Agile autonomy: learning tightly-coupled perception-action for high-speed quadrotor flight in the wild

Loquercio, Antonio

University of Zurich[UZH]

Department of Informatics

Agile Autonomy: Learning Tightly-Coupled Perception-Action for High-Speed Quadrotor Flight in the Wild

Dissertation submitted to the Faculty of Business,
Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Antonio Loquercio
from Naples, Italy

approved in July 2021

at the request of
Prof. Dr. Davide Scaramuzza
Prof. Dr. Angela Schoellig
Prof. Dr. Pieter Abbeel
Prof. Dr. Roland Siegwart

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, 21.07.2021

Chairman of the Doctoral Board: Prof. Dr. Thomas Fritz

To my advisor,
who showed me what robots can learn.
&
To my family and friends,
who showed me what robots cannot learn.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Davide Scaramuzza, who accepted me as a PhD student and provided me with many exciting opportunities and advice over the years. I would also like to particularly thank my colleague Elia Kaufmann, without whom the majority of the work in this thesis would not exist. Elia always supported me and helped me to transform my crazy ideas into reality.

I wish to express my gratitude to all the current and past members, visitors, and students who I encountered during my journey as a PhD student at the Robotics and Perception Group. I would particularly like to thank Matthias Fässler, Henri Rebecq, Davide Falanga, Titus Cieslewski, Zichao Zhang, Philipp Föhn, Mathias Gehrig, Daniel Gehrig, Manasi Muglikar, Yunlong Song, Giovanni Cioffi, Nico Messikommer, Jeff Delmerico, Suseong Kim, Guillermo Gallego, Dario Brescianini, Dimche Kostadinov, Javier Hidalgo Carrio, Christian Pfeiffer, Julien Kohler, Thomas Längle, Manuel Sutter, Kevin Kleber, Kunal Shrivastava, Stefano Ghidoni, Rubén Gómez Ojeda, Naveen Kuppuswamy, Timo Stoffregen, Francisco Javier Perez Grau, Bianca Sangiovanni, Yuto Suebe, Roberto Tazzari, Sihao Sun, Ana Maqueda, and Tamar Tolcachier. I also had the pleasure to work with great students, namely Mario Bonsembiante, Lorenzo Ferrini, Simone Arreghini, Alessandro Saviolo, Francesco Milano, Daniel Mouritzen, Bojana Nenezic, Yawei Ye, Mattia Segu, Christoph Meyer, Simon Muntwiler, Moritz Zimmermann.

Furthermore, I was fortunate enough to work with great international collaborators, namely Yanchao Yang, Stefano Soatto, Alexey Dosovitskiy, Rene Ranftl, Matthias Muller, Vladlen Koltun, Daniele Palossi, Francesco Conti, Eric Flamand, Luca Benini, Antoni Rosinol, and Luca Carlone. I would like to thank the agencies funding my research, namely Intel, the National Centre of Competence in Research (NCCR) Robotics, the Swiss National Science Foundation, and the European Research Council.

I would like to thank Prof. Angela Schoellig, Prof. Pieter Abbeel, and Prof. Roland Siegwart for accepting to review my thesis.

Last but not least, I am very grateful to Noëmi, my family, and my friends without whom this work would make no sense.

Zurich, April 2020 *Antonio Loquercio*

Abstract

"The robot revolution has arrived", says the stunning title of a 2020 article in the National Geographic. Over the course of the last decades, autonomous robots have had a tremendous impact on our global economy. Indeed, machines now perform all sorts of tasks: they take inventory and clean floors in big stores; they shelve goods and fetch them for mailing in warehouses; they patrol borders; and they help children with autism. In the majority of those applications, they co-exist and interact with their surroundings, being that humans or other autonomous systems, to create an ecosystem that maximizes efficiency and productivity.

Yet, a major challenge for autonomous systems to operate in unconstrained settings is to cope with the constant variability and uncertainty of the real world. This challenge currently limits the application of robots to structured environments, where they can be closely monitored by expensive suites of sensors and/or human operators. One way to tackle this challenge is exploiting the synergy between perception and action which characterizes natural and artificial agents alike. For embodied agents, action controls the amount of information coming from sensory data, and perception guides and provides feedback to action. While this seamless integration of sensing and control a fundamental feature of biological agents [91], I argue that a tight perception-action loop is fundamental in enriching the autonomy of artificial agents.

My thesis investigates this question in the context of high-speed agile quadrotor flight. Given their agility, limited cost, and widespread availability, quadrotors are the perfect platform to demonstrate the advantages of a tightly coupled perception and action loop. To date, only human pilots can fully exploit their capabilities in unconstrained settings. Autonomous operation has been limited to constrained environments and/or low speeds. Having reached human-level performance on several tasks, data-driven algorithms, *e.g.* neural networks, represent the ideal candidate to enhance drones' autonomy. However, this computational tool has mainly proven its impact on disembodied datasets or in very controlled conditions, *e.g.* in standardized visual recognition tasks or games. Therefore, exploiting their potential for high-speed aerial robotics in a constantly changing and uncertain world poses both technical and fundamental challenges.

This thesis presents algorithms for tightly-coupled robotic perception and action in unstructured environments, where a robot can only rely on on-board sensing and computation to accomplish a mission. In the following, we define a policy as tightly-coupled if it directly goes from on-board sensor observations to actions without intermediate steps. Doing so creates a synergy between perception and action, and it enables end-to-end optimization to the downstream task. To design such policy, I explore the possibilities of data-driven algorithms. Among others, this thesis provides contributions to achieving high-speed agile quadrotor flight in the wild, pushing

the platform closer to its physical limits than what was possible before. Specifically, I study the problem of navigation in natural and man-made (hence in the wild) previously unknown environments with only on-board sensing and computations (neither a GPS nor external links are available to the robot). To achieve this goal, I introduce the idea of sensing and action abstraction to achieve knowledge transfer between different platforms, *e.g.* from ground to aerial robots, and domain, *e.g.* from simulation to the real world. This thesis also presents novel ways to estimate uncertainty in neural network predictions, which open the door to integrating data-driven methods in robotics in a safe and accountable fashion. Finally, this thesis also explores the possibilities that the embodied nature of robots brings to tackle complex perception problems, both for low-level tasks such as depth estimation, or high-level tasks (*e.g.* moving object detection). Overall, the contributions of this work aim to answer the question *Why learning tightly-coupled sensorimotor policies in robotics?* In the following is a list of contributions:

- The introduction of data-driven algorithms to the problem of high-speed agile quadrotor flight in unstructured environments with only on-board sensing and computation.
- A novel architecture for sensorimotor control of a drone, which is trained only with data collected from ground platforms, *e.g.* cars and bicycles. This method allows navigation in both indoor and outdoor environments and is robust to dynamic obstacles.
- A theoretically motivated method based on abstraction to transfer knowledge between platforms and domains, *e.g.* from simulation to the physical world.
- The application of this method to train deep sensorimotor policies for drone racing, acrobatics, and navigation in the wild, demonstrating that a policy trained entirely in simulation can push robots close to their physical limits.
- A general and theoretically motivated framework to estimate the uncertainty of neural network predictions, which is demonstrated on several perception and control tasks.
- A framework for unsupervised training of neural networks on the task of moving object detection. The framework only uses video data and its statistical properties to predict moving objects.
- A novel neural architecture and an associated training procedure to learn 3D perception from monocular image data. The architecture is trained with only the supervision that a robot would observe while exploring a scene: images and very sparse depth measurements.

List of Contributions

Journal Publications

- **Antonio Loquercio**, Elia Kaufmann, Rene Ranftl, Matthias Mueller, Vladlen Koltun, and Davide Scaramuzza. "Agile Autonomy: Learning High-Speed Flight in the Wild". *Science Robotics* (2021).
  Links: Appendix G

- **Antonio Loquercio**, Alexey Dosovitskiy, and Davide Scaramuzza. "Learning Depth with Very Sparse Supervision". In: *IEEE Robotics and Automation Letters (RA-L)* (2020).
  DOI: 10.1109/LRA.2020.3009067
  Links: PDF, Code Appendix E

- **Antonio Loquercio**, Mattia Segu, and Davide Scaramuzza. "A General Framework for Uncertainty Estimation in Deep Learning". In: *IEEE Robotics and Automation Letters (RA-L)* (2020).
  DOI: 10.1109/LRA.2020.2974682
  Links: PDF, Video, Code, Appendix D

- **Antonio Loquercio**, Elia Kaufmann, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. "Deep Drone Racing: From Simulation to Reality with Domain Randomization". In: *IEEE Transactions on Robotics (T-RO)* (2019).
  DOI: 10.1109/TRO.2019.2942989
  Links: PDF, Code Video, Appendix C

- Daniele Palossi, **Antonio Loquercio**, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. "A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones". In: *IEEE Internet of Things (IoT)* (2019).
  DOI: 10.1109/JIOT.2019.2917066
  Links: PDF, Code Video

- **Antonio Loquercio**, A. Maqueda, C. del-Blanco, and D. Scaramuzza. "DroNet: Learning to Fly by Driving". In: *IEEE Robotics and Automation Letters (RA-L)* (2018).
  DOI: 10.1109/LRA.2018.2795643
  Links: PDF, Code, Video Appendix A

Peer-Reviewed Conference Papers

The * symbol indicates shared first authorship.

- **Antonio Loquercio**, and Davide Scaramuzza. "Agile Autonomy: High-Speed Flight with On-Board Sensing and Computing". In: *Conference on Robotics and Intelligent Machines (I-RIM3D)* (2020). Links: PDF, Video Pitch

- Francesco Milano, **Antonio Loquercio**, Antoni Rosinol, Davide Scaramuzza, Luca Carlone. "Primal-Dual Mesh Convolutional Neural Networks". In: *Conference on Neural Information Processing Systems (NeurIPS)* (2020). Links: PDF, Code

- Yunlong Song, Selim Naji, Elia Kaufmann, **Antonio Loquercio**, and Davide Scaramuzza. "Flightmare: A Flexible Quadrotor Simulator". In: *Conference on Robotic Learning (CoRL)* (2020).
Links: PDF, Video Website

- Nico Messikommer, Daniel Gehrig, **Antonio Loquercio**, and Davide Scaramuzza. "Event-based Asynchronous Sparse Convolutional Networks". In: *IEEE European Conference on Computer Vision (ECCV)* (2020).
DOI: 10.1007/978-3-030-58598-3_25
Links: PDF, Video, Code

- Elia Kaufmann*, **Antonio Loquercio***, Rene Ranftl, Matthias Mueller, Vladlen Koltun, and Davide Scaramuzza. "Deep Drone Acrobatics". In: *Robotics: Science, and Systems (RSS)* (2020).
DOI: 10.15607/RSS.2020.XVI.040
Links: PDF, Video, Code, Appendix F

- Daniel Gehrig, **Antonio Loquercio**, Konstantinos G. Derpanis, and Davide Scaramuzza "End-to-End Learning of Representations for Asynchronous Event-Based Data". In: *IEEE International Conference on Computer Vision (ICCV)* (2019).
DOI: 10.1109/ICCV.2019.00573
Links: PDF, Video, Code

- Yanchao Yang*, **Antonio Loquercio***, Davide Scaramuzza, and Stefano Soatto. "Unsupervised Moving Object Detection via Contextual Information Separation".
In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).
DOI: 10.1109/CVPR.2019.00097
Links: PDF, Video, Code, Appendix B,

- Elia Kaufmann*, **Antonio Loquercio***, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. "Deep Drone Racing: Learning Agile Flight in Dynamic Environments". In: *Conference on Robotic Learning (CoRL)* (2018). Links: PDF, Code Video

- Ana Maqueda, **Antonio Loquercio**, Guillermo Gallego, Narciso Garcia, and Davide Scaramuzza. "Event-based Vision meets Deep Learning on Steering Prediction for Self-driving Cars". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).
DOI: 10.1109/CVPR.2018.00568
Links: PDF, Video

- Yawei Ye, Titus Cieslewski, **Antonio Loquercio**, and Davide Scaramuzza. "Place Recognition in Semi-Dense Maps: Geometric and Learning-Based Approaches". In: *British Machine Vision Conference (BMVC)* (2017).
DOI: 10.5244/C.31.74
Links: PDF, Poster

Open-source Software

- DroNet: Learning to Fly by Driving

- Unsupervised Moving Object Detection via Contextual Information Separation
- Deep Drone Acrobatics
- Deep Drone Racing:From Simulation to Reality with Domain Randomization
- A General Framework for Uncertainty Estimation in Deep Learning

Awards

- **IEEE TRO King-Sun Fu Memorial Best Paper Award (Honorable Mention), 2020.**
- **IRIM-3D Best Paper Award (Honorable Mention), 2020.**
- **RSS Best Paper Award (Honorable Mention), 2020.**
- **CORL Best System Paper Award, 2018.**

Contents

1 Introduction

This thesis presents algorithms for tightly-coupled robotic perception and action. According to this paradigm, in an iterative and infinite loop, action controls the amount of information coming from sensory data, and perception guides and provides feedback to action. Ecological psychology showed that the seamless integration of sensing and control is a fundamental feature of biological agents [91]. Can also artificial and embodied agents benefit from a tightly-coupled perception and action loop? My work addresses this question in the context of high-speed agile quadrotor flight.

In the following, we define as tightly-coupled any policy which directly predicts actions from (an history of) sensor observations, without explicit intermediate blocks.[1] Data-driven algorithms, *e.g.* neural networks, represent the ideal candidate to represent tightly-coupled policies: data-driven methods have recently achieved human-level performance in a series of standardized visual recognition and decision-making tasks. For example, the combination of supervised learning and large-scale datasets resulted in very high-performance systems for tasks like image classification [108, 119], segmentation [39], and detection [231]. Similarly, the combination of reinforcement learning algorithms with large-scale computing enabled the creation of systems that outperform humans in Atari games [196] and the complex game of Go [262]. However, all the previous tasks have a common denominator: they operate on a disembodied dataset or in controlled conditions, and they have an explicitly defined *score* function (a label is either wrong or correct in case of visual recognition, an action either decreases or increases the player's score in games). In general, such assumptions cannot be satisfied when a robot operates in unstructured and possibly dynamic environments. Therefore, while data-driven methods have the potential to enhance robots' perception and control, their application to robotics poses a unique set of fundamental and technical challenges. Specifically:

- How can robots collect enough data to learn the complex mapping between noisy sensor measurements and motor skills?
- How can we trasfer knowledge between different domain or tasks?

[1] Different research communities have associated different, and potentially ambiguous, meanings to the term tightly coupled. For instance, in state estimation [309], it refers to the problem of planning actions that maximize the visibility of landmarks during navigation.

Figure 1.1 – A tightly-coupled perception-action loop, powered by the combination of traditional model-based robotics and state-of-the-art machine learning tools, enables autonomous high-speed operation in unstructured environments. The quadrotor is required to reach a waypoint 50 meters in front of it given by a user. The environment is not known in advance, and the robot has only access to on-board sensing and computation (no GPS or external tracking systems are used).

- How can we verify and validate the safety of a deep sensorimotor policy controlling a robot?
- How can the interactive and embodied nature of robots be used to learn complex perception tasks?

To address these questions, my work combines theories and methods from machine learning, computer vision, and robotics, along with inspiration from cognitive science and psychology.

As a common demonstrator, I will consider the problem of high-speed quadrotor navigation in previously unknown real-wold environments (hence in the wild) with only on-board sensing and computation (only a camera and an IMU are available to the agent). While previous work mainly considered manipulators or disembodied agents [316, 315, 160, 222, 3], quadrotors are the perfect test-bed to demonstrate the advantages of a tightly coupled perception and action loop. To date, only expert human pilots have been able to fully exploit their capabilities. Autonomous operation with onboard sensing and computation has been limited to low speeds. Indeed, agile quadrotor flight in unstructured environments requires low latency, robustness to perception disturbances, *e.g.* motion blur, and high precision, since the slack to avoid a collision is extremely limited. These characteristics push the boundaries of current state-of-the-art perception and control systems. My research shows that directly mapping sensory observations to navigation commands is a key principle to enable high-speed navigation. By mastering such a challenging

navigation task, this work presents compelling evidence that the coupling of sensing and control is a fundamental step towards the development of a general-purpose robot autonomy in the physical world.

This thesis is split into three parts. First, given the difficulties to collect training data with quadrotors in the real world, I introduce an approach to transfer knowledge between domains, *e.g.* from simulation to reality. I also demonstrate the applicability of this method for a set of tasks, namely drone racing, acrobatic flight, and navigation of unstructured and dynamic environments. Second, I present a general algorithm to decode the uncertainty of neural network predictions, providing a way to integrate neural networks into safety-critical robotics systems as drones. Finally, I exploit the interactive nature of robots to train policies without any external or explicit supervision.

All parts address the driving research question of this work - *Why learning tightly-coupled perception and action policies in robotics?* - from a different perspective but with the same goal: building the next generation of autonomous robots. To facilitate training and development, I argue that future navigation systems will strongly rely on simulation. Therefore, effective methods are required to transfer knowledge between the simulated and real domain (Part I). In addition, for being successfully integrated into safety critical robotics systems and accepted by both governmental agencies and the general public, end-to-end policies need to automatically detect possible failure cases, *e.g.* due to noise in the data of lack of sufficient training data (Part II). Finally, to adapt to the conditions they observe during deployment and cannot be anticipated or modeled in simulation, robots will need to adapt their knowledge in the absence or with very limited supervisor signal. In such cases, self-supervised cues, *e.g.* visual and temporal consistency, can provide enough learning signal to train or tune even complex perception policies (Part III).

This thesis is structured in the form of a collection of papers. An introductory section that highlights the concepts and ideas behind the thesis is followed by self-contained publications in the appendix. Section 1.1 states and motivates the research objectives of this work. Section 1.2 places this research in the context of the related work. Chapter 2 summarizes the papers in the appendix and their connections with respect to each other. Finally, Chapter 3 provides future research directions.

1.1 Motivation

During the last decade, commercial drones have been disrupting industries ranging from agriculture to transport, security, infrastructure, entertainment, and search and rescue. In contrast to their grounded counterparts, flying robots can cover large distances in a very limited amount of time, which makes them an essential tool in missions where a fast response is crucial. Among commercial drones, quadrotors are the most agile: thanks to their agility they can navigate through complex structures, fly through damaged buildings, and reach remote locations inaccessible to

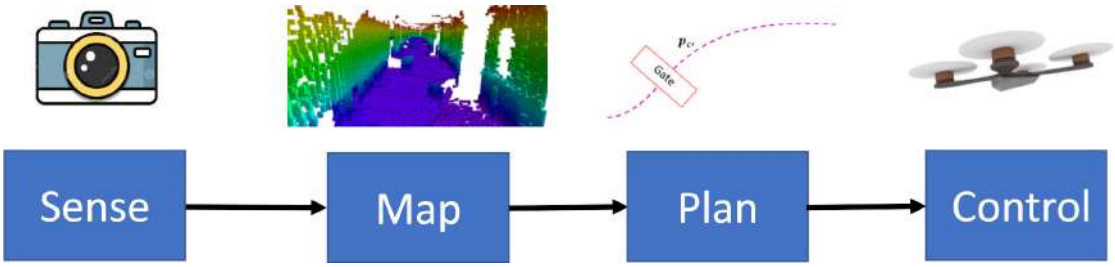


Figure 1.2 – The majority of existing methods for navigation in previously unknown environments divides the navigation problem into a series of consecutive subtasks. While this process is attractive from an engineering prospective and favours interpretability, it completely neglects the interactions between the different stages and thus compound errors.

other robots.

However, developing fully autonomous quadrotors that can approach or even outperform the agility of birds or human drone pilots in unknown and cluttered environments is very challenging and still unsolved. Changing lighting conditions, perceptual aliasing (i.e. strong similarity between different places), and motion blur, all common during agile flight in unstructured environments, are well-known and unsolved problems of state-of-the-art perception systems [214, 73, 180, 59]. In addition, the uncertainties coming from approximate models of the environment, sensors, and actuators, along with the requirements of closed-loop safety and a low latency perception-action loop add a significant layer of complexity to the problem of high-speed navigation.

Current state-of-the-art works addressed agile quadrotor navigation by splitting the task into a series of consecutive subtasks: perception, map building, planning, and control. This approach, illustrated in Figure 1.2, constitutes the foundation of almost all current robotic systems, from autonomous cars to drones. The division of the navigation task into sequential subtasks is attractive from an engineering perspective since it enables parallel progress on each component and makes the overall system interpretable. However, it leads to pipelines that largely neglect interactions between the different stages and thus compound errors [309]. These errors are generated by sensor noise, *e.g.* motion blur or missing pixels, and have both a systematic and aleatoric nature. While the aleatoric part can only be accounted for in expectation, the systematic nature can be completely addressed. However, engineering robustness to all possible failure cases of a sensor is difficult to impossible. Therefore, noisy perception leads to imperfect mapping, suboptimal planning, and inaccurate control. In addition, the sequential nature of the blocks also introduces additional latency, which can be deterimental for agile flight [64]. While these issues can be mitigated to some degree by careful hand-tuning and engineering, the divide-and-conquer principle that has been prevalent in research on autonomous flight in unknown environments for many years imposes fundamental limits on the speed and agility that a robotic system can achieve [170].

In contrast to these traditional pipelines, I propose a seamless and tight integration of perception

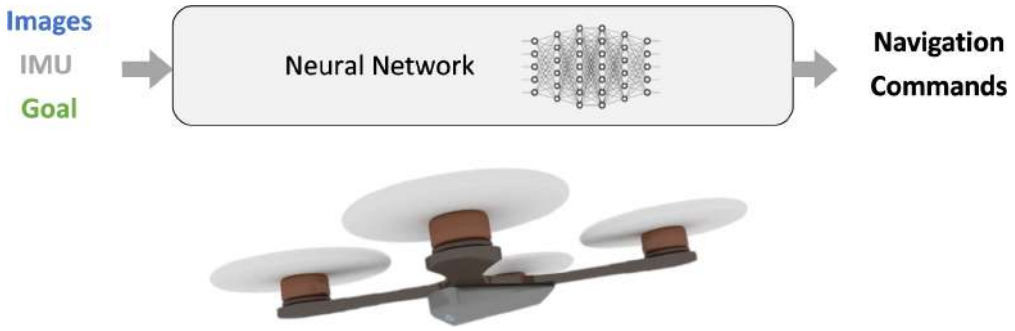


Figure 1.3 – Conversely to traditional methods, my thesis proposes a tight integration between sensing and control via a deep sensorimotor policy. This tight perception-action loop decreases the latency between perception and action and improves robustness against perception artifacts, *e.g.* motion blur or sensor noise. However, it also comes with several challenges, *i.e.* sample complexity, interpretability, and generalization to new environments and conditions, which this thesis aims to address.

and action through a deep sensorimotor policy (Figure 1.3). This holistic approach comes with a series of advantages and challenges with respect to its traditional counterpart, which I describe in the next section.

1.1.1 Advantages

A tight perception-action loop, powered by advanced learning-based algorithms, has a series of advantages.

