercio, A., Ranftl, R., Dosovitskiy, A., Koltun, V., & Scaramuzza, D. (2018). Deep Drone Racing: Learning Agile Flight in Dynamic Environments. *CoRR, arXiv :**1806.08548***, **Video**.

[20] Kaufmann, E., Gehrig M., Foehn P., Ranftl R., Dosovitskiy, A. , Koltun, V., & Scaramuzza, D. (2018). Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing. CoRR, arXiv :1810.06224

Kim, D.K., & Chen, T. (2015). Deep Neural Network for Real-Time Autonomous Indoor Navigation. *CoRR, **a**rXiv :1511.04668*.