Low Latency. We define the *sensor-action latency* as the time required for a sensor measurement to influence the action executed by a robot. The action can be at a different level of abstraction, and it can vary from low-level motor commands to high-level velocity or position commands. However, the higher the action level, the longer it takes to transform it into motor commands. In the case of a deep sensorimotor policy, this latency generally corresponds to the time of a neural network forward pass. Thanks to the continuously evolving technology on hardware acceleration for deep networks, the latency of a sensorimotor policy is now in the order of milliseconds on on-board hardware, even for large architectures, and up to 10 times lower than the one of a sequential system (Paper G). Having a low latency enables a fast control loop, which is fundamental, for instance, to promptly react to unexpected obstacles. In addition, the combination of deep sensorimotor policies and hardware acceleration enables closed-loop control of nanorobots with extremely low power and computational budget [217].

Robustness to Imperfect Perception. All sensors have nonidealities. For example, a depth sensor operating on stereo images produces observations with discretization artifacts, missing data, or erroneous measurements (c.f. Figure 1.4). Similarly, an image recorded from a camera can be corrupted by motion-blur or artifacts caused by the low dynamic range or the image

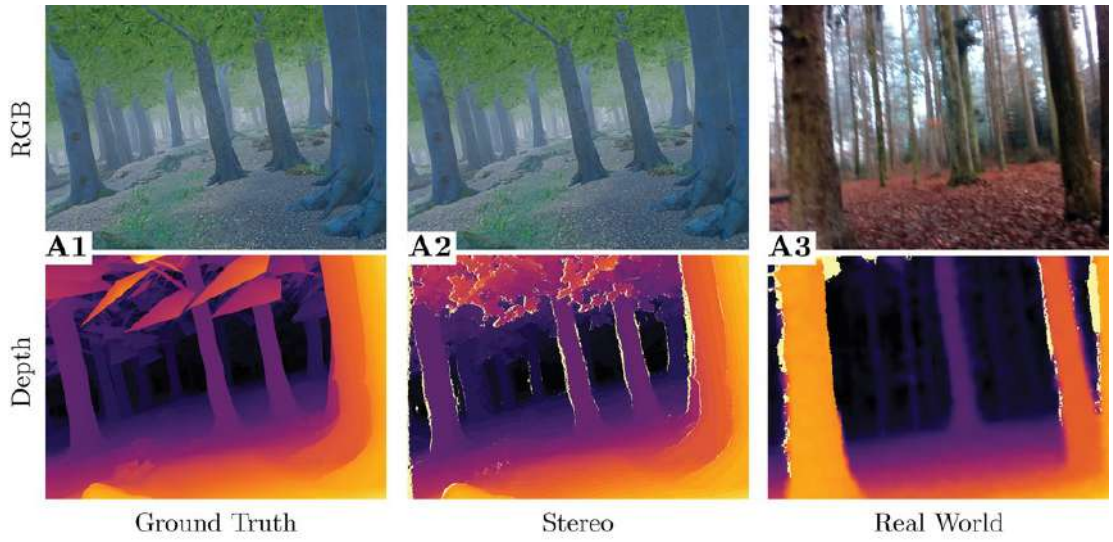


Figure 1.4 – The non-idealities of sensor observations can strongly affect performance in traditional pipelines since those errors compound over the pipeline. In contrast, a tight perception-action cycle learns to cope with the systematic nature of those errors by leveraging regularities in the data. Learning those regularities does not necessarily require training in the real world: for instance, a simulated depth estimated by stereo matching (**A2**) contains the typical failure cases of a depth sensor (**A3**), *e.g.* missing values and noise.

capturing method (*e.g.* global or local shutter). When provided with enough training data, deep sensorimotor policies become robust to perception nonidealities. Indeed, such nonidealities generally have an epistemic, *i.e.* systematic, nature, which a neural network learns to leverage by using regularities in the data. Directly identifying systematic errors and engineering solutions to them is a very challenging task. For this reason, traditional methods are generally more prone to failures due to compounding errors through the pipeline.

Ease to Develop and Deploy The fast-growing scale of machine learning research pushed the development of many open-source tools for the creation and validation of learning-based systems. These tools are not only easy to use on normal computers but are sometimes even optimized for computationally constrained on-board hardware [217]. In addition, the machine learning community has strongly promoted an open-source philosophy aimed at reproducibility, and it spent large efforts to build simulators [266, 257, 55, 78] and datasets [46, 52, 166], which are all readily available to the machine learning practitioner. Conversely, traditional methods are significantly more diverse and specialized than their learning counterparts and often rely on the end-user to both develop the low-level functionalities and optimize them on hardware. Despite this limitation being not fundamental (large companies have hundreds of experienced engineers to develop their systems), it strongly affects the speed at which new ideas can be implemented and tested on a physical system.

1.1.2 Challenges

Despite their numerous advantages over traditional sequential methods, learning-based systems present several challenges that need to be overcome to unlock their full potential for computer vision and robotics. The main challenge arises from their data complexity: a sufficiently large and representative dataset is required to successfully train a deep sensorimotor policy for a specific task. While domain knowledge can help to decrease the data complexity, it is not necessarily clear how to inject such prior knowledge into the training process. In addition, contrary to traditional methods, which are transparent and easy to interpret, learning-based systems are often black-boxes and therefore difficult to interpret and debug. Thus, guaranteeing performance during deployment is challenging – especially for algorithms that are expected to cope with the inherent uncertainties of the real world.

Sample Complexity Machine learning algorithms are data-hungry. Popular datasets for training networks on standard visual recognition tasks generally have training samples in the order of millions. When the task is closed-loop control of a robot the data complexity can be even larger, since it is necessary to expose the system to all the possible situations it could encounter during operation (*i.e.* to sufficiently explore the state space). Building such training datasets is a tedious and expensive process, but also raises the problem of how to collect enough "negative" experiences (*e.g.* very close to obstacles). This region of the state space is generally far from the positive (*i.e.* safe) states observed by an expert (*e.g.* a human pilot) and challenging to observe and sufficiently represent in a dataset. Addressing this problem is one of the core endeavors of my thesis. I propose to tackle this problem by recycling data collected from different platforms (Paper A) or by training the sensorimotor policies entirely in simulation, as shown in Figure 1.5 (Papers C, F, G). Doing so waives the requirement of a human expert to provide demonstrations, it cannot harm the physical system during training, and it can be used to learn behaviors that are challenging even for the best human pilots. In Section 2.1, I show how it is possible to enable knowledge transfer between different platforms, *e.g.* a car and a drone, or different domains, *e.g.* simulation and the real world.

Use of Domain Knowledge. Over the years, roboticists have developed a large knowledge base in how to model [169] and control [22] complex systems with high accuracy. Using this knowledge to decrease the data complexity and to favor generalization to new environments and tasks is a major opportunity for learning-based methods. However, how can this knowledge be injected into a deep sensorimotor policy? In this thesis, I propose two different ways to address this problem: (i) use domain knowledge to build experts, generally operating on *privileged* information, that can supervise the training of the sensorimotor policy (Papers F, C, and G), and (ii) to make the sensorimotor student operate on high-level control commands, *e.g.* a desired trajectory, which is then executed by a low-level model-based controller (Papers C, G, and A). Both approaches simplify the training procedure with respect to model-free approaches, in which the laws of physics need to be learned from scratch.

Figure 1.5 – Learning the complex mapping between noisy sensor observations and actions generally requires large and diverse training datasets. Collecting these datasets in the real world is a tedious and expensive procedure, that also raises fundamental questions on how to sufficiently cover the state space to enable closed-loop performance. To address this problem, I propose to train sensorimotor policies entirely in simulation and then transfer the acquired knowledge to a physical platform via domain-invariant abstraction.

Interpretability. When operating a physical robot, wrong decisions could not only fail the mission but also put human lives at risk, *e.g.* if the robot is an autonomous car or a medical device. Traditional methods, generally composed of sequential modules tackling different sub-tasks, can naturally be interpreted, tested, and inspected for the identification of errors. In contrast, deep sensorimotor policies, directly mapping noisy sensor observations to actions, do not offer a straightforward way to be interpreted or validated. In addition, neural network predictions are known to be unreliable when the inputs are out of the training distribution [80]. To tackle this problem, in Section 2.2 I present a general framework for uncertainty estimation in deep neural network predictions. Having a measure of uncertainty enables the detection of sensor failure, out-of-distribution samples, and gives valuable feedback during closed-loop control [174].

Generalization. To build a general-purpose robot autonomy, artificial agents need to develop skills to quickly adapt to new environments and conditions, while efficiently learning new tasks in a handful of trials. However, current robot automation systems, whether or not data-driven, suffer from a limited generalization. Indeed, simply changing the operation environment, but keeping the task definition unchanged, might require intensive re-training or fine-tuning of the system. To improve robustness to visual changes in the environment, I have proposed to use domain adaption (Paper C and G). In addition, to enable adaption, I have developed techniques to train from unsupervised or weakly-supervised data, which can be collected during operation and directly leverage the interactive nature of robots (Papers B and E). However, generalization remains one of the open challenges of robotics and, in my opinion, holds the key to a new era of general-purpose robot autonomy in the real world.

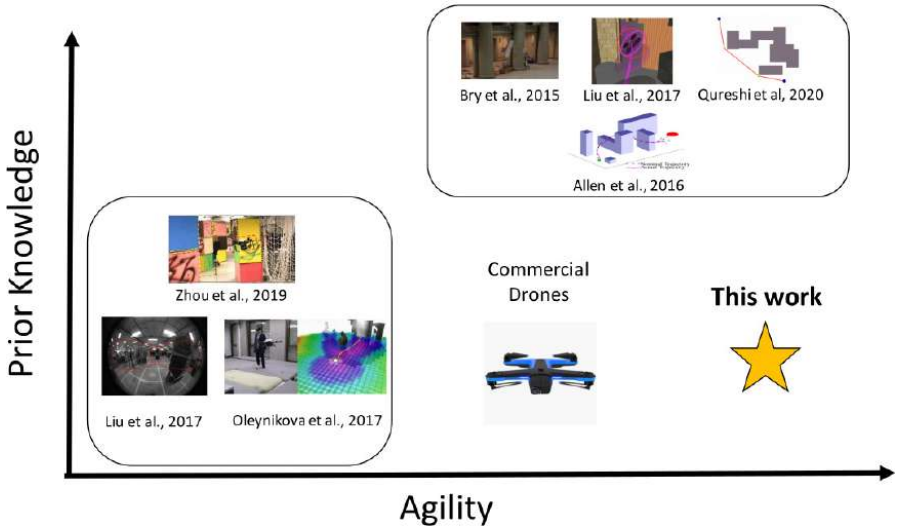


Figure 1.6 – Overview of existing approaches for drone navigation in challenging and cluttered environments. While state-of-the-art approaches make strong assumptions in terms of prior knowledge or are only limited to low speeds, the primary goal of this thesis is to develop methods that can exploit the agility of quadrotors with no prior knowledge about the environment or external sensing.

1.2 Related Work

This section summarizes the state of the art in perception and control for agile drone navigation. It also indicates how the work of this thesis relates to and improves on state-of-the-art methods.

1.2.1 Autonomous Drone Navigation in Unknown Environments

Autonomous, vision-based drone navigation gives rise to fundamental challenges for both perception and control. On the perception side, motion blur, challenging lighting conditions, and aliasing can cause severe drift in vision-based state estimation [73, 206, 180]. Other sensory modalities, e.g. LIDAR or event-based cameras, could partially alleviate these problems [27, 238]. Those sensors are however either too bulky or too expensive to be used on small quadrotors. Moreover, state-of-the-art state estimation methods are designed for a predominantly-static world, where no dynamic changes to the environment occur.

From the control perspective, plenty of work has been done to enable high-speed navigation, both in the context of autonomous drones (Figure 1.7) [179, 1, 201, 200] and autonomous cars [154, 134, 140, 245]. Lupashin et al. [179] propose iterative learning of control strategies to enable platforms to perform multiple flips. Abbeel et al. [1] learn to perform a series of acrobatic maneuvers with autonomous helicopters. Their algorithm leverages expert pilot demonstrations to learn task-specific controllers. Similarly, Li et al. [162] propose an imitation learning approach for training visuomotor agents for the task of quadrotor flight. While these works proved the ability to fly machines to perform agile maneuvers, they did not consider the perception problem.

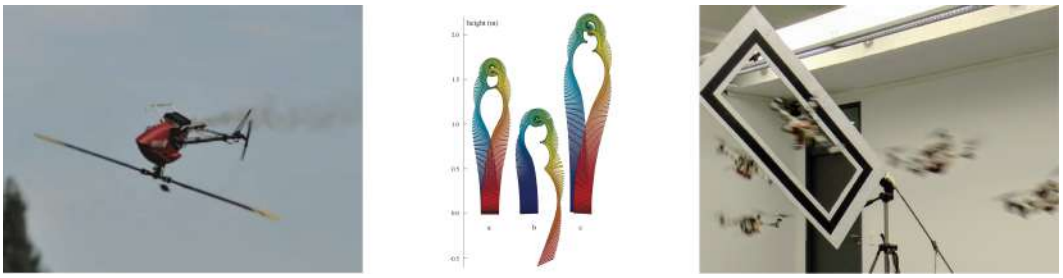


Figure 1.7 – While prior works can exploit the ability of flying machines to perform agile maneuvers, they assume that near-perfect state estimation is available during the maneuver. In practice, this entails disregarding the challenges of perception and instrumenting the environment with dedicated sensors.

Indeed, they assume that near-perfect state estimation is available during the maneuver, which in practice requires instrumenting the environment with dedicated sensors.

Agile flight with on-board sensing (camera and IMU) and computation is a challenging research problem, especially for navigation in previously unknown real-world environments. The first attempts in this direction were made by Shen et al. [260], who demonstrated agile vision-based flight. The work was limited to low-acceleration trajectories, therefore only accounting for part of the control and perception problems encountered at high speed. More recently, Loianno et al. [171] and Falanga et al. [66] demonstrated aggressive flight through narrow gaps with only onboard sensing. Even though these maneuvers are agile, they are very short and cannot be repeated without re-initializing the estimation pipeline. Overall, these works are specifically tailored to the operation environment, *e.g.* a narrow opening [66, 171], and cannot be used for general navigation of previously unknown environments.

We will now discuss approaches that have been proposed to overcome the aforementioned problems. A taxonomy of prior works is presented in Figure 1.6.

Traditional Methods. Various approaches to enable autonomous flight have been proposed in the literature. Some works tackle only perception and build high-quality maps from imperfect measurements [109, 79, 250, 62, 19], while others focus on planning without considering perception errors [28, 233, 5, 168]. Numerous systems that combine online mapping with traditional planning algorithms have been proposed to achieve autonomous flight in previously unknown environments [214, 215, 197, 15, 311, 244, 284, 42, 309].

Oleynikova *et al.* [214] propose a computationally efficient method - Voxblox - to build a map of the environment with associated obstacle distance information. This mapping scheme is combined with a trajectory optimization method to navigate a drone in a cluttered environment with only on-board computation. However, this approach is only demonstrated at relatively low speeds ($1\,\mathrm{m\,s^{-1}}$) and in combination with off-board sensing (near-perfect state estimation coming from a Vicon tracking module), which restricts its applications to controlled environments

(Figure 1.8).

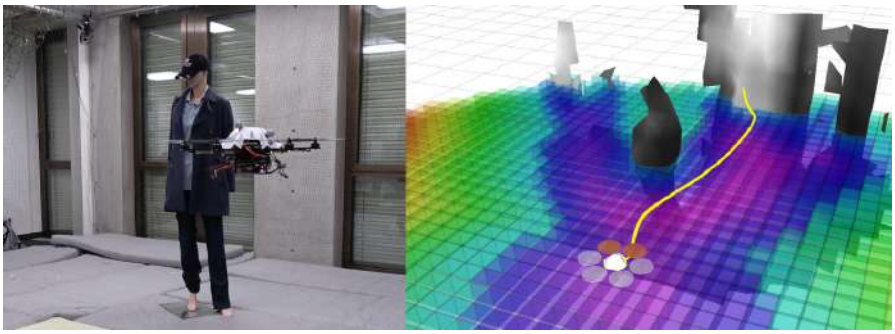


Figure 1.8 – A quadrotor navigating to a point behind the mannequin by combining a voxblox-generated map with a local re-planning trajectory optimization-based method. All computation runs entirely on-board, but state sensing is off-board and generated by an external tracking module.

In the context of the DARPA Fast Lightweight Autonomy (FLA) program, Mohta *et al*. [197] developed a system for fast and robust aerial robot autonomous navigation in cluttered, GPS-denied environments (Figure 1.9). The system assumes little to no prior knowledge of the scene and relies on on-board sensing and computation for state estimation, control, mapping, and planning. Navigation commands are predicted by a system by combining a lidar-based mapping module and an A^* planner module, and they are executed with a cascade of LQR controllers. However, the system requires bulky sensors, *e.g*. Lidar, it is very computationally expensive and prone to compounding errors over the pipeline. Such requirements pose a limit on the agility of the drone, preventing high-speed agile navigation.

(a) (b)

Figure 1.9 – The system proposed by Mohta *et al*. can navigate previously unknown environments with only on-board sensing and computation (a). However, it relies on a bulky computational and sensing unit (b), which limits the agility and speed of the system.

Zhou *et al*. [311] introduced a robust and perception-aware replanning framework to support fast and safe flight, which currently constitutes the state-of-the-art for vision-based agile navigation (Figure 1.10). Its key contributions are a path-guided optimization approach to efficiently find feasible and high-quality trajectories and a perception and risk-aware planning strategy to actively observe and avoid unknown obstacles. This approach demonstrated agile drone navigation in previously unknown cluttered environments with only on-board vision-based perception and

(a) (b)

Figure 1.10 – The system proposed by Zhou *et al*. enables agile flight in indoor (a) and outdoor (b) environments. However, it is limited to static environments and relatively low speeds. In addition, its sequential nature makes it subject to sensing errors, typical during agile flight, which compound over the pipeline.

computation. However, this approach is still limited to static environments, and it only achieves maximum speeds of $3\,\mathrm{m\,s^{-1}}$, thus not exploiting the full agility of drones.

Overall, tfhe division of the navigation task into the mapping and planning subtasks is attractive from an engineering perspective, since it enables parallel progress on each component and makes the overall system interpretable. However, it leads to pipelines that largely neglect interactions between the different stages and thus compound errors [309]. Their sequential nature also introduces additional latency, making high-speed and agile maneuvers difficult to impossible [64]. While these issues can be mitigated to some degree by careful hand-tuning and engineering, the divide-and-conquer principle that has been prevalent in research on autonomous flight in unknown environments for many years imposes fundamental limits on the speed and agility that a robotic system can achieve [170].

Data-Driven Methods. In contrast to these traditional pipelines, some recent works propose to learn navigation policies directly from data, abolishing the explicit division between mapping and planning (Figure 1.11). These policies are trained by imitating a human [240] or an algorithmic expert [308], or directly from experience collected in the real world [84]. Zhang et al. [308] train a deep sensorimotor policy directly from the demonstrations provided by an MPC controller. While the latter has access to the full state of the platform and knowledge of obstacle positions, the policy only observes on-board sensor measurements, *i.e*. laser range finder, and inertial measurements. Gandhi *et al*. [84] predict the collision probability with a neural network trained with real-world data and command the drone towards the direction minimizing the collision risk. As the number of samples required to train general navigation policies is very high, existing approaches impose constraints on the quadrotor's motion model, for example by constraining the platform to planar motion [308, 84, 240] and/or discrete actions [240, 84], at the cost of reduced maneuverability and agility.

Figure 1.11 – To overcome the limitations of traditional navigation pipelines, previous work proposed to learn complete navigation policies directly from data, collected either by a human (left) [240] or an automated procedure (right) [84]. However, to address the policies' sample complexity, they imposed strong motion-constraints, *e.g.* planar motion, which do not exploit the agility of the platform.

Transfer from Simulation to Reality. Learning navigation policies from real data has a shortcoming: the high cost of generating training data in the physical world. Data needs to be carefully collected and annotated, which can involve significant time and resources. To address this problem, a recent line of work has investigated the possibility of training a policy in simulation and then deploying it on a real system (Figure 1.12). Work on the transfer of sensorimotor control policies has mainly dealt with manual grasping and manipulation [99, 299, 23, 127, 243, 247]. In the context of drone control, Sadeghi and Levine [246] learned a collision-avoidance policy by using 3D simulation with extensive domain randomization [279]. Despite this policy being able to control a drone in the real world, the proposed model-free training procedure requires constraining the actuation space, *i.e.* discrete actions, to converge. Indeed, the policy only achieves very low speeds and cannot profit from the agility of the drone.

In driving scenarios, synthetic data was mainly used to train perception systems for high-level tasks, such as semantic segmentation and object detection [235, 129]. One exception is the work of Müller et al. [202], who propose a novel strategy based on abstraction to transfer a policy trained in simulation on a physical ground vehicle. Specifically, they propose to train a control policy from abstract representations, *e.g.* semantic segmentation, instead of the raw sensory inputs, *e.g.* color images. This training strategy strongly simplifies the transfer problem, since the abstract representations are more similar between domains than the raw sensor measurements. Indeed, they empirically show that their policy trained in simulation generalizes to multiple environments and lighting conditions.

In Paper F, I derive the theoretical foundations of simulation to reality via abstraction using information theory [2]. The resulting framework is then validated on multiple tasks including drone racing (Paper C), acrobatic flight (Paper F), and high-speed flight in the wild (Paper G).

Figure 1.12 – Instead of training with real-world data, prior work explored the possibility to train entirely in simulation and then transfer to the real platform using abstractions of inputs (left) [202] or domain randomization (right)[246]. Yet, these transfer procedures require careful tuning of abstraction/randomization parameters and constraint the platforms to relatively slow motion in 2D.

Simulators The landscape of currently available simulators for quadrotor control is fragmented (Figure 1.13): some are extremely fast, *e.g.* Mujoco [280], while others have either really accurate dynamics [120, 78] or highly photo-realistic rendering [98]. Both RotorS [78] and Hector [152] are popular Micro Aerial Vehicle (MAV) simulators built on Gazebo [150], which is a general robotic simulation platform and generally used with the popular Robot Operating System (ROS). These gazebo-based simulators have the capability of accessing multiple high-performance physics engines and simulating various sensors, ranging from laser range finders to RGB cameras. Nevertheless, Gazebo has limited rendering capabilities and is not designed for efficient parallel dynamics simulation, which makes it unattractive for the development of learning-based systems. AirSim [257] is an open-source photo-realistic simulator for drones built on Unreal Engine. Despite being very photorealistic, AirSim has tight integration between the physics and the rendering module. This is a disadvantage for two reasons: (i) the physics engine is not specialized for racing quadrotors and cannot be easily modified, and (ii) it strongly limits the simulation speed if images are not required. This limitation makes it difficult to apply the simulator to challenging model-free reinforcement learning tasks. FlightGoggles [98] is a photo-realistic sensor simulator for perception-driven robotic vehicles. This simulator is very useful for rendering camera images given trajectories and inertial measurements from flying vehicles in real-world, in which the collected dataset [8] is used for testing vision-based algorithms. However, it has a very limited sensor suite and can only simulate a single quadrotor at a time, limiting the speed at which large-scale datasets can be collected. Eventually, Muller *et al.* [203] built Sim4CV, a photo-realistic quadrotor simulator that is however only aimed at computer vision applications.

During my Ph.D., I have contributed to building a simulator that is specifically tailored to machine learning applications: Flightmare [266]. Flightmare is composed of two main components: a configurable rendering engine built on Unity and a flexible physics engine for dynamics simulation. Those two components are decoupled and can run independently of each other. This makes our simulator extremely fast: rendering achieves speeds of up to 230 Hz, while physics

Figure 1.13 – Several quadrotor simulators are available to develop, train, and test navigation algorithms. While some offer very accurate dynamics (left) [78], others have highly photo-realistic rendering (center) [257]. I have contributed to build Flightmare (right) [266], a simulator which offers both and is specifically tailored to machine learning applications.

simulation of up to 200,000 Hz on a laptop. I have used this simulator for the development of Paper G. Specifically, I have used Flightmare to train a policy that can navigate quadrotors at high speed in cluttered environments. The policy, trained entirely in simulation, was then deployed in multiple indoor and outdoor scenarios in the physical world.

1.2.2 Autonomous Drone Racing

The popularity of drone racing has recently kindled significant interest in the robotics research community (Figure 1.14). The classic solution to this problem is image-based visual servoing, where a robot is given a set of target locations in the form of reference images or patterns. Target locations are then identified and tracked with hand-crafted detectors [276, 66, 161]. Li *et al*. [163] tailor this approach to the problem of drone racing. Specifically, they propose to extract gate information from images and then fuse it with attitude estimates from an inertial measurement unit (IMU) to guide the drone towards the next visible gate. While the approach is computationally very light-weight, it struggles with scenarios where multiple gates are visible and does not allow to employ more sophisticated planning and control algorithms which, e.g., plan several gates ahead. Jung *et al*. [131] retrieve a bounding box of the gate and a line-of-sight-based control law aided by optic flow is then used to steer the drone towards the detected gate. However, these methods cannot cope with the cases when no gate is visible, which is typical during drone racing.

To account for this problem, previous work [137] combines the gate detector with a pre-computed global plan and a visual odometry system with uncertainty estimation and Kalman filtering. Foehn *et al*. [71] extend this idea to account for multiple visible gates and improve the localization and planning scheme. Overall, the above methods strongly rely on the quality of the gate detection system, which can however quickly become unreliable in the presence of occlusions, partial visibility, and motion blur. In addition, the separation between perception, planning, and control may lead to compounding errors over the pipeline, leading to a lack of robustness to perception disturbances, *e.g*. challenging illumination conditions. In Paper C, I show that combining the perception and planning block into a convolutional neural network and leveraging the abundance of simulated data, favors robustness against dynamic environments and perception disturbances.

Figure 1.14 – The classic approach to autonomous drone racing is to detect the next gate and steer the drone towards it (left) [131]. To account for the cases when no gate is visible, it is possible to combine state estimation and gate detection through Kalman filtering (right) [71]. Multi-agent autonomous drone racing also raises interesting game-theoretical problems (center) [267].

1.2.3 Uncertainty Estimation for Safe Deep Learning

To successfully integrate learning methods into robotics, neural networks should reliably estimate the uncertainty in their predictions [275]. In this section, I review existing methods for uncertainty estimation and some of their applications to robotics.

Estimating Uncertainties in Neural Networks Predictions A neural network is generally composed of a large number of parameters and non-linear activation functions, which makes the (multi-modal) posterior distribution of network predictions intractable. To approximate the posterior, existing methods deploy different techniques, mainly based on Bayesian inference and Monte-Carlo sampling (Figure 1.15).

To recover probabilistic predictions, Bayesian approaches represent neural network weights through parametric distributions, e.g., exponential-family [20, 75, 111, 292]. Consequently, networks' predictions can also be represented by the same distributions and can be analytically computed using non-linear belief networks [75] or graphical models [271]. More recently, Wang et al. [292] propose natural parameter networks, that model inputs, parameters, nodes, and targets by Gaussian distributions. Overall, this family of approaches can recover uncertainties in a principled way. However, they generally increase the number of trainable parameters in a super-linear fashion and require specific optimization techniques [111] which limits their impact in real-world applications.

To decrease the computational burden, Gast et al. [87] proposed to replace the network's input, activations, and outputs with distributions, while keeping the network's weights deterministic. Similarly, probabilistic deep state-space models retrieve data uncertainty in sequential data and use it for learning-based filtering [74, 298]. However, disregarding weights uncertainty generally results in over-confident predictions, in particular for inputs not well represented in the training data.

Instead of representing neural network parameters and activations by probability distributions,

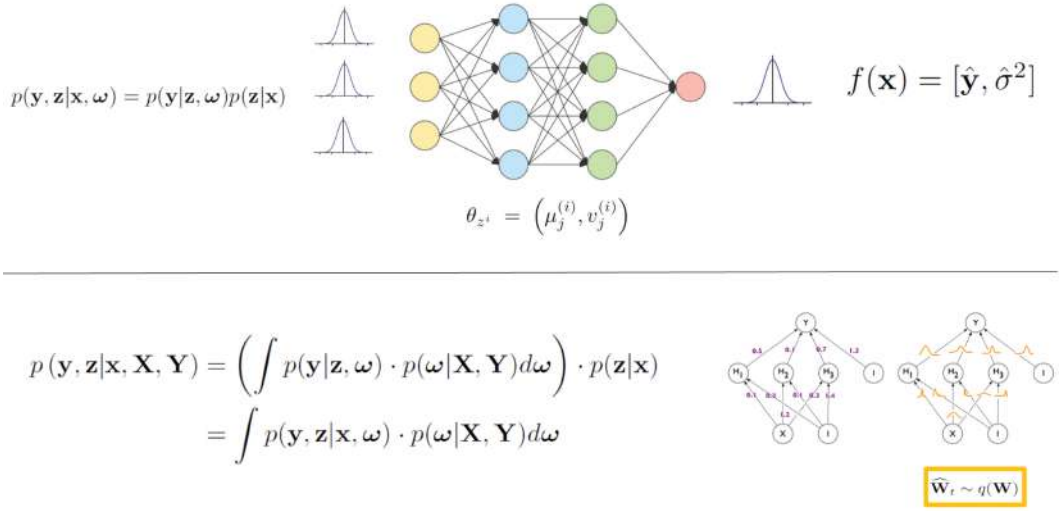


Figure 1.15 – Prediction uncertainty in deep neural networks generally derives from two sources: data uncertainty and model uncertainty. Data uncertainty (above) arises because of noise in the data, usually caused by the sensors' non-idealities. Such uncertainty can be estimated through Bayesian Belief Networks [87] or specialized training [141]. Model uncertainty (below) is generated from unbalances in the training data distribution, and it is generally estimated through sampling [80].

another class of methods proposed to use Monte-Carlo (MC) sampling to estimate uncertainty. The MC samples are generally computed using an ensemble of neural networks. The prediction ensemble could either be generated by differently trained networks [143, 133, 156], or by keeping drop-out at test-time [80]. While this class of approaches can represent well the multi-modal posterior by sampling, it cannot generally represent data uncertainty, due for example to sensor noise. A possible solution is to tune the dropout rates [81], however, it is always possible to construct examples where this approach would generate erroneous predictions [123]. Recent work shows the possibility to estimate model uncertainty without sampling, decreasing the run-time computational budget required for uncertainty estimation [226, 258] However, also these methods do specifically account for the data uncertainty.

To model data uncertainty, Kendall et al. [141] proposed to add to each output a "variance" variable, which is trained by a maximum-likelihood (a.k.a. heteroscedastic) loss on data. Combined with Monte-Carlo sampling, this approach can predict both the model and data uncertainty. However, this method requires changing the architecture, due to the variance addition, and using the heteroscedastic loss for training, which is not always an option for a general task.

Akin to many of the aforementioned methods, in Paper D I use Monte-Carlo samples to predict model uncertainty. Through several experiments, I show why this type of uncertainty, generally ignored or loosely modeled by Bayesian methods [87], cannot be disregarded. In addition to

Figure 1.16 – Estimating uncertainty is of paramount importance to integrate learning-based methods into robotics systems. Kahn *et al*. [133] used model-uncertainty to speed-up the learning process for an autonomous car (left). Kaufmann *et al*. [137] estimated the uncertainty of gate detection to enable bayesian decision making during drone racing (center). Chua *et al*. [41] accounted for model errors during model-based reinforcement learning by leveraging their uncertainty (left).

Monte-Carlo sampling, my approach also computes the prediction uncertainty due to the sensors' noise by using gaussian belief networks [75] and assumed density filtering [24]. Therefore, my approach can recover the full prediction uncertainty for any given (and possibly already trained) neural network, without requiring any architectural or optimization change.

Uncertainty Estimation in Robotics Given the paramount importance of safety, autonomous driving research has allocated a lot of attention to the problem of uncertainty estimation, from both the perception [68, 210] and the control side [143, 133] (Figure 1.16). Feng. et al. [68] showed an increase in performance and reliability of a 3D Lidar vehicle detection system by adding uncertainty estimates to the detection pipeline. Predicting uncertainty was also shown to be fundamental to cope with sensor failures in autonomous driving [143], and to speed-up the reinforcement learning process on a real robot [133].

For the task of autonomous drone racing, Kaufmann et al. [137] demonstrated the possibility to combine optimal control methods to a network-based perception system by using uncertainty estimation and filtering. Also for the task of robot manipulation, uncertainty estimation was shown to play a fundamental role to increase the learning efficiency and guarantee the manipulator safety [269, 41].

In Paper D, I show that uncertainty estimation is beneficial in both perception and control tasks. In perception tasks, I show that neural networks naturally capture the uncertainty of inherently ambiguous tasks, *e.g*. future motion prediction. In the task of closed-loop control of a quadrotor, I demonstrate that identifying possible wrong predictions or out-of-distribution input samples is beneficial to control performance.

1.2.4 Unsupervised and Weakly-Supervised Learning Robot Vision

Robust perceptions are necessary for enabling robots to understand and interact with their surroundings. The majority of state-of-the-art perception algorithms, included some of the methods presented in this thesis, rely on large annotated datasets or specifically designed reward functions. In contrast, natural agents learn (part of) their skills with interaction with the environment. This section explores prior work that tries to invert this paradigm and learn in a more natural and scalable fashion. Specifically, I examine this problem on a low-level task (*i.e.* depth estimation) and a high-level task (*i.e.* moving object detection).

3D Perception The problem of recovering the three-dimensional structure of a scene from its two-dimensional projections has been long studied in computer vision [172, 106, 105, 285]. Classic methods are based on multi-view projective geometry [104]. However, key characteristics of these classic methods are that they crucially rely on projective geometry, require laborious hand-engineering, and are not able to exploit non-motion-related depth cues.

To make optimal use of all depth cues, machine learning methods can either be integrated into the classic pipeline or replace it altogether (Figure 1.17). The challenge for supervised learning methods is the collection of training data: obtaining ground truth camera poses and geometry for large realistic scenes can be extremely challenging. Therefore, while supervised learning methods have demonstrated impressive results [57, 229, 289, 313], it is desirable to develop algorithms that function in the absence of large annotated datasets.

Unsupervised (or self-supervised) learning provides an attractive alternative to label-hungry supervised learning. The dominant approach is inspired by classic 3D reconstruction techniques and makes use of projective geometry and photometric consistency across frames. Among the methods for learning depth maps, some operate in the stereo setup [86, 93], while others address the more challenging monocular setup, where the training data consists of monocular videos with arbitrary camera motions between the frames [314, 182]. Reprojection-based approaches can often yield good results in driving scenarios, but they crucially rely on geometric equations and precisely known camera parameters (one notable exception being the recent work in [95], which learns the camera parameters automatically) and enough textured views.

In Paper E, I show how is possible to train a neural network to predict metric depth with the feedback an agent would observe interacting with the environment: a sequence of RGB frames and very sparse depth measurement (down to one pixel per image). In contrast to geometric unsupervised methods, my approach does not require knowing the camera parameters in advance and is robust in low-textured indoor scenarios.

Several works similar to mine aim to learn 3D representations without explicitly applying geometric equations [277, 232, 60]. However, they build implicit 3D representations, and therefore they cannot be directly used for downstream robotic tasks such as navigation or motion planning. Moreover, at training time it requires knowing the camera pose associated with each

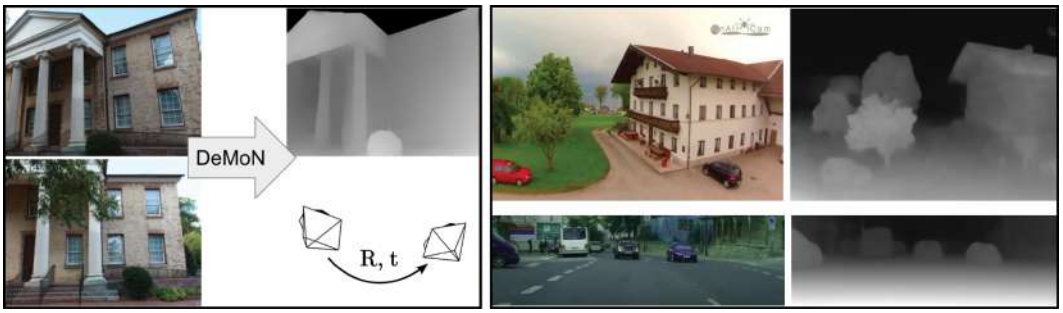


Figure 1.17 – To recover the 3D structure of the scene, state-of-the-art methods train neural networks with large annotated datasets (left) [289], which are expensive and tedious to collect. Self-supervised methods wave the requirement for training datasets by using projective geometry and photometric consistency across frames (right) [95]. However, these methods suffer when the views contain reflective surfaces or are poorly textured.

image. My method, in contrast, does not require camera poses and grounds its predictions in the physical world via very sparse depth supervision. This allows us to learn an explicit 3D representation in the form of depth maps.

Moving Object Detection The ability to detect moving objects is primal for animals and robots alike, so there is a long history of motion-based segmentation, or moving object detection (Figure 1.18). Early attempts to explicitly model occlusions include the layer model [293] with piecewise affine regions, with computational complexity improvements using graph-based methods [261] and variational inference [47, 26, 272, 306] to jointly optimize for motion estimation and segmentation. Ochs *et al*. [212] use long-term temporal consistency and color constancy, making however the optimization more difficult and sensitive to parameter choices. Similar ideas were applied to motion detection in crowds [25], traffic monitoring [17] and medical image analysis [58].

More recent data-driven methods [282, 281, 40, 265] learn discriminative spatio-temporal features and differ mainly for the type of inputs and architectures. Inputs can be either image pairs [265, 40] or image plus dense optical flow [282, 281]. Architectures can be either time-independent [281], or with recurrent memory [282, 265]. Overall, those methods outperform traditional ones on benchmark datasets [212, 223], but at the cost of requiring a large amount of labeled training data and with evidence of poor generalization to previously unseen data.

In Paper B, I present an approach for moving object detection that, similarly to the classic object detection literature, does not need any annotated training data. However, like modern learning methods, my approach can use contextual cues to help prediction, which would be impossible to engineer given the complexity of image formation and scene dynamics.

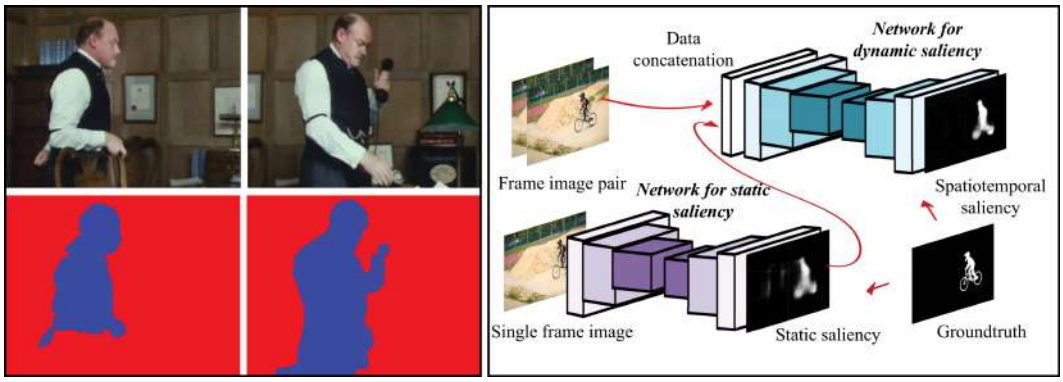


Figure 1.18 – Classical methods for moving object detection relies on super-pixels and color constancy to assign object relationship over time (left) [212]. However, they are generally very sensitive to hyper-parameters or require the solution of differential equations at run time. Conversely, data-driven methods have a very good generalization and outperform traditional ones on public benchmarks (right) [281]. However, they rely on large annotated datasets for training.

2 Contributions

This chapter summarizes the key contributions of the papers that are reprinted in the appendix. It further highlights the connections between the individual results and refers to related work and video contributions.

In total, this research has been published in 4 peer-reviewed conference publications and 5 journal publications (one in the *IEEE Transactions on Robotics*, one in *Science Robotics*, and three in the *Robotics Automation Letters (RA-L)*).

These works led to several research awards and open-source software.

Awards.

- IEEE Transactions on Robotics King-Sun Fu Memorial Best Paper Award (Honorable Mention), 2020
- IRIM-3D Best Paper Award (Honorable Mention), 2020
- RSS Best Paper Award (Honorable Mention), 2020
- CORL Best System Paper Award, 2018

Software.

- Dronet: Learning to Fly by Driving
- Unsupervised Moving Object Detection via Contextual Information Separation
- Deep Drone Racing: From Simulation to Reality with Domain Randomization
- A General Framework for Uncertainty Estimation in Deep Learning
- Deep Drone Acrobatics

2.1 Transfer Learning for Agile Drone Navigation

In this part of the thesis, we consider the problem of short-range navigation for autonomous drones. In this task, the drone has access to only on-board sensor observations from a camera and an IMU, as well as a user-defined reference, indicating the goal or the desired path. This reference is not necessarily collision-free and it generally the output of an higher level planning or exploration algorithm [42]. The task consists of following this reference while possibly avoiding obstacles. Neither prior knowledge about the application environment nor external information, *e.g.* GPS, are available to the agent.

One approach to train neural networks to address this task is to collect a set of human demonstrations and learn a policy from them [1]. However, this approach has some limitations. The most obvious one is related to the cost (both in terms of resources and time) to collect large enough datasets: expert human pilots are a scarce and expensive resource. In addition, data coming from human experts are more saddle than one might expect. How can we collect enough "negative" experiences, far from the positive (*i.e.* safe) states observed by a human pilot? How can we push the neural network to learn to navigate, and not just overfit to the particular scene/situation in which the data was collected?

The work conducted during this thesis contributes to addressing these questions in several ways. First, I developed an approach to train drones to fly in an urban environment by only collecting data from completely different platforms, *e.g.* grounded vehicles (Paper A). This approach was also applied for the control of palm-sized drones [217] in a collaborative effort between me and the Integrated System Lab (ISL) at ETH Zurich. Then, I developed a theoretically sound approach to transfer knowledge between platforms and domains, *e.g.* from simulation to the physical world. Such framework was used to train navigation algorithms for drone racing (Paper C), acrobatics (Paper F), and navigation in the wild (Paper G). These algorithms were trained exclusively in simulation and never saw a real image before being deployed on physical drones. I also contributed to building the infrastructure required to develop such algorithms by developing a high-quality drone simulator [266]. This simulator was openly released to the public to foster progress in the community.

2.1.1 Paper A: Dronet: Learning to fly by driving

(P1) Antonio Loquercio, Ana I. Maqueda, Carlos R. del-Blanco, and Davide Scaramuzza. "DroNet: Learning to Fly by Driving". In: *IEEE Robotics Autom. Lett.* 3.2 (2018), pp. 1088–1095

In unstructured and highly dynamic scenarios drones face numerous challenges to navigate autonomously feasibly and safely. In this work, we explore a data-driven approach to cope with the challenges encountered when flying in urban environments. Specifically, we propose 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. We train this policy from data collected by cars and bicycles, which, already integrated into urban environments, offer the perfect opportunity to collect data in a scalable fashion. This work demonstrates that specifically tailored output representations enable to transfer of knowledge between different platforms and allow to build generalizable sensorimotor policies. Dronet can successfully fly not only at the same height as cars, but also at relatively high altitudes, and even in indoor environments, such as parking lots and corridors.

Related Software

(S1) https://github.com/uzh-rpg/rpg_public_dronet

Related Videos

(V1) https://youtu.be/ow7aw9H4BcA

Figure 2.1 – DroNet is a lightweight sensorimotor policy that can control drones in urban environments. DroNet can follow street rules (left), fly at high altitude (center), and also avoid dynamic obstacles (right). In addition, it also generalizes to indoor environments and parking lots.

DroNet Application to Nano-Drones

The DroNet algorithm which I developed enabled the first demonstration of an autonomous nano-drone capable of closed-loop end-to-end learning-based visual navigation. Building an autonomous system at a nano-scale is very challenging since nano-drones can carry very little

weight, and therefore computation: a full-stack navigation system made of state-estimation, planning, and control is out of reach of the computational budget available at these scales. Therefore, to achieve closed-loop visual navigation, we developed a complete methodology for parallel execution of DroNet directly on-board of a resource-constrained milliwatt-scale processor. Our system is based on GAP8, a novel parallel ultra-low-power computing platform, and a 27 g commercial, open-source CrazyFlie 2.0 nano-quadrotor.

(A1) Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. "A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones". In: *IEEE Internet of Things Journal* (2019)

Related Videos

(V2) https://youtu.be/57Vy5cSvnaA

Related Software

(S1) https://github.com/pulp-platform/pulp-dronet

Related Demonstrations

(D1) Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. "A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones". In: *Demo at IROS* (2018)

Figure 2.2 – PULP-DroNet: a deep learning-powered visual navigation engine that enables autonomous navigation of a pocket-size quadrotor in a previously unseen environment.

(a) (b)

Figure 2.3 – To perceive the environment and navigate to the next gate, I've generated data only with a non-photorealistic simulator. Due to the abundance of such data, generated with domain randomization (a), the trained CNN can be deployed on a physical quadrotor without any finetuning (b).

2.1.2 Paper C: Deep Drone Racing: From Simulation to Reality with Domain Randomization

(P2) Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. "Deep Drone Racing: From Simulation to Reality With Domain Randomization". In: *IEEE Trans. Robotics* 36.1 (2019), pp. 1–14

Dynamically changing environments, unreliable state estimation, and operation under severe resource constraints are fundamental challenges that limit the deployment of small autonomous drones. I have addressed these challenges in the context of autonomous, vision-based drone racing in dynamic environments. A racing drone must traverse a track with possibly moving gates at high speed. I have enabled this functionality by combining the performance of a state-of-the-art planning and control system with the perceptual awareness of a convolutional neural network (CNN). But how to collect enough data to train the CNN? A previous version of this work proposed to manually collect it in the real world [139], but the procedure is tedious and error-prone. Therefore, in this work, I proposed to train the perception system directly in simulation. Using domain randomization, the trained perception system directly generalizes to the real world. The abundance of simulated data makes our system more robust than its counterpart trained with real-world data to changes of illumination and gate appearance.

Related Publications

(R1) Elia Kaufmann*, Antonio Loquercio*, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. "Deep drone racing: Learning agile flight in dynamic environments". In: *Conference on Robot Learning (CoRL)*. 2018

Related Software

(S2) https://github.com/uzh-rpg/sim2real_drone_racing

Related Videos

(V3) https://youtu.be/vdxB89lgZhQ

2.1.3 Paper F: Deep Drone Acrobatics

(P3) Elia Kaufmann*, Antonio Loquercio*, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. "Deep Drone Acrobatics". In: *RSS: Robotics, Science, and Systems* (2020)

Performing acrobatic maneuvers with quadrotors is extremely challenging. Acrobatic flight requires high thrust and extreme angular accelerations that push the platform to its physical limits. In this work, I show that deep sensorimotor controllers trained entirely in simulation can be used for these extremely challenging navigation tasks. Specifically, I propose to learn a sensorimotor policy that enables an autonomous quadrotor to fly extreme acrobatic maneuvers with only onboard sensing and computation. I developed a theoretically sound framework for knowledge transfer between domains and found why appropriate abstractions of the visual input decrease the simulation-to-reality gap. Using this framework, I train a policy in simulation that can be directly deployed in the physical world without any fine-tuning on real data. This approach enables a physical quadrotor to fly maneuvers such as the Power Loop, the Barrel Roll, and the Matty Flip, during which it incurs accelerations of up to 3g

Related Videos

(V4) https://youtu.be/bYqD2qZJlxE

Related Software

(S3) https://github.com/uzh-rpg/deep_drone_acrobatics

Figure 2.4 – A quadrotor performs a Barrel Roll (left), a Power Loop (middle), and a Matty Flip (right). We safely train acrobatic controllers in simulation and deploy them with no fine-tuning (*zero-shot transfer*) on physical quadrotors. The approach uses only onboard sensing and computation. No external motion tracking was used.

2.1.4 Paper G: Agile Autonomy: Learning High-Speed Flight in the Wild

(P4) Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. "Agile Autonomy: Learning High-Speed Flight in the Wild". In: *Science Robotics* 7 (58 2021), pp. 1–12

In this work, we use the same theoretical framework for simulation to real-world transfer developed in Paper F to address one of the key problems in mobile robotics research: navigation in previously unknown cluttered environments with only on-board sensing and computation. Major industry players strive for such capabilities in their products, but current approaches [310, 70, 213] are very limited in the agility and speed they achieve in arbitrary unknown environments. The main challenge for autonomous agile flight in arbitrary environments is the coupling of fast and robust perception with effective planning. State-of-the-art works rely on systems that divide the navigation task into separate modules, i.e. sensing, mapping, and planning. Despite being successful at low speeds, these systems become brittle at higher speeds due to their modular nature, which leads to high latency, compounding errors across modules, and sensitivity to perception failures. These issues impose fundamental limits on the speed and agility that an autonomous drone can achieve. I've departed from this modular paradigm by directly mapping noisy sensory observations to navigation commands with a specifically designed convolutional neural network. This holistic paradigm drastically reduces latency and is more robust to perception errors by leveraging regularities in the data. Similarly to the work of Paper F, the approach was trained exclusively in a simplistic simulation environment. Controlled experiments in simulation show that the proposed method decreases the failure rate up to ten times in comparison to prior works. Without any adaptation, the learned controller was used to fly a real quadcopter in different challenging indoor and outdoor environments at very high speeds.

Related Videos

(V5) https://youtu.be/uTWcC6IBsE4

Figure 2.5 – Samples of man-made and natural environments where our sensorimotor policy has been deployed. All environments are previously unknown. The navigation policy was trained entirely in simulation and uses only onboard sensing and computation.

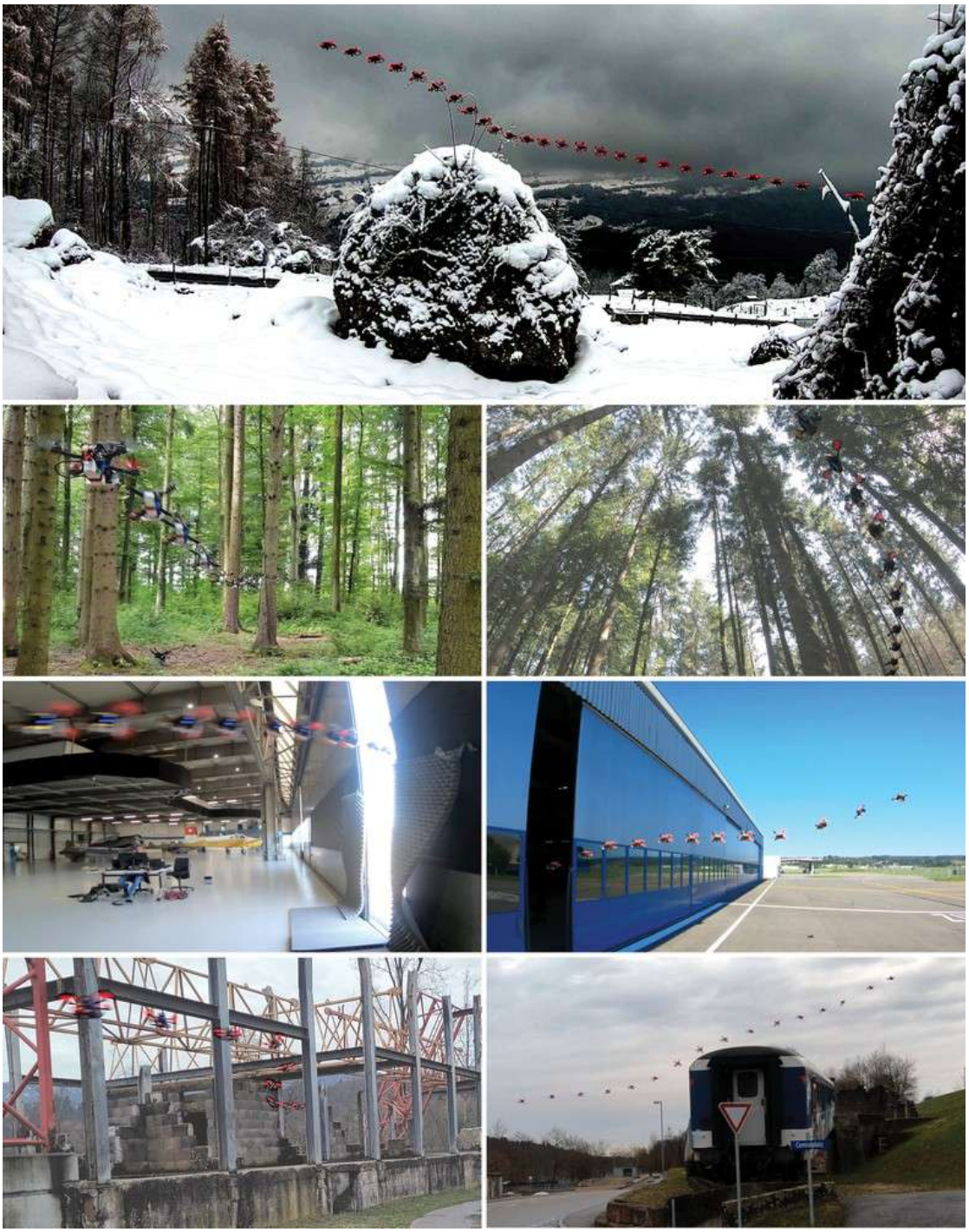


Figure 2.6 – Deployment of the proposed approach in a set of challenging environments. To get a better sense of the speed and agility of the autonomous system, please watch the supplementary video.

2.1.5 Limitations of Transfer Learning via Abstraction

I have shown the validity of the proposed framework for transfer learning via abstraction in multiple works and applications: from drone racing to high-speed navigation in unstructured environments. However, the framework also comes with some limitations. First, the required abstraction function is by design domain agnostic but task-specific. Therefore, if we change the task, *e.g.* from drone racing to acrobatics, the abstraction will have to be adjusted accordingly. One possibility to do so in an automated fashion is using domain randomization, as done in my work on racing [176]. However, domain randomization requires strong simulation engineering and expensive trial and error in the real world to define the randomization bounds. Another possibility is to pre-define the abstraction function using domain knowledge, *e.g.* feature tracks [138] or depth maps [175]. This choice favors sample efficiency, simplifies training, and promotes generalization. However, the pre-defined abstraction could potentially be suboptimal for the downstream task.

The second limitation of the proposed method is that despite abstractions, it is impossible to account for effects or motions never observed at training time. For example, if only training on planar data [177], the policy will likely not generalize to complex 3D motion. Indeed, when deploying the policies in the real world, we expect the policy to be robust to the effects observed at training time. If such effects cannot be eliminated via abstraction, *e.g.* in the case of aerodynamic drag, sudden drops in the power supply, or drifts in inertial measurements, the general practice consists of randomizing them at training time. Yet, the extent to which simulators should represent reality is unclear, and empirical investigations in this direction are exciting research avenues for future work. However, there will always be complex or too computationally intensive effects to simulate, *e.g.* contact forces or interactions with other (artificial or biological) agents. To address this limitation, I believe that the policy will require online adaptation to the environment and task. Doing so in the real world could be challenging due to the lack of privileged information or explicit reward signals but could be supported, for example, by self-supervised learning [102].

2.2 Uncertainty Estimation for Safe Deep Learning

As highlighted in the previous section, learning-based methods offer major opportunities for robust and low-latency control of drones in previously unknown environments. However, it is well known that neural network predictions are unreliable when the input sample is out of the training distribution or corrupted by noise [141]. Being able to detect such failures automatically is fundamental to integrate deep learning algorithms into robotics. Indeed, estimating uncertainties enables the interpretability of neural networks' predictions and provides valuable information during decision making [275]. This problem is even more important in the context of simulation to reality transfer, where it is not known a priori whether the training data collected in simulation is sufficient for generalization in the physical world.

Prediction uncertainty in deep neural networks generally derives from two sources: *data* uncertainty and *model* uncertainty. The former arises because of data noise, which is generally due to sensors' imperfections. The latter instead is generated from unbalances in the training data distribution. For example, a rare sample should have higher model uncertainty than a sample that appears more often in the training data. Both components of uncertainty play an important role in robotic applications. A sensor can indeed never be assumed to be noise-free, and training datasets cannot be expected to cover all the possible edge-cases.

Current approaches for uncertainty estimation require changes to the network and optimization process, typically ignore prior knowledge about the data, and tend to make over-simplifying assumptions that underestimate uncertainty. These problems have hindered their application to robotics, slowing down the integration of neural networks to large-scale, robotic systems. To address these challenges, in Paper D I have proposed a novel general framework for uncertainty estimation in deep learning.

Figure 2.7 – A neural network trained for steering angle prediction can be fully functional on a clean image (left) but generate unreliable predictions when processing a corrupted input (right). In this chapter, I propose a general framework to associate each network prediction with an uncertainty (illustrated above in red) that allows the detection of such failure cases automatically.

2.2.1 Paper D: A General Framework for Uncertainty Estimation in Deep Learning

(P5) Antonio Loquercio, Mattia Segù, and Davide Scaramuzza. "A General Framework for Uncertainty Estimation in Deep Learning". In: *IEEE Robotics Autom. Lett.* 5.2 (2020), pp. 3153–3160. DOI: 10.1109/LRA.2020.2974682

In Paper D, I present a general framework for uncertainty estimation which combines Bayesian belief networks [75, 87, 24] with Monte-Carlo sampling. Specifically, I proposed two key innovations with respect to previous works on uncertainty estimation: the use of prior information about the data, e.g., sensor noise, to compute data uncertainty, and the modeling of the relationship between data and model uncertainty. These innovations enable my framework to capture prediction uncertainties better than state-of-the-art methodologies.

Due to the large number of (possibly non-linear) operations required to generate predictions, the posterior distribution $p(\mathbf{y}|\mathbf{x})$, where $\mathbf{y}$ are output predictions and $\mathbf{x}$ is the input of a neural network, is intractable. Formally, I define the total prediction uncertainty as $\sigma_{tot} = \mathrm{Var}_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y})$. This uncertainty comes from two sources: data and model uncertainty. To estimate σ_{tot}, I derive a tractable approximation of $p(\mathbf{y}|\mathbf{x})$, which is general, i.e. agnostic to the architecture and learning process, and computationally feasible. My framework, summarized in Fig. 2.8, can be easily applied to all networks and already trained ones, given its invariance to the architecture or the training procedure. While not being specifically tailored to robotics, my approach's fundamental properties make it an appealing solution to learning-based perception and control algorithms, enabling them to be better integrated into robotic systems [275].

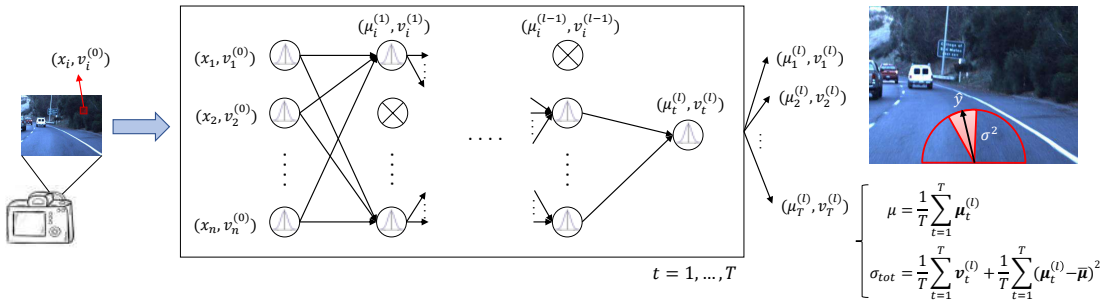


Figure 2.8 – Given an input sample $\mathbf{x}$, associated with noise $\mathbf{v}^{(0)}$, and a trained neural network, my proposed framework computes the confidence associated with the network output. To do so, it first transforms the given network into a Bayesian belief network. Then, it uses an ensemble of T such networks, created by enabling dropout at test time, to generate the final prediction $\boldsymbol{\mu}$ and uncertainty σ_{tot}.

To show the generality of our framework, we perform experiments on four challenging tasks: end-to-end steering angle prediction, obstacle future motion prediction, object recognition, and closed-loop control of a quadrotor. In these tasks, we outperform existing methodologies for uncertainty estimation by up to 23% in terms of prediction accuracy.

Related Videos

(V6) https://youtu.be/X7n-bRS5vSM

(V7) https://youtu.be/HZOxLgsk8E0

Related Software

(S4) https://github.com/uzh-rpg/deep_uncertainty_estimation

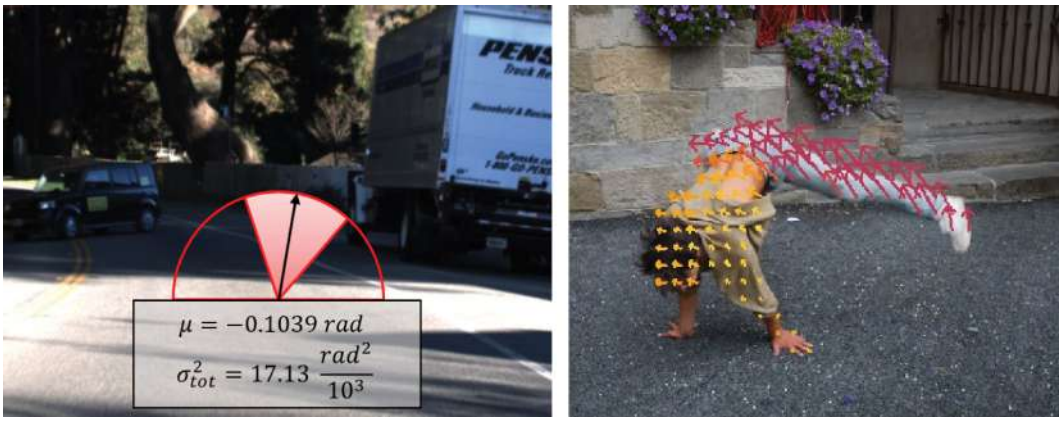


Figure 2.9 – Application of the proposed framework for uncertainty estimation on the task of steering angle prediction (left) and objects' future motion (right). For poorly illuminated frames or for motions which can't be easily predicted (*e.g.* a dancer's legs), the network is highly uncertain about its predictions.

2.2.2 Limitations of Proposed Framework for Uncertainty Estimation

The proposed approach for uncertainty estimation outperforms existing methods on several metrics and tasks. However, it also comes with some limitations. First, to extract uncertainty, a neural network needs to be converted into a bayesian one with assumed density filtering. Such conversion doubles the memory requirements (a mean and a standard deviation have to be stored for each node) and the computations required for a forward pass. In addition, the conversion can be cumbersome for complex models, particularly for recurrent ones. For example, specific activation functions or temporal aggregations might require approximations of the backward and forward passes since no closed-form solutions are known.

The second limitation of the proposed framework is that uncertainty computation requires sampling. This limitation can hinder applying the framework to the complex neural network in real-time estimation or control problems. However, I have shown that for low-dimensional problems, *e.g.* full state control of a quadrotor, a relatively small amount of samples (20) are sufficient for accurate prediction and enable real-time estimation. Recent work has shown the possibility to compute model uncertainty in a sample-free fashion [226]. However, they yet require substantial simplifications of the optimization procedure, *e.g.* linearization of the parameters' jacobian, which could lead to underestimation of the total uncertainty. Therefore,

I argue that combining such methods with our approach could lead to accurate and efficient uncertainty estimation methods.

2.3 Unsupervised and Weakly-Supervised Learning of Robot Vision

Robust and effective perception is one of the keys to the integration of robots in the real world. However, state-of-the-art perception systems generally build on a supervised learning paradigm, where an expert, *e.g.* a human or an automated algorithm with privileged information, provides labels for each observation. Motivated by the astonishing capabilities of natural intelligent agents and inspired by theories from psychology, this chapter explores the idea that perception could naturally emerge from interaction with the environment. Specifically, I examine this problem on a low-level task (*i.e.* depth estimation) and a high-level task (*i.e.* moving object detection).

In Paper E, I introduce a novel approach to learn 3D perception from the data observed by a robot exploring an environment: images and extremely sparse depth measurements, down to even a single pixel per image. Then, in Paper B, I propose an adversarial contextual model for detecting moving objects in images from just video data. In both papers, only raw data is used, without any supervision coming from annotated training data or some form of hard-coded geometrical constraint. These works show that – to some extent – it is possible to learn perception tasks from the data naturally observed by a robot when interacting with its surroundings, reaching the performance, or even outperforming, the performance of traditional supervised learning.

2.3.1 Paper E: Learning Depth with Very Sparse Supervision

(P6) Antonio Loquercio, A. Dosovitskiy, and D. Scaramuzza. "Learning Depth With Very Sparse Supervision". In: *IEEE Robotics and Automation Letters* 5 (2020), pp. 5542–5549

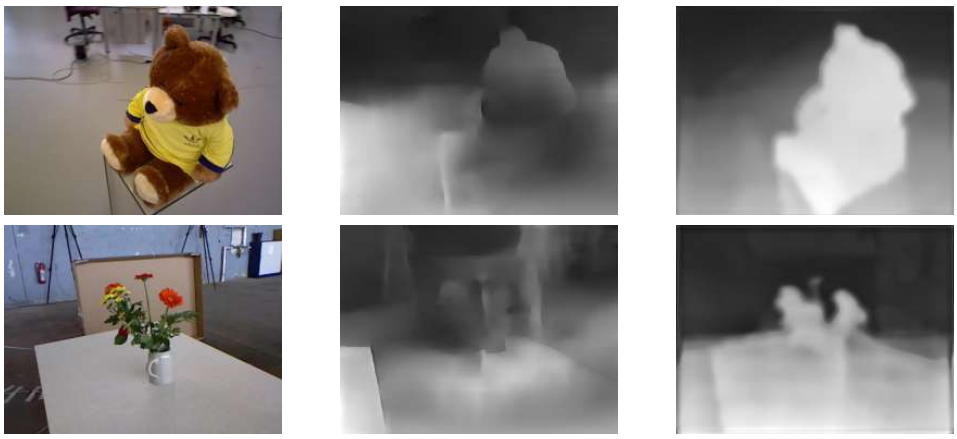
This work proposes to learn a dense depth estimation system from the data available to robots interacting with the environment: a sequence of frames and sparse depth measurements (up to one pixel per image). To learn a depth estimator from such assumptions, we design a specialized global-local deep architecture consisting of two modules, global and local. The global module takes as input two images and optical flow between them and outputs a compact latent vector of "global parameters". We expect those parameters to encode information about the observer's motion, the camera's intrinsics, and scene's features, *e.g.*, planarity. The local module then generates a compact fully convolutional network, conditioned on the "global parameters", and applies it to the optical flow field to generate the final depth estimate. The global and the local modules are trained jointly end-to-end with the available sparse depth labels and without the camera's pose or intrinsics ground truth. The proposed approach outperforms other architectural baselines and unsupervised depth estimation methods in several controlled experiments.

Related Software

(S5) https://tinyurl.com/5kzn4r72

Related Videos

(V8) https://youtu.be/a-d40Md6tKo



RGB-Image Struct2Depth [33] Ours

Figure 2.10 – I propose a novel architecture and training procedure to predict monocular depth with what would be available to a robot interacting with the environment: images and very sparse depth measurements.

2.3.2 Paper B: Unsupervised Moving Object Detection via Contextual Information Separation

(P7) Yanchao Yang*, Antonio Loquercio*, Davide Scaramuzza, and Stefano Soatto. "Unsupervised Moving Object Detection via Contextual Information Separation". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2019, pp. 879–888. DOI: 10.1109/CVPR.2019.00097

In this work, I propose a framework to automatically detect moving objects in a video. Being able to rapidly detect independently moving objects in a wide variety of scenes from images is functional to the survival of animals and autonomous vehicles alike. To detect moving objects, I identify parts of the image whose motion cannot be explained by that of their surroundings. In other words, the motion of the background is uninformative of the motion of the foreground and vice-versa. Although this method requires *no supervision* whatsoever, it outperforms several methods that are pre-trained on large annotated datasets. The proposed approach can be thought of as a generalization of classical variational generative region-based segmentation, but in a way that avoids explicit regularization or solution of partial differential equations at run-time.

Related Videos

(V9) https://youtu.be/01vClielQBw

Related Software

(S6) http://rpg.ifi.uzh.ch/unsupervised_detection.html

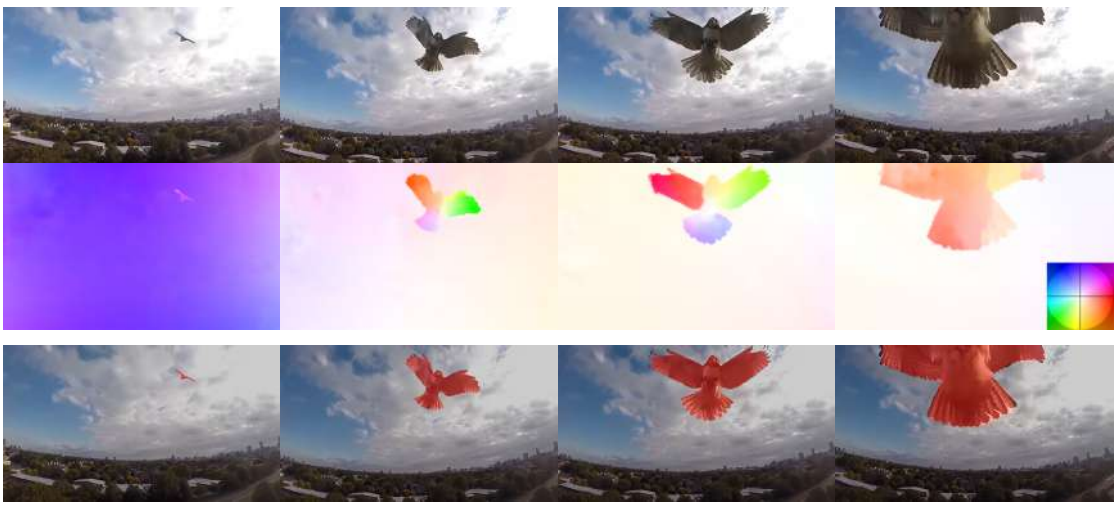


Figure 2.11 – An encounter between a hawk and a drone (top). The latter will not survive without being aware of the attack. Detecting moving objects is crucial to the survival of animal and artificial systems alike. Note that the optical flow (middle row) is quite diverse within the region where the hawk projects: It changes both in space and time. Grouping this into a moving object (bottom row) is the goal of this work.

2.3.3 Limitations of Unsupervised or Weakly-Supervised Learning

Learning via supervision, either explicit in terms of labels or implicit in reward functions, has governed both the robotics and computer vision research communities. Recently such paradigm has been challenged by the concept of self-supervised and unsupervised learning, which promises to scale up the training process (no supervision needed) and improve generalization. However, especially in robotics, yet a significant gap exists between unsupervised and supervised methods.

The methods presented in this thesis make a step forward towards self-supervised learning for robotic perception. These methods are general and flexible, but they come with two main limitations. The first one is the gap of performance with supervised methods. At the time of publication, the proposed methods achieve inferior performance than supervised ones if enough supervised labeled data exists. It is reasonable to expect that more advanced learning and optimization algorithms will likely invert this trend soon. A second limitation of the proposed methods is that they are mainly effective in perception tasks, but they have not yet proven successful in training sensorimotor controllers. The biggest challenge is defining the learning signal: why should an action be considered better than another one if no supervision is present? One possible solution to this problem is back-propagation-through-time (BBTT). Such an approach propagates forward in time a (potentially learned) model of the robot in function of actions and compares the resulting predictions to the desired states. Being the model differentiable, one can optimize actions by any optimization technique, *e.g.* gradient descent. I argue that such an unsupervised learning method will achieve results that are comparable or even superior to the ones obtained by current supervised methods in the field of sensorimotor control [138, 176], while keeping the generality and flexibility of optimization-based methods [22].

2.4 Additional Contributions

I now present additional contributions of this thesis, which consist of a public drone racing demonstrator, as well as a set of publications unrelated to the main topic of this thesis.

2.4.1 Drone Racing Demonstrator at Switzerland Innovation Park

I believe that live demonstrations are one of the best ways to communicate research results while pushing to develop systems that are lightweight and work in the real world. My lab participated in the inauguration of the Switzerland Innovation Park Zurich [1] in March 2018. Drone racing with only on-board sensing and computation, one of the core contributions of this thesis, was demonstrated on that occasion in front of a general public and members of the Swiss government. In this context, we built a race track of 10 gates at different heights with length of approximately 60 m (Figure 2.12b). The demonstration consisted of flying this track repeatedly while constantly moving gates during execution. Figure 2.12 shows some pictures captured during this event.

(a) (b) (c)

Figure 2.12 – Autonomous Drone racing demonstrations given during the inauguration of the Switzerland Innovation Park Zurich.

2.4.2 Unrelated Contributions

During my Ph.D., I developed a strong interest in a biologically-inspired neuromorphic vision sensor called event camera. Event cameras record asynchronous streams of per-pixel brightness changes, referred to as "events". They have appealing advantages over frame-based cameras for computer vision and robotics, including high temporal resolution, high dynamic range, and no motion blur. I believe that these properties will make them play a fundamental role in the future of mobile robotics. Instead of capturing brightness images at a fixed rate, event cameras measure brightness changes (called events) for each pixel independently, preventing the use of classic computer vision algorithms. I have contributed to several works to make state-of-the-art deep learning systems accessible to event cameras. I have also applied some of these ideas in the context of 3D data processing.

[1] https://www.switzerland-innovation.com/zurich/

Event-based Steering Angle Prediction for Self-driving Cars

(U1) Ana I. Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso Garcia, and Davide Scaramuzza. "Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 5419–5427. DOI: 10.1109/CVPR.2018.00568

This paper presents a deep neural network approach that unlocks the potential of event cameras on a challenging motion-estimation task: prediction of a vehicle's steering angle. To make the best out of this sensor–algorithm combination, we adapt state-of-the-art convolutional architectures to the output of event sensors and extensively evaluate the performance of our approach on a publicly available large scale event-camera dataset ($\approx$ 1000km). We present qualitative and quantitative explanations of why event cameras allow robust steering prediction even in cases where traditional cameras fail, *e.g.* challenging illumination conditions and fast motion.

Related Videos

(V10) https://youtu.be/_r_bsjkJTHA

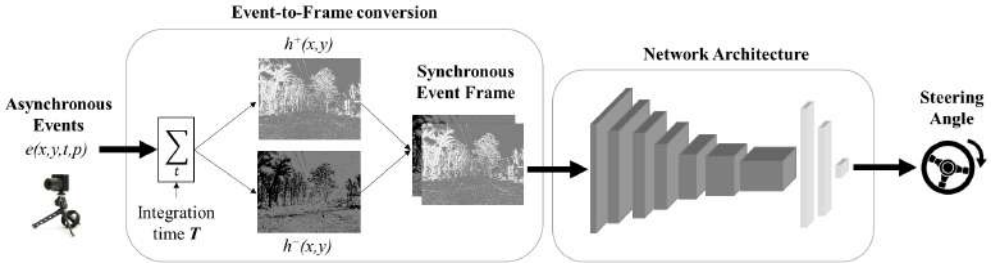


Figure 2.13 – Block diagram of the proposed approach. The output of the event camera is collected into frames over a specified time interval T, using a separate channel depending on the event polarity (positive and negative). The resulting synchronous event frames are processed by a ResNet-inspired network, which produces a prediction of the steering angle of the vehicle.

Learning of Representations for Asynchronous Event-Based Data

(U2) Daniel Gehrig, Antonio Loquercio, Konstantinos G. Derpanis, and Davide Scaramuzza. "End-to-End Learning of Representations for Asynchronous Event-Based Data". In: *International Conference on Computer Vision, ICCV*. 2019, pp. 5632–5642. DOI: 10.1109/ICCV.2019.00573

We introduce a general framework to convert asynchronous event streams into grid-based representations through a sequence of differentiable operations. Our framework comes with two main advantages: (i) allows learning the input event representation together with the task dedicated network in an end-to-end manner, and (ii) lays out a taxonomy that unifies the majority of extant event representations in the literature and identifies novel ones. Empirically, we show that our approach to learning the event representation end-to-end yields an improvement of approximately 12% on optical flow estimation and object recognition over state-of-the-art methods.

Related Publications

(R2) Nico Messikommer, Daniel Gehrig, Antonio Loquercio, and Davide Scaramuzza. "Event-Based Asynchronous Sparse Convolutional Networks". In: *European Conference Computer Vision (ECCV)*. vol. 12353. 2020, pp. 415–431. DOI: 10.1007/978-3-030-58598-3_25

Related Videos

(V11) https://youtu.be/bQtSx59GXRY

Related Software

(S7) https://github.com/uzh-rpg/rpg_event_representation_learning

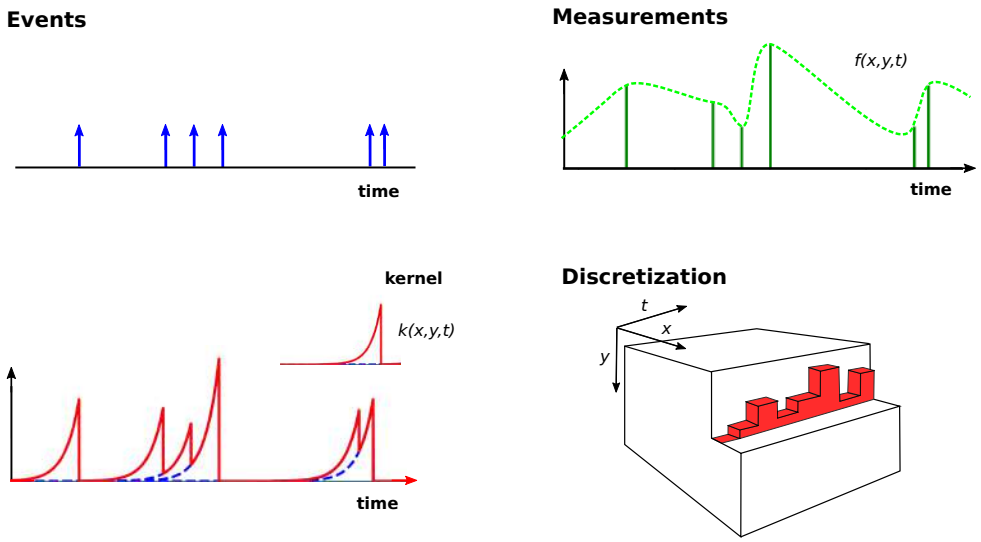


Figure 2.14 – An overview of our proposed framework. Each event is associated with a measurement (green) which is convolved with a (possibly learnt) kernel. This convolved signal is then sampled on a regular grid. Finally, various representations can be instantiated by performing projections over the temporal axis or polarities.

3D Mesh Processing

(U3) Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. "Primal-Dual Mesh Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020

We propose a novel method that allows performing inference tasks on three-dimensional geometric data by defining convolution and pooling operations on triangle meshes. We extend a primal-dual framework drawn from the graph-neural-network literature to triangle meshes and define convolutions on two types of graphs constructed from an input mesh. Our method takes features for both edges and faces of a 3D mesh as input and dynamically aggregates them using an attention mechanism. At the same time, we introduce a pooling operation with a precise geometric

interpretation, that allows handling variations in the mesh connectivity by clustering mesh faces in a task-driven fashion. We provide extensive empirical results and theoretical insights of our approach using tools from the mesh-simplification literature.

Related Videos

(V12) https://youtu.be/O8lgQgqQeuo

Related Software

(S8) https://github.com/MIT-SPARK/PD-MeshNet

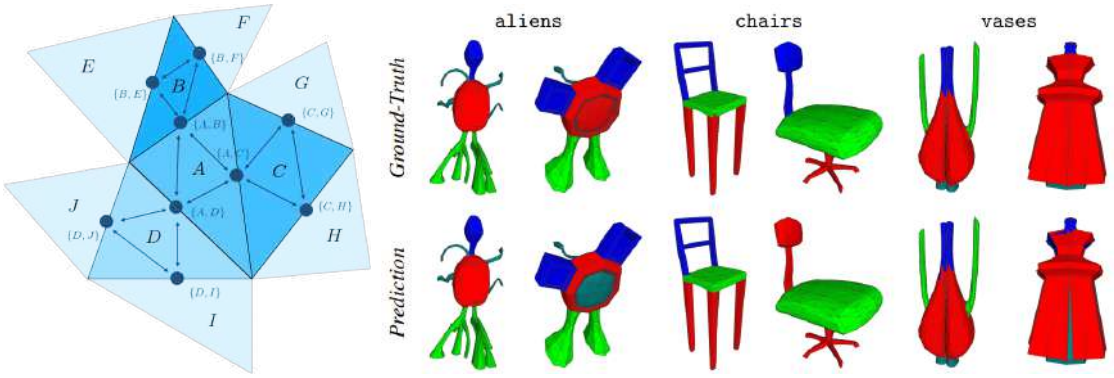


Figure 2.15 – Primal-dual graphs associated to an input mesh in PD-MeshNet (a) and qualitative results for the segmentation task on the COSEG dataset [295](b).

3 Future Directions

Deep learning is an emerging technology with the promise of addressing the synergy between robotic perception and action. Research on learning-based robotics is still in the early stages and has not yet reached the same maturity level as traditional model-based algorithms. Yet, considerable progress has been made in the last few years, which has clearly shown the potential of deep learning to overcome some of the limitations of conventional model-based autonomy.

A tight perception-action loop has numerous advantages (low latency, robustness to imperfect perception, quick adaptation to new conditions and tasks). However, it also comes with interesting challenges (sample complexity, interpretability, use of domain knowledge, generalization) which this aimed to address. Using the demonstrator of high-speed agile flight, I have shown that a tight perception-action loop enables operation in new scenarios previously inaccessible to quadrotors. I now outline a few promising research directions to address the limitations of sample complexity and generalizability of the proposed methods.

Beyond Quadrotor Flight The main demonstrator of the proposed methodologies has been agile quadrotor flight. However, the contributions of this thesis go beyond this application. Indeed, the idea of learning through input and output abstraction is a general methodology for knowledge transfer. Such an idea can be particularly effective when the input and output space is high dimensional: not only can abstractions narrow the gap between domains but also they can simplify training and favor generalization. Indeed, the idea of abstraction has been recently proved successful on both ground robots and manipulators [249, 312]. The main limitation of the approach is the lack of generalization to effects that cannot be modeled or abstracted away. For example, one challenge is to build policies for agent-to-agent or agent-to-human interaction. Simulation to reality transfer has therefore experienced only limited success in this domain. One possible solution that I envision is sample efficient adaption to the environment and task (more on this is discussed below).

Similarly, the proposed uncertainty estimation framework is not limited to the computer vision and control tasks presented in this thesis. Indeed, I argue that all safety-critical applications of machine learning can benefit from it. Due to the complexity and variability of the real world, eliminating the errors in neural network predictions due to noise in the inputs or out-of-

distribution testing is impossible. Such errors should therefore be detected and used for continual policy improvement. Yet, the main limitation lies in its computational efficiency, which can be prohibitively high for complex models with high-dimensional input and/or output spaces. Such limitation can be addressed by combining my framework with recent methods for sampling-free uncertainty estimation [226, 258].

Online Adaptive Learning. Natural agents operate in a radically different way than artificial agents. For the former, the concept of *training data* does not exist: they continuously learn to cope with their senses and experiences in the real world. In contrast, the behaviors of artificial agents are generally defined in advance and remain largely unchanged during operation. Building into artificial agents the ability to adapt could enable the efficient generalization of their knowledge to new tasks and conditions. Recent research has developed new machine learning algorithms to enable few-shot learning to new tasks and conditions, *e.g.* meta learning [69, 230, 7] or adaptive learning [252, 179, 146]. While their application in controlled environments and standard visual tasks shows very promising results, it is still unclear to what extent they can be applied to more challenging, real-world robotic tasks.

Causal Inference. The process of determining the independent effects of a particular phenomenon (cause) in a system is generally referred to as causal inference. Humans use causal cues and their related effects to make both short-term and long-term predictions, to understand the mechanism that leads to changes in the system, and to support decision making [101]. Can causal reasoning be similarly beneficial for robots? Arguably, it would enable the development of more complex representations of the world, favor adaptation and generalization to previously unseen scenarios. In addition, it could naturally provide a transparent decision-making process. For instance, a robot could understand that the effects of crashing into objects are always negative, independently of the context and application, without explicitly forcing this behavior in the training process(*e.g.* with domain randomization). Causal inference has had a large impact in several disciplines, including computer science [221], economics [290], and epidemiology [241]. Very recently causal inference also made its first steps into computer vision [248] and robotics [159], leading to increased robustness in the former and computational efficiency in the latter. However, it is not yet clear how to build causal structures from unstructured data, and how to efficiently use causal cues to support decision making in complex robotics tasks. I believe that the interactive nature of perception and action can help to solve this problem – for example, it could provide structure to the sensor data observed during operation.

Social Learning. The process of acquiring skills through the interaction with other agents is generally defined as *social learning*. Some of the most unique human cognitive skills, *e.g.* linguistic communication, abstract reasoning, and complex motor skills, are rooted in social learning [97]. Given its importance for humans, social learning has also started to play a fundamental role in machine learning and robotics. *Self-play* is arguably its most popular incarnation: in a reinforcement learning setting, agents improve by competing against each other [262, 13].

However, social learning is not all about competition: very recent research has shown its potential for achieving cooperation and coordination between artificial agents [128], developing efficient inter-communication schemes [296], and improving open-domain conversational agents [89]. Given the technical difficulties to build and control multiple robots in the real world, social learning is still in the early stages of robotics. Most of the current research has focused on the problem of human-robot interaction [96] or information exchange minimization [116]. Nonetheless, I believe that social learning could also benefit from the interactive nature of robots, powered by a tight perception-action loop, and lead to numerous advantages, *e.g.* improving the learning speed of new tasks, acquiring more robust motor skills, or exploiting unstructured social guidance typically offered by humans.

Beyond traditional visual sensors. Robots strongly rely on vision-based sensors, *e.g.* RGB cameras, to understand and interact with their surroundings. While other alternatives exist, *e.g.* distance-based sensors like lidar or radar, they have a lower resolution than cameras, are generally bulky and more expensive, and are strongly affected by environmental factors like reflective surfaces or bad weather. In contrast, cameras have very high spatial resolution and are generally agnostic to environmental features, which made them popular in robotics. However, despite their advantages, traditional cameras also have several non-idealities. For example, they are subject to motion-blur, low dynamic range, and low temporal resolution, which makes them unreliable when navigating at high-speed or in high-dynamic-range scenarios (*e.g.* when moving from indoor to outdoor). Novel bio-inspired vision sensors could be complementary to traditional cameras, offering the ability to still sense the environment in those extreme conditions. Event cameras are one of them: in contrast to frame-based cameras, they record asynchronous streams of per-pixel brightness changes, referred to as "events". They have appealing advantages over frame-based cameras for computer vision and robotics, including high temporal resolution, high dynamic range, and no motion blur. The research on event cameras is still in its early stages compared to traditional computer vision [82]. Still, already several robotic applications have shown benefits of event cameras, like flying in the dark [238], avoiding dynamic obstacles [65], and flying despite robot failure [274]. Alternative bio-inspired imaging sensors have also recently emerged, such as cellular processor arrays (SCAMP [32]), in which every pixel has a processor that allows performing several types of computations with the brightness of the pixel and its neighbors. These novel sensors, combined with specifically tailored learning algorithms (*e.g.* Spiking Neural Networks [256]) and neuromorphic processing units (*e.g.* the Intel Loihi [50]), have the potential to significantly decrease the perception latency. Lower perception latency can result in a faster perception-action loop, which in turn can enable to perform complex tasks at high speeds in challenging scenarios.

A Dronet: Learning to Fly by Driving

Reprinted, with permission, from:

Antonio Loquercio, Ana I. Maqueda, Carlos R. del-Blanco, and Davide Scaramuzza. “DroNet: Learning to Fly by Driving”. In: *IEEE Robotics Autom. Lett.* 3.2 (2018), pp. 1088–1095

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. In contrast to traditional "map-localize-plan" methods, this paper explores a data-driven approach to cope with the above challenges. To accomplish this, we propose 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 two outputs for each single input image: 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. The challenge is however 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 other 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 the urban environment, would not endanger other vehicles and pedestrians. 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. To share our findings with the robotics community, we publicly release all our datasets, code, and trained networks.

Supplementary Material

For supplementary video see: https://youtu.be/ow7aw9H4BcA. The project's code, datasets and trained

Figure A.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 off 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 in indoor corridors and parking lots.

models are available at:http://rpg.ifi.uzh.ch/dronet.html.

A.1 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 [251, 193, 62]. A robotic system facing the above tasks should simultaneously solve many challenges in perception, control, and localization. These become particularly difficult when working in urban areas, as the one illustrated in Fig. A.1. In those cases, 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 [251, 259]. Even though advanced SLAM algorithms enable localization under a wide range of conditions [180], 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 [240, 164, 254]. 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 [240, 92, 84], but they still leave the issue of collecting enough expert trajectories to imitate. Additionally, as pointed out by [84], collision trajectories 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 to dangerous situations exactly as any other ground vehicle would do. Therefore, we herein propose to use data collected from ground vehicles which are already integrated in environments as aforementioned. Overall, this work makes the following contributions:

- We propose a residual convolutional architecture which, by predicting the steering angle and the collision probability, can perform a safe flight of a quadrotor 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 our system can be applied to new application spaces without any initial knowledge about them. Indeed, with neither a map of the environment nor retraining or fine-tuning, our method generalizes to scenarios completely unseen at training time including indoor corridors, parking lots, and high altitudes.

Even though our system achieves remarkable results, we do not aim to replace traditional "map-localize-plan" approaches for drone navigation, but rather investigate whether a similar task could be done with a single shallow neural network. Indeed, we believe that learning-based and traditional approaches will one day complement each other.

A.2 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.

A 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 [251, 259]. 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 SLAM, where the robot simultaneously builds a map of the environment and self-localizes in it [180]. 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 control commands for a safe and reliable flight from it.

Recently, there has been an increasing research effort in 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) [164, 315] and (ii) methods based on supervised learning [240, 132, 92, 84, 264].

While RL-based algorithms have been successful in learning generalizing policies [164, 254], they usually require a large amount of robot experience which is costly and dangerous to acquire in real safety-critical systems. In contrast, supervised learning 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 [240], hard-coded trajectories [84], or model predictive control [132]. 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 [240]. Additionally, the domain-shift between expert and agent might hinder generalization capabilities of supervised learning methods. Indeed, previous work in [92, 264] 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 use simulations to get training data for reinforcement or imitation learning tasks, while testing the learned policy in the real world [246, 184, 315]. Clearly, this approach suffers from the domain shift between simulation and reality and might require some real-world data to be able to generalize [315]. To our knowledge, current simulators still fail to model the large amount of variability present in an urban scenario and are therefore not fully acceptable for our task. Additionally, even though some pioneering work has been done in [246], it is still not entirely clear how to make policies learned in simulation generalize into the real world.

To overcome the above-mentioned limitations, we propose to train a neural network policy by imitating expert behaviour which is generated from wheeled manned vehicles only. Even though there is a significant body of literature on the task of steering angle prediction for ground vehicles [301, 147], our goal is not to propose yet another method for steering angle prediction, but rather to prove that we can deploy this expertise also on flying platforms. The result is a single shallow network that processes all visual information concurrently, and directly produces control commands for a flying drone. The coupling between perception and control, learned end-to-end, provides several advantages, such as a simpler and lightweight system and high generalization abilities. Additionally, 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.

A.3 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 which 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 relatively small size. The resulting network, which we call DroNet, is shown in Fig. A.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. 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.3.1 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. [108], are shown in Fig. A.2 (b). Dotted lines represent skip connections defined as 1×1 convolutional shortcuts to allow the input and output of the residual blocks to be added. Even though an advantage of ResNets is to tackle the vanishing/exploding gradient problems in very deep networks, its success lies in its learning principle. Indeed, the residual scheme has been primarily introduced to address the degradation problem generated by difficulties in networks' optimization [108]. Therefore, since residual architectures are known to help generalization on both shallow and deep networks [108], we adapted this design choice to increase model performance. 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 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. Although 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' magnitudes that each loss produces. More specifically, imposing no weighting between the two losses during training results in convergence to a very poor solution. This can be explained by difference of gradients' magnitudes in the classification and regression task at the initial stages of training, which can be problematic for optimization [16]. Indeed, the gradients from the regression task are initially much larger, since

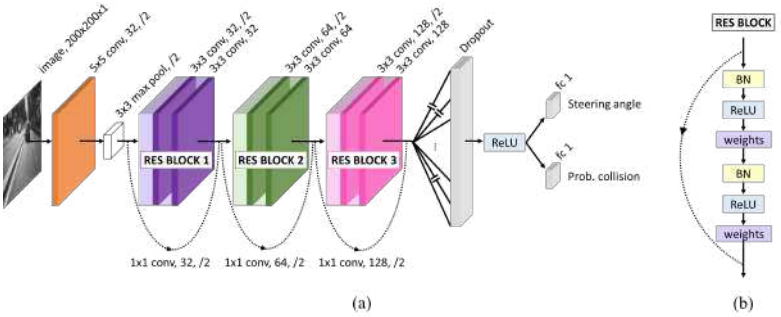


Figure A.2 – (a) DroNet is a forked Convolutional Neural Network that predicts, from a single 200×200 frame in gray-scale, a steering angle and a collision probability. The shared part of the architecture consists of a ResNet-8 with 3 residual blocks (b), followed by dropout and ReLU non-linearity. Afterwards, the network branches into 2 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 it is different from 1.

the MSE gradients' norms is proportional to the absolute steering error. Therefore, we give more and more weight to the classification loss in later stages of training. Once losses' magnitudes are comparable, the optimizer will try to find a good solution for both at the same time. For the aforementioned reasons, imposing no or constant loss weight between the two losses would likely result in sub-optimal performance or require much longer optimization times. This can be seen as a particular form of curriculum learning [16]. In detail, the weight coefficient corresponding to BCE is defined in (A.1), while the one for MSE is always 1. For our experiments, we set $decay = \frac{1}{10}$, and $epoch_0 = 10$.

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

The Adam optimizer [148] 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 for the optimization to focus on those samples which are the 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. (A.1). We define k so that it decreases over time.

A.3.2 Datasets

To learn steering angles from images, we use one of the publicly available datasets from Udacity's project [288]. 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.

Figure A.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.

For our experiment, we only use images from the forward-looking camera (Fig. A.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 (Fig. A.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. A.3(b). Collision frames are the types of data that cannot be easily obtained by a drone but are necessary to build a safe and robust system.

A.3.3 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, continuous inputs ($0 \leq \alpha \leq 1$):

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

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:

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

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. As a result, we obtain a reactive navigation policy that can reliably control a drone from a single forward-looking camera. An interesting aspect of our approach is that we can produce a collision probability from a single image without any information about the platform's speed. Indeed, we conjecture the network to make decision on the base of the distance to observed objects in the field of view. Convolutional networks are in fact well known to be successful on the task of monocular depth estimation [184]. An interesting question that we would like to answer in future work is how this approach compares to an LSTM [115] based solution, making decisions over a temporal horizon.

A.4 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 against a set of navigation baselines.

A.4.1 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.

A.4.2 Regression and Classification Results

We first evaluate the regression performance of our model employing the testing sequence from the Udacity dataset [288]. 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 prediction, we use average classification accuracy and the F-1 score[2].

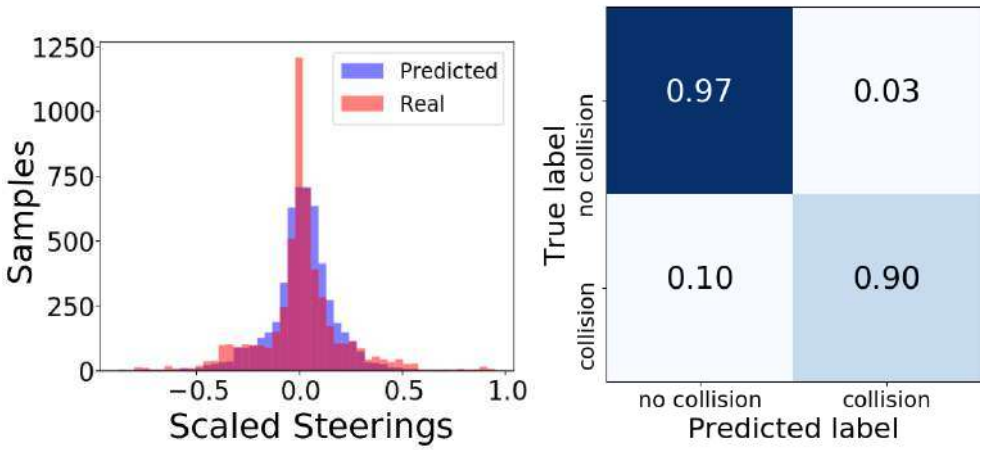
Table A.1 compares DroNet against a set of other architectures from the literature [108, 263, 92]. Additionally, we use as weak baselines a constant estimator, which always predicts 0 as steering angle and "no collision", and a random one. From these results we can observe that our design, even though 80 times smaller than the best architecture, maintains a considerable prediction performance while achieving real-time operation (20 frames per second). Furthermore, the positive comparison against the VGG-16 architecture indicates the advantages in terms of

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

[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. [92]	0.672	0.125	91.2%	0.823	6	5.8×10^4	23
ResNet-50 [108]	0.795	0.097	96.6%	0.921	50	2.6×10^7	7
VGG-16 [263]	0.712	0.119	92.7%	0.847	16	7.5×10^6	12
DroNet (Ours)	0.737	0.109	95.4%	0.901	8	3.2×10^5	20

Table A.1 – 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) (b)

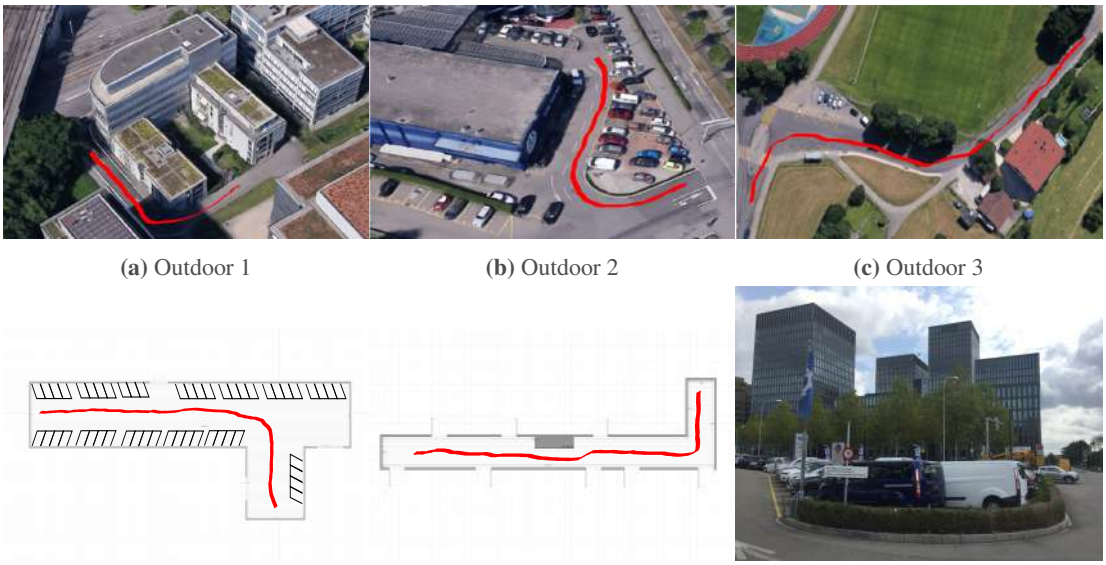
Figure A.4 – Model performance: (a) Probability Density Function (PDF) of actual vs. predicted steerings of the Udacity dataset testing sequence. (b) Confusion matrix on the collision classification evaluated on testing images of the collected dataset. Numbers in this matrix indicate the percentage of samples falling in each category.

generalization due to the residual learning scheme, as discussed in Section A.3.1. Our design succeeds at finding a good trade-off between performance and processing time as shown in Table A.1 and Fig. A.4. Indeed, in order 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.

A.4.3 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. A.5 and Fig. A.7. We compare our approach against two baselines:

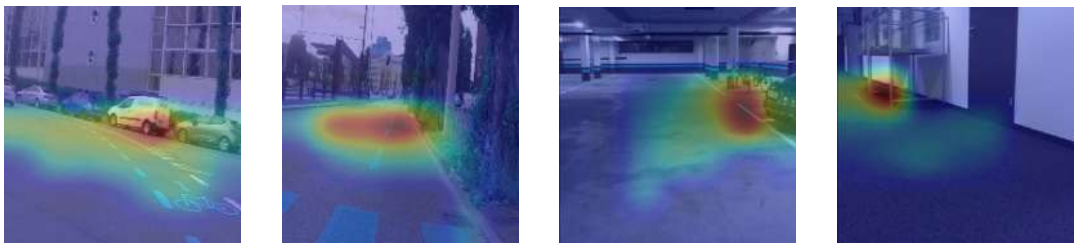
(a) *Straight line policy*: trivial baseline consisting in following a straight path in open-loop. This



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

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

Figure A.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. A.7. (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) **(b)** **(c)** **(d)**

Figure A.6 – 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 network concentrates its attention to "line-like" patterns, which approximately indicate the steering direction.

	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
Gandhi et al. [84]	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 A.2 – Average travelled distance before stopping: We show here navigation results using three different policies on a several environments. Recall that [84] 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. A.7

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 [84], which was shown to have very good control capabilities in indoor environments. We employ the same architecture as in DroNet along with our collected dataset in order to estimate the collision probability.

As metric we use the average distance travelling before stopping or colliding. Results from Table A.2 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, like pedestrians or bicycles, occasionally occluding its path. Another interesting feature of our method is that DroNet usually drives the vehicle to a random direction in open spaces and at intersections. In contrast, the baseline policy of minimizing the probability of collision was very often confused by intersections and open spaces, which 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 A.2. 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 different from a ground 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 indoor environments such as a corridor or 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 which were sometimes too narrow to be smoothly performed by our algorithm. In contrast, as we expected, the baseline policy of [84], specifically designed to work in narrow indoor spaces, outperformed our method. Still, we believe that it is very surprising that a UAV trained on outdoor streets can actually perform well even in indoor corridors.

Figure A.7 – High altitude Outdoor 1: In order 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 A.2 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.

A.4.4 Qualitative Results

In Fig. A.8 and, more extensively in the supplementary video, it is possible to observe the behaviour of DroNet in some of the considered testing environments. Unlike previous work [92], our approach always produced a safe and smooth flight. In particular, the drone always reacted promptly 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 [255]. Fig. A.6 shows which part of an image is the most important for DroNet to generate a steering decision. Intuitively, the network mainly concentrates on the "line-like" patterns present in a frame, which roughly indicate the steering direction. Indeed, the strong coupling between perception and control renders perception mainly sensitive to the features important for control. This explains why DroNet generalizes so well to many different indoor and outdoor scenes that contain "line–like" features. Conversely, we expect our approach to fail in environments missing those kind of features. This was for example the case for an experiment we performed in a forest, where no evident path was visible. However, placed in a forest surrounding with a clearly visible path, the drone behaved better.

Furthermore, the importance of our proposed methodology is supported by the difficulties encountered while carrying out outdoor city experiments. If we want a drone to learn to fly in

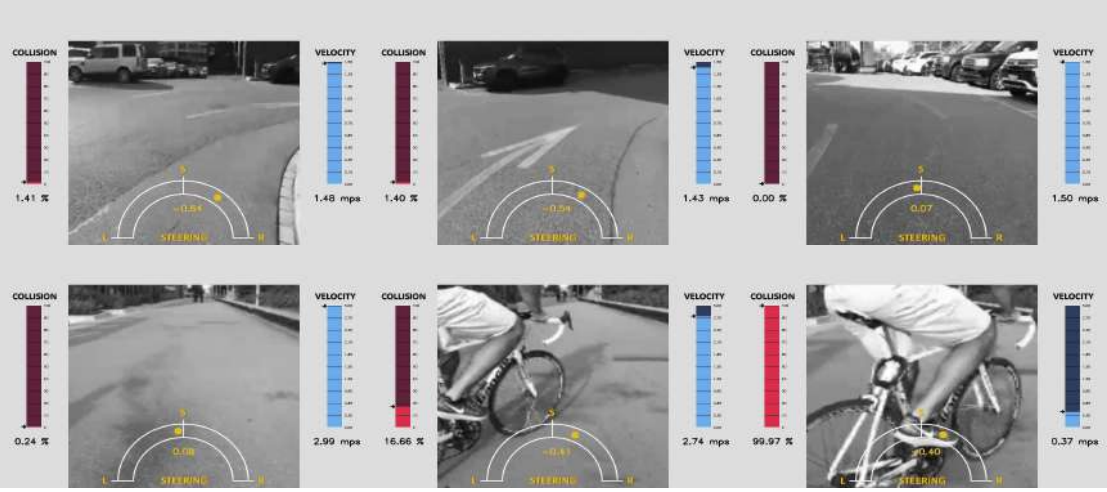


Figure A.8 – DroNet predictions: The above figures show predicted steering and probability of collision evaluated over several experiments. Despite the diverse scenarios and obstacles types, DroNet predictions always follow common sense and enable safe and reliable navigation.

a city, it is crucial to take advantage of cars, bicycles or other manned vehicles. As these are already integrated in the urban streets, they allow to collect enough valid training data safely and efficiently.

A.5 Discussion

Our methodology comes with the advantages and limitations inherent to both traditional and learning-based approaches. The advantages are that, using our simple learning and control scheme, we allow a drone to safely explore previously unseen scenes while requiring no previous knowledge about them. More specifically, in contrast to traditional approaches, there is no need to be given a map of the environment, or build it online, pre-define collision-free waypoints and localize within this map. An advantage with respect to other CNN-based controllers [264, 92, 132, 240, 315] is, that we can leverage the large body of literature present on steering angle estimation [301, 147] on both the data and the algorithmic point of view. As shown in the experiments, this gives our method high generalization capabilities. Indeed, the flying policy we provide can reliably fly in non-trivial unseen scenarios without requiring any re-training or fine-tuning, as it is generally required by CNN-based approaches [315]. Additionally, the very simple and optimized network architecture can make our approach applicable to resource constrained platforms. The limitations are primarily that the agile dynamics of drones is not fully exploited, and that it is not directly possible to explicitly give the robot a goal to be reached, as it is common in other CNN-based controllers [264, 92, 303]. There are several ways to cope with the aforementioned limitations. To exploit the drone agility, one could generate 3D collision-free trajectories, as *e.g.* in [303], when high probability of collision is predicted. To generalize to goal-driven tasks, one could either provide the network with a rough estimate of the distance to the goal [224], or, if a coarse 2D map of the environment is available, exploit recent

learning-based approaches developed for ground robots [85]. Moreover, to make our system more robust, one could produce a measure of uncertainty, as in [234]. In such a way, the system could switch to a safety mode whenever needed.

A.6 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 a UAV in such an uncontrolled environment is a laborious and dangerous task, our model learns to navigate by imitating cars and bicycles, which already follow the traffic rules. Designed to trade off performance for processing time, DroNet simultaneously predicts the collision probability and the desired steering angle, enabling a UAV to promptly react to unforeseen events and obstacles. We showed through extensive evaluations that a drone can learn to fly in cities by imitating manned vehicles. Moreover, we demonstrated interesting generalization abilities in a wide variety of scenarios. Indeed, it could be complementary to traditional "map-localize-plan" approaches in navigation-related tasks, *e.g.* search and rescue, and aerial delivery. For this reason, we release our code and datasets to share our findings with the robotics community.

B Unsupervised Moving Object Detection via Contextual Information Separation

Reprinted, with permission, from:

Yanchao Yang*, Antonio Loquercio*, Davide Scaramuzza, and Stefano Soatto. “Unsupervised Moving Object Detection via Contextual Information Separation”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2019, pp. 879–888. DOI: 10.1109/CVPR.2019.00097

Unsupervised Moving Object Detection via Contextual Information Separation

Yanchao Yang*, Antonio Loquercio*, Davide Scaramuzza, Stefano Soatto

Abstract — We propose an adversarial contextual model for detecting moving objects in images. A deep neural network is trained to predict the optical flow in a region using information from everywhere else but that region (context), while another network attempts to make such context as uninformative as possible. The result is a model where hypotheses naturally compete with no need for explicit regularization or hyper-parameter tuning. Although our method requires no supervision whatsoever, it outperforms several methods that are pre-trained on large annotated datasets. Our model can be thought of as a generalization of classical variational generative region-based segmentation, but in a way that avoids explicit regularization or solution of partial differential equations at run-time. We publicly release all our code and trained networks.[1]

B.1 Introduction

Consider Fig. B.1: Even relatively simple objects, when moving in the scene, cause complex discontinuous changes in the image. Being able to rapidly detect independently moving objects in a wide variety of scenes from images is functional to the survival of animals and autonomous vehicles alike. We wish to endow artificial systems with similar capabilities, without the need to pre-condition or learn similar-looking backgrounds. This problem relates to motion segmentation, foreground/background separation, visual attention, video object segmentation as we discuss in Sect. B.3. For now, we use the words "object" or "foreground" informally[2] to mean (possibly multiple) connected regions of the image domain, to be distinguished from their surrounding,

[1] http://rpg.ifi.uzh.ch/unsupervised_detection.html
*These two authors contributed equally.

[2] The precise meaning of these terms will be formalized in Sect. B.2.

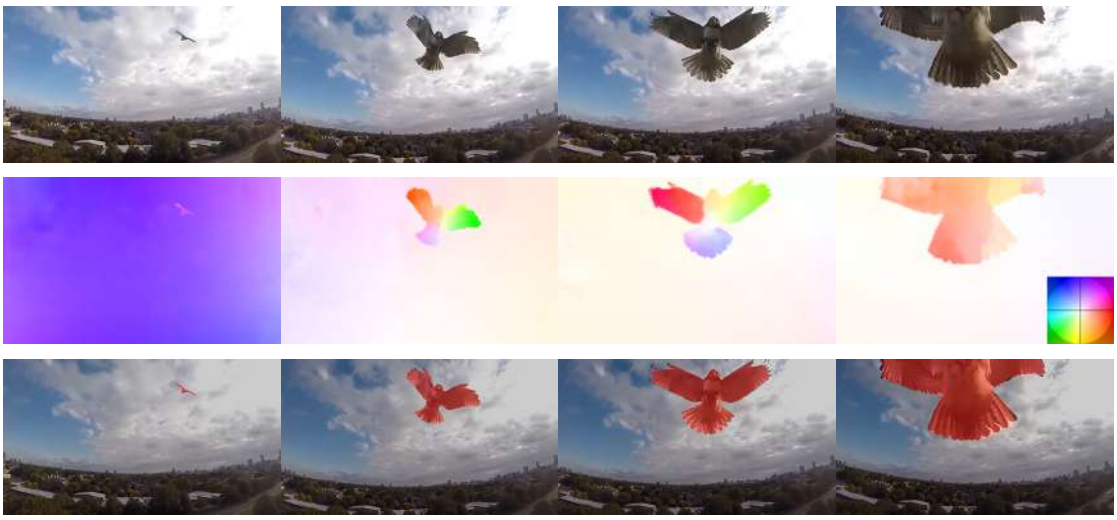


Figure B.1 – An encounter between a hawk and a drone (top). The latter will not survive without being aware of the attack. Detecting moving objects is crucial to the survival of animal and artificial systems alike. Note that the optical flow (middle row) is quite diverse within the region where the hawk projects: It changes both in space and time. Grouping this into a moving object (bottom row) is our goal in this work. Note the object is detected by our algorithm across multiple scales, partial occlusions from the viewpoint, and complex boundaries.

which we call "background" or "context," according to *some* criterion.

Since objects exist in the scene, not in the image, a method to infer them from the latter rests on an operational definition based on measurable image correlates. We call moving objects regions of the image whose motion cannot be explained by that of their surroundings. In other words, the motion of the background is uninformative of the motion of the foreground and vice-versa. The "information separation" can be quantified by the information reduction rate (IRR) between the two as defined in Sect. B.2. This naturally translates into an adversarial inference criterion that has close connections with classical variational region-based segmentation, but with a twist: Instead of learning a generative model of a region that explains the image *in that region* as well as possible, our approach yields a model that tries to explain it *as poorly as possible* using measurements from *everywhere else but* that region.

In generative model-based segmentation, one can always explain the image with a trivial model, the image itself. To avoid that, one has to impose model complexity bounds, bottlenecks or regularization. Our model does not have access to trivial solutions, as it is forced to predict a region without looking at it. What we learn instead is a contextual adversarial model, without the need for explicit regularization, where foreground and background hypotheses compete to explain the data with no pre-training nor (hyper)parameter selection. In this sense, our approach relates to adversarial learning and self-supervision as discussed in Sect. B.3.

The result is a completely unsupervised method, unlike many recent approaches that are called unsupervised but still require supervised pre-training on massive labeled datasets and can perform

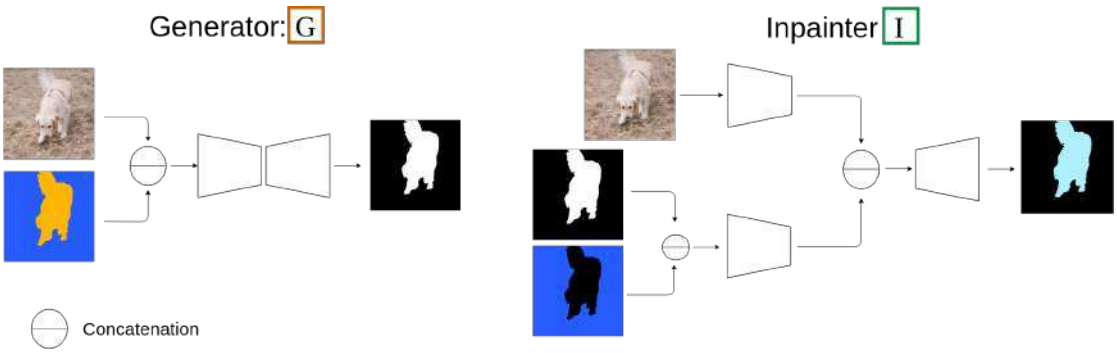


Figure B.2 – During training, our method entails two modules. One is the generator (G) which produces a mask of the object by looking at the image and the associated optical flow. The other module is the inpainter (I) which tries to inpaint back the optical flow masked out by the corresponding mask. Both modules employ the encoder-decoder structure with skip connections. However, the inpainter (I) is equipped with two separate encoding branches. See Sect. B.4.1 for network details.

poorly in contexts that are not well represented in the training set. Despite the complete lack of supervision, our method performs competitively even compared with those that use supervised pre-training (Sect. B.4).

Summary of Contributions

Our method captures the desirable features of variational region-based segmentation: Robustness, lack of thresholds or tunable parameters, no need for training. However, it does not require solving a partial differential equation (PDE) at run-time, nor to pick regularizers or Lagrange multipliers, nor to restrict the model to one that is simple-enough to be tractable analytically. It also exploits the power of modern deep learning methods: It uses deep neural networks as the model class, optimizes it efficiently with stochastic gradient descent (SGD), and can be computed efficiently at run time. However, it requires no supervision whatsoever.

While our approach has close relations to both classical region-based variational segmentation and generative models, as well as modern deep learning-based self-supervision, discussed in detail in Sect. B.3, to the best of our knowledge, it is the first *adversarial contextual model* to detect moving objects in images. It achieves better or similar performance compare to unsupervised methods on the three most common benchmarks, and it even edges out methods that rely on supervised pre-training, as described in Sect. B.4. On one of the considered benchmarks, it outperforms all methods using supervision, which illustrates the generalizability of our approach. In Sect. B.5 we describe typical failure modes and discuss limitations of our method.

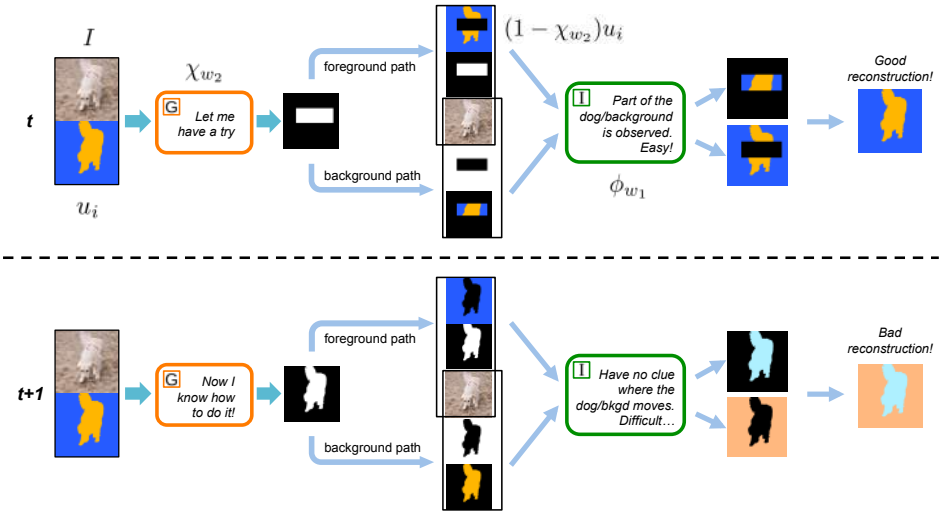


Figure B.3 – The two diagrams illustrate the learning process of the mask generator (G), after the inpainter (I) has learned how to accurately inpaint a masked flow. The upper diagram shows a poorly trained mask generator which does not precisely detect the object. Due to the imprecise detection, the inpainter can observe part of the object's flow, and perform an accurate reconstruction. At the same time, the inpainter partially observes the background's flow in the complementary mask. Consequently, it can precisely predict missing parts of the background's flow. In contrast, the lower diagram shows a fully trained mask generator which can precisely tell apart the object from the background. In this case, the inpainter observes the flow only outside the object and has no information to predict the flow inside it. At initialization time the inpainter does not know the conditionals to inpaint masked flows. Therefore, we propose to train both the generator and the inpainter jointly in an adversarial manner (see Sect. B.2).

B.2 Method

We call "moving object(s)" or "foreground" any region of an image whose motion is unexplainable from the context. A "region of an image" Ω is a compact and multiply-connected subset of the domain of the image, discretized into a lattice D. "Context" or "background" is the complement of the foreground in the image domain, $\Omega^c = D \setminus \Omega$. Given a measured image I and/or its optical flow to the next (or previous) image u, foreground and background are uncertain, and therefore treated as random variables. A random variable u_1 is "unexplainable" from (or "uninformed" by) another u_2 if their mutual information $\mathbb{I}(u_1; u_2)$ is zero, that is if their joint distribution equals the product of the marginals, $P(u_1, u_2) = P(u_1)P(u_2)$.

More specifically, the optical flow $u : D_1 \to \mathbb{R}^2$ maps the domain of an image $I_1 : D_1 \to \mathbb{R}^3_+$ onto the domain D_2 of I_2, so that if $x_i \in D_1$, then $x_i + u_i \in D_2$, where $u_i = u(x_i)$ up to a discretization into the lattice and cropping of the boundary. Ideally, if the brightness constancy constraint equation that defines optical flow was satisfied, we would have $I_1 = I_2 \circ u$ point-wise.

If we consider the flow at two locations i, j, we can formalize the notion of foreground as a region Ω that is uninformed by the background:

$$\begin{cases} \mathbb{I}(u_i, u_j | I) > 0, i, j \in \Omega \\ \mathbb{I}(u_i, u_j | I) = 0, i \in \Omega, j \in D \setminus \Omega. \end{cases} \tag{B.1}$$

As one would expect, based on this definition, if the domain of an object is included in another, then they inform each other (c.f. Supplement).

B.2.1 Loss function

We now operationalize the definition of foreground into a criterion to infer it. We use the information reduction rate (IRR) γ, which takes two subsets $\boldsymbol{x}, \boldsymbol{y} \subset D$ as input and returns a non-negative scalar:

$$\gamma(\boldsymbol{x}|\boldsymbol{y}; I) = \frac{\mathbb{I}(u_x, u_y|I)}{\mathbb{H}(u_x|I)} = 1 - \frac{\mathbb{H}(u_x|u_y, I)}{\mathbb{H}(u_x|I)} \tag{B.2}$$

where $\mathbb{H}$ denotes (Shannon) entropy. It is zero when the two variables are independent, but the normalization prevents the trivial solution (empty set).[3] As proven in the supplementary, objects as we defined them are the regions that minimize the following loss function

$$\mathcal{L}(\Omega; I) = \gamma(\Omega|\Omega^c; I) + \gamma(\Omega^c|\Omega; I). \tag{B.3}$$

Note that $\mathcal{L}$ *does not have a complexity term*, or regularizer, as one would expect in most region-based segmentation methods. This is a key strength of our approach, that involves no modeling hyperparameters, as we elaborate on in Sect. B.3.

Tame as it may look, (B.3) is intractable in general. For simplicity we indicate the flow inside the region(s) Ω (foreground) with $u^{\text{in}} = \{u_i,\ i \in \Omega\}$, and similarly for u^{out}, the flow in the background Ω^c. The only term that matters in the IRR is the ratio $\mathbb{H}(u^{\text{in}}|u^{\text{out}}, I)/\mathbb{H}(u^{\text{in}}|I)$, which is

$$\frac{\int \log P(u^{\text{in}}|u^{\text{out}}, I) dP(u^{\text{in}}|u^{\text{out}}, I)}{\int \log P(u^{\text{in}}|I) dP(u^{\text{in}}|I)} \tag{B.4}$$

that measures the information transfer from the background to the foreground. This is minimized when knowledge of the background flow is sufficient to predict the foreground. To enable computation, we have to make draconian, yet common, assumptions on the underlying probability model, namely that

$$\begin{aligned} P(u^{\text{in}} = x|I) &\propto \exp\left(-\frac{\|x\|^2}{\sigma^2}\right) \\ P(u^{\text{in}} = x|u^{\text{out}} = y, I) &\propto \exp\left(-\frac{\|x - \phi(\Omega, y, I)\|^2}{\sigma^2}\right) \end{aligned} \tag{B.5}$$

where $\phi(\Omega, y, I) = \int u^{\text{in}} dP(u^{\text{in}}|u^{\text{out}}, I)$ is the conditional mean given the image and the complementary observation. Here we assume $\phi(\Omega, \emptyset, I) = 0$, since given a single image the

[3] A small constant $0 < \epsilon \ll 1$ is added to the denominator to avoid singularities, and whenever $\boldsymbol{x} \neq \emptyset$, $\mathbb{H}(u_x|I) \gg \epsilon$, thus we will omit ϵ from now on.

most probable guess of the flow is zeros. With these assumptions, (B.4) can be simplified, to

$$\frac{\int \|u^{\text{in}} - \phi(\Omega, u^{\text{out}}, I)\|^2 dP(u^{\text{in}}|u^{\text{out}}, I)}{\int \|u^{\text{in}}\|^2 dP(u^{\text{in}}|I)} \approx \frac{\sum_{i=1}^{N} \|u_i{}^{\text{in}} - \phi(\Omega, u_i{}^{\text{out}}, I)\|^2}{\sum_{i=1}^{N} \|u_i{}^{\text{in}}\|^2} \tag{B.6}$$

where $N = |\mathcal{D}|$ is the cardinality of $\mathcal{D}$, or the number of flow samples available. Finally, our loss (B.3) to be minimized can be approximated as

$$\mathcal{L}(\Omega; I) = 1 - \frac{\sum_{i=1}^{N} \|u_i{}^{\text{in}} - \phi(\Omega, u_i{}^{\text{out}}, I)\|^2}{\sum_{i=1}^{N} \|u_i{}^{\text{in}}\|^2 + \epsilon} + 1 - \frac{\sum_{i=1}^{N} \|u_i{}^{\text{out}} - \phi(\Omega^c, u_i{}^{\text{in}}, I)\|^2}{\sum_{i=1}^{N} \|u_i{}^{\text{out}}\|^2 + \epsilon}. \tag{B.7}$$

In order to minimize this loss, we have to choose a representation for the unknown region Ω and for the function ϕ.

B.2.2 Function class

The region Ω that minimizes (B.7) belongs to the power set of D, that is the set of all possible subsets of the image domain, which has exponential complexity.[4] We represent it with the indicator function

$$\chi : D \to \{0, 1\}, \qquad i \mapsto 1 \text{ if } i \in \Omega;\ 0 \text{ otherwise} \tag{B.8}$$

so that the flow inside the region Ω can be written as $u_i{}^{\text{in}} = \chi u_i$, and outside as $u_i{}^{\text{out}} = (1 - \chi)u_i$.

Similarly, the function ϕ is non-linear, non-local, and high-dimensional, as it has to predict the flow in a region of the image of varying size and shape, given the flow in a different region. In other words, ϕ has to capture the context of a region to *recover* its flow.

Characteristically for the ages, we choose both ϕ and χ to be in the parametric function class of deep convolutional neural networks, as shown in Fig. B.2, the specifics of which are in Sect. B.4.1. We indicate the parameters with w, and the corresponding functions ϕ_{w_1} and χ_{w_2}. Accordingly, after discarding the constants, the *negative* loss (B.7) can be written as a function of the parameters

$$\mathcal{L}(w_1, w_2; I) = \frac{\sum_i \|\chi_{w_2}(u_i - \phi_{w_1}(\chi_{w_2}, u_i{}^{\text{out}}, I))\|^2}{\sum_i \|u_i{}^{\text{in}}\|^2} \tag{B.9}$$

$$+ \frac{\sum_i \|(1 - \chi_{w_2})(u_i - \phi_{w_1}(1 - \chi_{w_2}, u_i{}^{\text{in}}, I)\|^2}{\sum_i \|u_i{}^{\text{out}}\|^2} \tag{B.10}$$

[4] In the continuum, it belongs to the infinite-dimensional set of compact and multiply-connected regions of the unit square.

ϕ_{w_1} is called the *inpainter network*, and must be chosen to *minimize* the loss above. At the same time, the region Ω, represented by the parameters w_2 of its indicator function χ_{w_2} called *mask generator network*, should be chosen so that u^{out} is as uninformative as possible of u^{in}, and therefore the same loss is *maximized* with respect to w_2. This naturally gives rise to a minimax problem:

$$\hat{w} = \arg\min_{w_1} \max_{w_2} \mathcal{L}(w_1, w_2; I). \tag{B.11}$$

This loss has interesting connections to classical region-based segmentation, but with a twist as we discuss next.

B.3 Related Work

To understand the relation of our approach to classical methods, consider the simplest model for region-based segmentation [34]

$$L(\Omega, c_i, c_o) = \int_{\Omega} |u^{\text{in}}(x) - c_i|^2 dx + \int_{\Omega^c} |u^{\text{out}}(x) - c_o|^2 dx \tag{B.12}$$

typically combined with a regularizing term, for instance the length of the boundary of Ω. This is a convex infinite-dimensional optimization problem that can be solved by numerically integrating a partial differential equation (PDE). The result enjoys significant robustness to noise, provided the underlying scene has piecewise constant radiance and is measured by image irradiance, to which it is related by a simple "signal-plus-noise" model. Not many scenes of interest have piecewise constant radiance, although this method has enjoyed a long career in medical image analysis. If we enrich the model by replacing the constants c_i with smooth functions, $\phi_i(x)$, we obtain the celebrated Mumford-Shah functional [205], also optimized by integrating a PDE. Since smooth functions are an infinite-dimensional space, regularization is needed, which opens the Pandora box of regularization criteria, not to mention hyperparameters: Too much regularization and details are missed; too little and the model gets stuck in noise-induced minima. A modern version of this program would replace $\phi(x)$ with a parametrized model $\phi_w(x)$, for instance a deep neural network with weights w pre-trained on a dataset $\mathcal{D}$. In this case, the loss is a function of w, with natural model complexity bounds. Evaluating ϕ_w at a point inside, $x \in \Omega$, requires knowledge of the entire function u *inside* Ω, which we indicate with $\phi_w(x, u^{\text{in}})$:

$$\int_{\Omega} |u^{\text{in}}(x) - \phi_w(x, u^{\text{in}})|^2 dx + \int_{\Omega^c} |u^{\text{out}}(x) - \phi_w(x, u^{\text{out}})|^2 dx. \tag{B.13}$$

Here, a network can just map $\phi_w(x, u^{\text{in}}) = u^{\text{in}}$ providing a trivial solution, avoided by introducing (architectural or information) bottlenecks, akin to explicit regularizers. We turn the table around and use the outside to predict the inside and vice-versa:

$$\int_{\Omega} |u^{\text{in}}(x) - \phi_w(x, u^{\text{out}})|^2 dx + \int_{\Omega^c} |u^{\text{out}}(x) - \phi_w(x, u^{\text{in}})|^2 dx \tag{B.14}$$

After normalization and discretization, this leads to our loss function (B.7). The two regions compete: for one to grow, the other has to shrink. In this sense, our approach relates to region competition methods, and specifically Motion Competition [48], but also to adversarial training, since we can think of ϕ as the "discriminator" presented in a classification problem (GAN [9]), reflected in the loss function we use. This also relates to what is called "self-supervised learning," a misnomer since there is no supervision, just a loss function that does not involve externally annotated data. Several variants of our approach can be constructed by using different norms, or correspondingly different models for the joint and marginal distributions (B.5).

More broadly, the ability to detect independently moving objects is primal, so there is a long history of motion-based segmentation, or moving object detection. Early attempts to explicitly model occlusions include the layer model [293] with piecewise affine regions, with computational complexity improvements using graph-based methods [261] and variational inference [47, 26, 272, 306] to jointly optimize for motion estimation and segmentation; [212] use of long-term temporal consistency and color constancy, making however the optimization more difficult and sensitive to parameter choices. Similar ideas were applied to motion detection in crowds [25], traffic monitoring [17] and medical image analysis [58]. Our work also related to the literature on visual attention [125, 30].

More recent data-driven methods [282, 281, 40, 265] learn discriminative spatio-temporal features and differ mainly for the type of inputs and architectures. Inputs can be either image pairs [265, 40] or image plus dense optical flow [282, 281]. Architectures can be either time-independent [281], or with recurrent memory [282, 265]. Overall, those methods outperform traditional ones on benchmark datasets [212, 223], but at the cost of requiring a large amount of labeled training data and with evidence of poor generalization to previously unseen data.

It must be noted that, unlike in Machine Learning at large, it is customary in video object segmentation to call *"unsupervised"* methods that *do* rely on massive amounts of manually annotated data, so long as they do not require manual annotation at run-time. We adopt the broader use of the term where unsupervised means that there is no supervision of any kind both at training and test time.

Like classical variational methods, our approach does not need any annotated training data. However, like modern learning methods, our approach learns a contextual model, which would be impossible to engineer given the complexity of image formation and scene dynamics.

B.4 Experiments

We compare our approach to a set of state-of-the-art baselines on the task of video object segmentation to evaluate the accuracy of detection. We first present experiments on a controlled toy-example, where the assumptions of our model are perfectly satisfied. The aim of this experiment is to get a sense of the capabilities of the presented approach in ideal conditions. In

the second set of experiments, we evaluate the effectiveness of the proposed model on three public, widely used datasets: Densely Annotated VIdeo Segmentation (DAVIS) [223], Freiburg-Berkeley Motion Segmentation (FBMS59) [212], and SegTrackV2 [286]. Provided the high degree of appearance and resolution differences between them, these datasets represent a challenging benchmark for any moving object segmentation method. While the DAVIS dataset has always a single object per scene, FBMS and SegTrackV2 scenes can contain multiple objects per frame. We show that our method not only outperforms the unsupervised approaches, but even edges out other supervised algorithms that, in contrast to ours, have access to a large amount of labeled data with precise manual segmentation at training time. For quantitative evaluation, we employ the most common metric for video object segmentation , *i.e.* the mean Jaccard score, a.k.a. intersection-over-union score, $\mathcal{J}$. Given space constraints, we add additional evaluation metrics in the supplementary.

B.4.1 Implementation and Networks Details

Generator, *G*: Depicted on the left of Fig. B.3, the generator architecture is a shrunk version of SegNet [10]. Its encoder part consists of 5 convolutional layers each followed by batch normalization, reducing the input image to $\frac{1}{4}$ of its original dimensions. The encoder is followed by a set of 4 atrous convolutions with increasing radius (2,4,8,16). The decoder part consists of 5 convolutional layers, that, with upsampling, generate an output with the same size of the input image. As in SegNet [10], a final softmax layer generates the probabilities for each pixel to be foreground or background. The generator input consists of an RGB image I_t and the optical flow $u_{t:t+\delta T}$ between I_t and $I_{t+\delta T}$, to introduce more variations in the optical flows conditioned on image I_t. At training time, δT is randomly sampled from the uniform distribution $\mathcal{U} = [-5, 5]$, with $\delta T \neq 0$. The optical flow $u_{t:t+\delta T}$ is generated with the pretrained PWC network [273], given its state-of-the-art accuracy and efficiency. The generator network has a total of 3.4M parameters.

Inpainter, *I*: We adapt the architecture of CPN [304] to build our inpainter network. Its structure is depicted on the right of Fig. B.3. The input to this network consists of the input image I_t and the flow masked according to the generator output, χu, the latter concatenated with χ, to make the inpainter aware of the region to look for context. Differently from the CPN, these two branches are balanced, and have the same number of parameters. The encoded features are then concatenated and passed to the CPN decoder, that outputs an optical flow $\hat{u} = \phi(\chi, (1-\chi)u, I_t)$ of the same size of the input image, whose inside is going to be used for the difference between u^{in} and the recovered flow inside. Similarly, we can run the same procedure for the complement part. Our inpainter network has a total of 1.5M parameters.

At test time, only the generator G is used. Given I_t and $u_{t:t+\delta T}$, it outputs a probability for each pixel to be foreground or background, $P_t(\delta T)$. To encourage temporal consistency, we compute

	DAVIS [223]	FBMS59 [212]	SegTrackV2 [286]
$\mathcal{J}\uparrow$	92.5	88.5	92.1

Table B.1 – Performance under ideal conditions: When the assumptions made by our model are fully satisfied, our approach can successfully detect moving objects.. Indeed, our model reaches near maximum Jaccard score in all considered datasets.

	PDB [265]	FSEG [126]	LVO [282]	ARP [151]	FTS [220]	NLC [63]	SAGE [294]	CUT [144]	Ours
DAVIS2016 [223] $\mathcal{J}\uparrow$	**77.2**	70.7	75.9	**76.2**	55.8	55.1	42.6	55.2	71.5
FBMS59 [212] $\mathcal{J}\uparrow$	**74.0**	68.4	65.1	59.8	47.7	51.5	61.2	57.2	**63.6**
SegTrackV2 [286] $\mathcal{J}\uparrow$	60.9	61.4	57.3	57.2	47.8	**67.2**	57.6	54.3	62.0
DNN-Based	Yes	Yes	Yes	No	No	No	No	No	Yes
Pre-Training Required	Yes	Yes	Yes	No	No	No	No	No	No

Table B.2 – Moving Object Segmentation Benchmarks: We compare our approach with 8 different baselines on the task of moving object segmentation. In order to do so, we use three popular datasets, *i.e.* DAVIS2016 [223], FBMS59 [212], and SegTrackV2 [286]. Methods in blue require ground truth annotations at training time and are pre-trained on image segmentation datasets. In contrast, methods in red are unsupervised and not require any ground-truth annotation. Our approach is top-two in all the considered benchmarks, comparing to the other unsupervised methods. **Bold** indicates best among all methods, while **Bold Red** and red represent the best and second best for unsupervised methods, respectively.

the temporal average:

$$\overline{P_t} = \sum_{\delta T=-5,\neq 0}^{\delta T=5} P_t(\delta T) \tag{B.15}$$

The final mask χ is generated with a CRF [153] post-processing step on the final $\overline{P_t}$. More details about the post-processing can be found in the appendix.

B.4.2 Experiments in Ideal Conditions

Our method relies on basic, fundamental assumptions: *The optical flow of the foreground and of the background are independent*. To get a sense of the capabilities of our approach in ideal conditions, we artificially produce datasets where this assumption is fully satisfied. The datasets are generated as a modification of DAVIS2016 [223], FMBS [212], and SegTrackV2 [286]. While images are kept unchanged, ground truth masks are used to artificially perturb the optical flow generated by PWC [273] such that foreground and background are statistically independent. More specifically, a different (constant) optical flow field is sampled from a uniform distribution independently at each frame, and associated to the foreground and the background, respectively. More details about the generation of those datasets and the visual results can be found in the Appendix. As it is possible to observe in Table B.1, our method reaches very high performance in all considered datasets. This confirms the validity of our algorithm and that our loss function (B.11) is a valid and tractable approximation of the functional (B.3).

B.4.3 Performance on Video Object Segmentation

As previously stated, we use the term *Unsupervised* with a different meaning with respect to its definition in literature of video object segmentation. In our definition and for what follows, the supervision refers to the algorithm's usage of ground truth object annotations at training time. In contrast, the literature usually defines methods as semi-supervised, if at test time they assume the ground-truth segmentation of the first frame to be known [14, 185]. This could be posed as tracking problem [305] since the detection of the target is human generated. Instead, here we focus on moving object detection and thus we compare our approach to the methods that are usually referred to as "unsupervised" in the video object segmentation domain. However we make further differentiation on whether the ground truth object segmentation is needed (supervised) or not (truly unsupervised) during training.

In this section we compare our method with other 8 methods that represent the state of the art for moving object segmentation. For comparison, we use the same metric defined above, which is the Jaccard score $\mathcal{J}$ between the real and predicted masks.

Table B.2 shows the performance of our method and the baseline methods on three popular datasets, DAVIS2016 [223], FBMS59 [212] and SegTrackV2 [286]. Our approach is top-two in each of the considered datasets, and even outperforms baselines that need a large amount of labelled data at training time, *i.e.* FSEG [126].

As can be observed in Table B.2, unsupervised baselines typically perform well in one dataset but significantly worse in others. For example, despite being the best performing unsupervised method on DAVIS2016, the performance of ARP [151] drops significantly in the FBMS59 [212] and SegTrackV2 [212] datasets. ARP outperforms our method by 6.5% on DAVIS, however, *our method outperforms ARP by 6.3% and 8.4%, on FBMS59 and SegTrackV2 respectively*. Similarly, NLC [63] and SAGE [294] are extremely competitive in the Segtrack and FBMS59 benchmarks, respectively, but not in others. NLC outperforms us on SegTrackV2 by 8.4%, however *we outperform NLC by 29.8% and 24.7%, on DAVIS and FBMS respectively*.

It has been established that being second-best in multiple benchmarks is more indicative of robust performance than being best in one [219]. Indeed, existing unsupervised approaches for moving object segmentation are typically highly-engineered pipeline methods which are tuned on one dataset but do not necessarily generalize to others. Also, consisting of several computationally intensive steps, extant unsupervised methods are generally orders of magnitude slower than our method (Table B.3).

Interestingly, a similar pattern is observable for supervised methods. This is particularly evident on the SegTrackV2 dataset [286], which is particularly challenging since several frames have very low resolution and are motion blurred. Indeed, supervised methods have difficulties with the covariate shift due to changes in the distribution between training and testing data. Generally, supervised methods alleviate this problem by pre-training on image segmentation datasets, but this solution clearly does not scale to every possible case. In contrast, our method can be finetuned

on any data without the need for the latter to be annotated. As a result, our approach outperforms the majority of unsupervised methods as well as all the supervised ones, in terms of segmentation quality and training efficiency.

B.4.4 Qualitative experiments and Failure Cases

In Fig. B.4 we show a qualitative comparison of the detection generated by our and others' methods on the DAVIS dataset. Our algorithm can segment precisely the moving object regardless of cluttered background, occlusions, or large depth discontinuities. The typical failure case of our method is the detection of objects whose motion is due to the primary object. An example is given in the last row of Fig. B.4, where the water moved by the surfer is also classified as foreground by our algorithm.

	ARP [151]	FTS [220]	NLC [63]	SAGE [294]	CUT [144]	Ours
Runtime(s)	74.5	0.5	11.0	0.88	103.0	**0.098**
DNN-based	No	No	No	No	No	Yes

Table B.3 – Run-time analysis: Our method is not only effective (top-two in each considered dataset), but also orders of magnitude faster than other unsupervised methods. All timings are indicated without optical flow computation.

B.4.5 Training and Runtime Analysis

The generator and inpainter network's parameters are trained at the same time by minimizing the functional (B.11). The optimization time is approximately 6 hours on a single GPU Nvidia Titan XP 1080i. Since both our generator and inpainter networks are relatively small, we can afford very fast training/finetuning times. This stands in contrast to larger modules, *e.g.* PDB [265], that require up to 40 hrs of training.

At test time, predictions $\overline{P_t}$ (defined in eq. B.15) are generated at 3.15 FPS, or with an average time of 320ms per frame, including the time to compute optical flow with PWC [273]. Excluding the time to generate optical flow, our model can generate predictions at 10.2 FPS, or 98ms per frame. All previous timings do not include the CRF post-processing step. Table B.3 compares the inference time of our method with respect to other unsupervised methods. Since our method at test time requires only a pass through a relatively shallow network, it is orders of magnitude faster than other unsupervised approaches.

B.5 Discussion

Our definition of objects and the resulting inference criterion are related to generative model-based segmentation and region-based methods popular in the nineties. However, there is an important difference: Instead of using the evidence inside a region to infer a model of that region

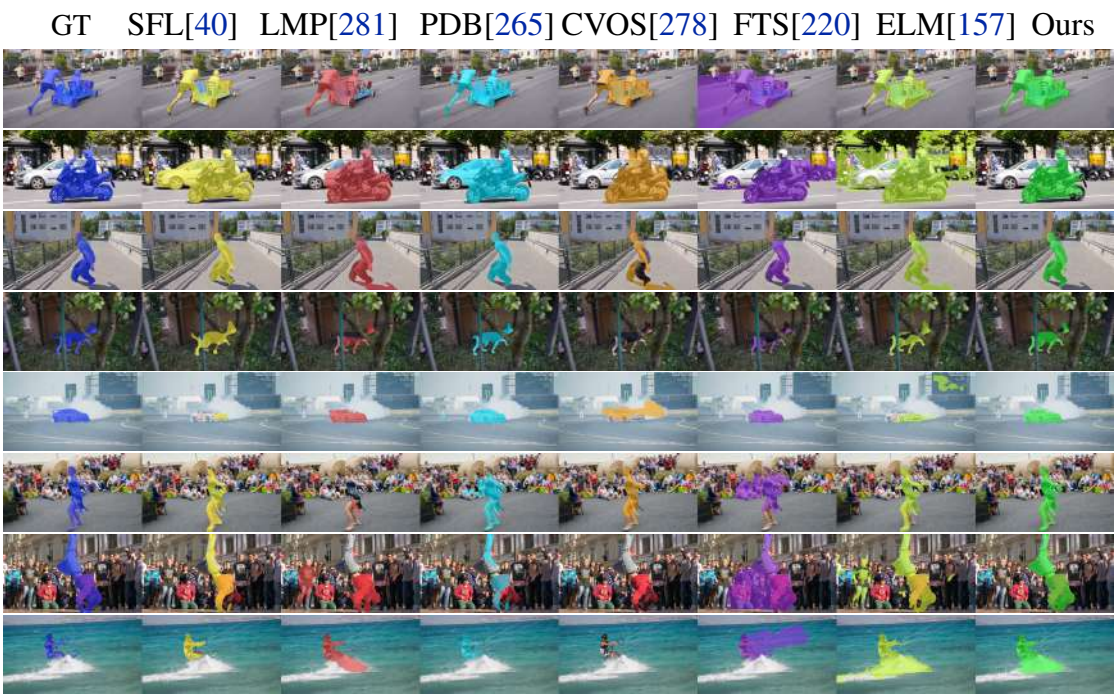


Figure B.4 – Qualitative Results: We qualitatively compare the performance of our approach with several state-of-the-art baselines as well as the Ground-Truth (GT) mask. Our prediction are robust to background clutter, large depth discontinuities and occlusions. The last row shows a typical failure case of our method, *i.e.* objects which are moved by the primary objects are detected as foreground (water is moved by the surfer in this case).

which is as accurate as possible, we use evidence *everywhere else but* that region to infer a model within the region, and we seek the model to be as bad as possible. This relation, explored in detail in Sect. B.3, forces learning a contextual model of the image, which is not otherwise the outcome of a generative model in region-based segmentation. For instance, if we choose a rich enough model class, we can trivially model the appearance of an object inside an image region as the image itself. This is not an option in our model: We can only predict the inside of a region by looking outside of it. This frees us from having to impose modeling assumptions to avoid trivial solutions, but requires a much richer class of function to harvest contextual information.

This naturally gives rise to an adversarial (min-max) optimization: An inpainter network, as a discriminator, tries to hallucinate the flow inside from the outside, with the reconstruction error as a quality measure of the generator network, which tries to force the inpainter network to do the lousiest possible job.

The strengths of our approach relate to its ability to learn complex relations between foreground and background without any annotation. This is made possible by using modern deep neural network architectures like SegNet [10] and CPN [304] as function approximators.

Not using ground-truth annotations can be seen as a strength but also a limitation: If massive

datasets are available, why not use them? In part because even massive is not large enough: We have shown that models trained on large amount of data still suffer performance drops whenever tested on a new benchmark significantly different from the training ones. Moreover, our method does not require any pre-training on large image segmentation datasets, and it can adapt to any new data, since it does not require any supervision. This adaptation ability is not only important for computer vision tasks, but can also benefit other applications, e.g. robotic navigation [177, 56] or manipulation [145].

Another limitation of our approach is that, for the task of motion-based segmentation, we require the optical flow between subsequent frames. One could argue that optical flow is costly, local, and error-prone. However, our method is general and could be applied to other statistics than optical flow. Such extensions are part of our future work agenda. In addition, our approach does not fully exploit the intensity image, although we use it as a conditioning factor for the inpainter network. An optical flow or an image can be ambiguous in some cases, but the combination of the two is rarely insufficient for recognition [306]. Again, our framework allows in theory exploitation of both, and in future work we intend to expand in this direction.

B.6 Supplementary

Notation

$\Omega \subset D$: Ω is a subset of the image domain D.
$i \in \Omega$: pixel i in region Ω.
u_Ω: optical flow u restricted to region Ω.
$\mathbb{I}(u_i, u_j|I)$: conditional mutual information between u_i and u_j given image I.

B.6.1 More on the definition of objects

Our definition Eq. (B.1) of object is in terms of mutual information between optical flows restricted to different regions. It is helpful to analyze some simple cases that follow directly from the definition.

statement 1: *a subset of an object informs the remaining part of this object.* If the object is Ω, and there is a subset $\hat{\Omega} \subset \Omega$, suppose $i \in \hat{\Omega}$, $j \in \Omega \setminus \hat{\Omega}$ respectively, then: $\mathbb{I}(u_{\hat{\Omega}}, u_{\Omega\setminus\hat{\Omega}}|I) \geq \mathbb{I}(u_i, u_{\Omega\setminus\hat{\Omega}}|I) \geq \mathbb{I}(u_i, u_j|I) > 0$ by Eq. (B.1).

statement 2: *a subset of the foreground does not inform a subset of the background.* Suppose Ω is the foreground, if $\hat{\Omega} \subset \Omega$, and $\Omega' \subset D \setminus \Omega$, then $\mathbb{I}(u_{\hat{\Omega}}, u_{\Omega'}|I) = 0$. Otherwise, we can find at least two pixels $i \in \hat{\Omega}$, $j \in \Omega'$ such that $\mathbb{I}(u_i, u_j|I) > 0$, which is contradictory to definition Eq. (B.1).

	PDB [265]	LVO [282]	FSEG [126]	LMP [281]	ARP [151]	SFL [40]	FTS [220]	NLC [63]	Ours
$\mathcal{J}$ mean ↑	77.2	75.9	70.7	70.0	76.2	67.4	55.8	55.1	71.5
$\mathcal{J}$ recall ↑	90.1	89.1	83.5	85.0	91.1	81.4	64.9	55.8	86.5
$\mathcal{J}$ decay ↓	0.9	0.0	1.5	1.3	7.0	6.2	0.0	12.6	9.5
$\mathcal{F}$ mean ↑	74.5	72.1	65.3	65.9	70.6	66.7	51.1	52.3	70.5
$\mathcal{F}$ recall ↑	84.4	83.4	73.8	79.2	83.5	77.1	51.6	51.9	83.5
$\mathcal{F}$ decay ↓	-0.2	1.3	1.8	1.5	7.9	5.1	2.9	11.4	7.0

Table B.4 – Performance on DAVIS2016 [223]: Methods in blue require ground truth annotations at training time, while the ones in red are fully unsupervised. Our approach reaches comparable performance to the state-of-the-art, outperforming several supervised methods.

B.6.2 The optimality and uniqueness of objects

In the main paper, we formalize the notion of foreground as a region Ω that is uninformed by the background, see Eq. (B.1). Objects as we defined them in Eq. (B.1) are the regions that minimize the loss function Eq. (B.3).

Proof: First we show that the estimate Ω^* right on the object achieves the minimum value of the loss function, since:

$$\mathcal{L}(\Omega^*; I) = \gamma(\Omega^* | D \setminus \Omega^*; I) + \gamma(D \setminus \Omega^* | \Omega^*; I) \tag{B.16}$$

$$= \frac{\mathbb{I}(u_{\Omega^*}, u_{D\setminus\Omega^*}|I)}{\mathbb{H}(u_{\Omega^*}|I)} + \frac{\mathbb{I}(u_{D\setminus\Omega^*}, u_{\Omega^*}|I)}{\mathbb{H}(u_{D\setminus\Omega^*}|I)} = 0 \tag{B.17}$$

by statement (2) above. Thus Ω^* achieves the minimum value of the loss Eq. (B.3). Now we need to show that Ω^* is unique, for which, we just need to check the following two mutually exclusive and collectively inclusive cases for $\hat{\Omega} \neq \Omega^*$ (note that $\mathcal{L}(\emptyset; I) = \mathcal{L}(D; I) = 1.0$ as $0 < \epsilon \ll 1$ is added to the denominator):

- $\hat{\Omega}$ is either a subset of foreground or a subset of background: $\hat{\Omega} \cap D \setminus \Omega^* = \emptyset$ or $\hat{\Omega} \cap \Omega^* = \emptyset$.
- $\hat{\Omega}$ is neither a subset of foreground nor a subset of background: $\hat{\Omega} \cap D \setminus \Omega^* \neq \emptyset$ and $\hat{\Omega} \cap \Omega^* \neq \emptyset$.

In both cases $\mathcal{L}(\hat{\Omega}; I)$ is strictly larger than 0 with some set operations under statements (1,2) above. Thus the object satisfies the definition Eq. (B.1) is a unique optima of the loss Eq. (B.3).

B.6.3 Extra quantitative evaluations

The performance of our algorithm compared with state-of-the-art systems on DAVIS2016 [223] can be found in Table B.4. The metrics used to perform the quantitative evaluation are the Jaccard score $\mathcal{J}$, and the mean boundary measure $\mathcal{F}$. For more details see [223]. According to the same metrics, we also provide the per-category score for DAVIS and FBMS59 in Table B.5 and B.6.

Category	$\mathcal{J}$ mean ↑	$\mathcal{J}$ recall ↑	$\mathcal{J}$ decay ↓	$\mathcal{F}$ mean ↑	$\mathcal{F}$ recall ↑	$\mathcal{F}$ decay ↓
blackswan	69.1	100.0	25.2	72.4	100.0	23.6
bmx-trees	59.2	74.4	12.6	84.1	100.0	-7.4
breakdance	82.4	100.0	-0.6	84.1	100.0	-3.3
camel	83.0	100.0	4.6	83.0	100.0	0.7
car-roundabout	87.6	100.0	-0.8	76.5	98.6	-5.5
car-shadow	78.6	100.0	15.4	74.3	100.0	6.2
cows	85.4	100.0	0.6	79.6	98.0	-1.3
dance-twirl	79.0	95.5	1.2	82.4	100.0	9.9
dog	80.0	100.0	10.2	76.1	94.8	17.8
drift-chicane	62.0	80.0	10.9	76.1	90.0	18.4
drift-straight	67.9	87.5	16.9	57.7	54.2	43.0
goat	26.9	17.0	-17.7	34.4	17.0	-14.7
horsejump-high	79.6	100.0	15.6	87.6	100.0	9.4
kite-surf	23.9	0.0	6.7	44.0	20.8	-5.2
libby	76.5	91.5	19.1	92.2	100.0	2.7
motocross-jump	70.6	78.9	-1.6	53.4	57.9	-7.0
paragliding-launch	72.7	100.0	25.0	44.3	43.6	33.3
parkour	83.4	98.0	5.4	88.9	100.0	7.9
scooter-black	76.5	95.1	-3.5	70.2	95.1	-2.0
soapbox	77.0	95.9	16.7	74.5	100.0	18.0
Mean	71.5	86.5	9.5	70.5	83.5	7.0

Table B.5 – Performance on DAVIS2016 [223]: Per category performance on the DAVIS2016 dataset.

Category	$\mathcal{J}$ mean ↑	$\mathcal{J}$ recall ↑	$\mathcal{J}$ decay ↓	$\mathcal{F}$ mean ↑	$\mathcal{F}$ recall ↑	$\mathcal{F}$ decay ↓
camel01	78.3	100.0	3.3	83.1	100.0	4.9
cars1	61.4	100.0	0.0	37.5	0.0	0.0
cars10	31.1	0.0	-9.4	22.9	0.0	0.8
cars4	83.6	100.0	7.6	79.5	100.0	11.6
cars5	79.0	100.0	9.2	78.5	100.0	14.1
cats01	90.0	100.0	5.1	90.0	100.0	19.2
cats03	75.8	100.0	-12.6	73.8	100.0	0.8
cats06	61.3	81.2	27.3	75.6	87.5	17.5
dogs01	73.4	77.8	35.9	74.3	88.9	21.3
dogs02	73.8	90.0	-9.7	74.4	90.0	10.9
farm01	82.4	91.7	-24.6	71.0	83.3	-29.0
giraffes01	38.5	46.7	-25.4	44.2	33.3	-3.0
goats01	47.3	61.5	55.8	61.0	61.5	27.8
horses02	63.1	81.8	18.3	67.3	90.9	36.1
horses04	59.3	82.1	1.5	60.0	87.2	-5.9
horses05	41.0	28.6	-10.3	38.7	19.0	11.6
lion01	53.5	57.1	12.7	63.3	100.0	-3.4
marple12	68.6	100.0	-15.6	58.6	83.3	-0.9
marple2	67.7	66.7	33.5	55.6	55.6	47.3
marple4	68.2	100.0	-3.9	58.9	100.0	3.0
marple6	53.7	57.9	17.6	41.8	21.1	22.0
marple7	58.1	78.6	-1.8	35.6	21.4	-17.5
marple9	66.0	100.0	2.7	41.5	16.7	-3.9
people03	62.4	87.5	23.3	56.2	100.0	7.8
people1	88.2	100.0	1.6	96.7	100.0	2.6
people2	82.1	100.0	-7.2	81.3	100.0	-0.6
rabbits02	50.2	66.7	-5.9	55.5	75.0	-4.9
rabbits03	45.4	40.0	-7.7	59.7	100.0	-3.5
rabbits04	44.0	50.0	31.2	50.9	57.1	26.6
tennis	70.9	100.0	-4.9	78.1	100.0	-2.4
Mean	63.6	78.2	4.9	62.2	72.4	7.02

Table B.6 – Performance on FBMS59 [212]: Per category performance on FBMS59.

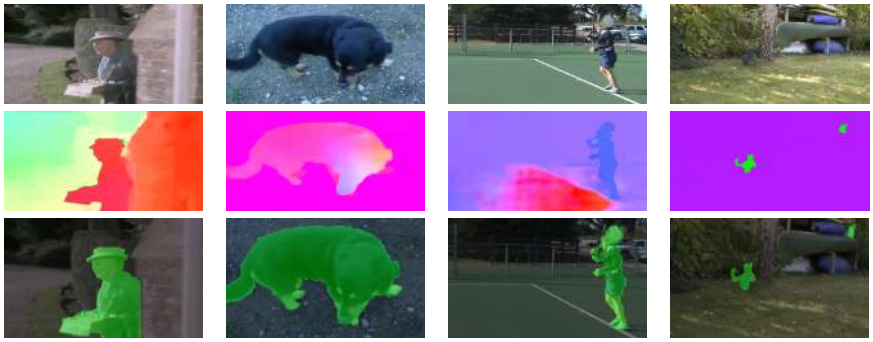


Figure B.5 – Experiment on FBMS in ideal conditions: The first row shows some samples of input images, the second row their idealized optical flows, and the latter row the segmentation generated by our algorithm.

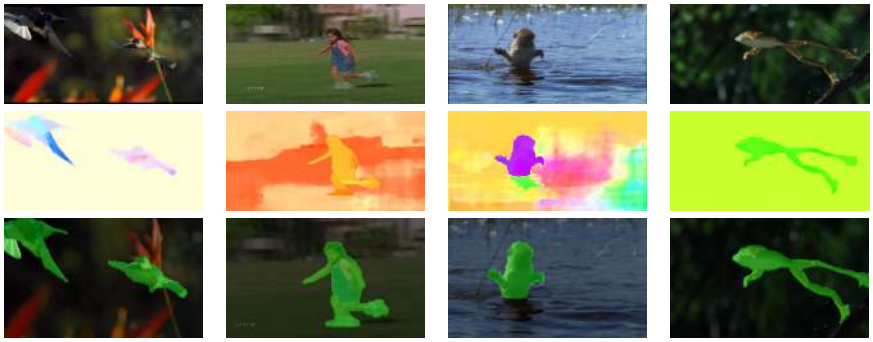


Figure B.6 – Experiment on SegTrackV2 in ideal conditions: The first row shows samples of input images, the second row the idealized optical flows, and the latter row the segmentation generated by our algorithm.

B.6.4 Details on CRF

We use the online implementation[5] of the CRF algorithm [153] for post-processing the mask $\overline{P_t}$. We only use the pairwise bilateral potential with the parameters: sxy=25, srgb=5, compat=5 as defined in the corresponding function.

B.6.5 Experiments in ideal conditions

The experiments under ideal conditions have been specifically designed to test the performance of our algorithm when its assumptions are fully satisfied. In particular, the derivation presented in Section 2 assumes that the foreground and the background motion are completely independent given the image. In real datasets, this conditions is not always true: during the generation of the benchmark datasets, many times the camera tracks the moving object, creating a relation between the two motions. To analyze the performance of our method when background and

[5] https://github.com/lucasb-eyer/pydensecrf

foreground are *completely independent*, we artificially modify our considered benchmark datasets. To generate an ideal sample, we proceed as follows: We first take two image, I_t and $I_{t+\delta T}$, with $\delta T \in \mathcal{U}[-5,5]$. Then, we generate the optical flow between the two frame $u_{t:t+\delta T}$ using PWC Net [273]. Now, using the ground-truth mask, we artificially add to the latter a random optical flow, different between the foreground and background. The random optical flows are generated from a rotation $r \in \mathcal{U}[-1,1]$ radians and translations $t_x, t_y \in \mathcal{U}[-30,30]$ pixels. Since the ground-truth mask is used to generate the flow, in several case a solution is easy to find (last column of Fig. B.5 and Fig. B.6). However, as the original optical flow can be complex, it is not always possible to easily observe the mask in the flow (first three columns of Fig. B.5 and Fig. B.6). Nonetheless, since the background and foreground are fully independent, our algorithm can accurately segment the objects.

C Deep Drone Racing: From Simulation to Reality with Domain Randomization

This work first appeared as a conference paper:

Elia Kaufmann*, Antonio Loquercio*, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. “Deep drone racing: Learning agile flight in dynamic environments”. In: *Conference on Robot Learning (CoRL)*. 2018

It was later extended and published in Transactions on Robotics (T-RO). The version presented here is reprinted, with permission, from:

Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. “Deep Drone Racing: From Simulation to Reality With Domain Randomization”. In: *IEEE Trans. Robotics* 36.1 (2019), pp. 1–14

Deep Drone Racing: From Simulation to Reality with Domain Randomization

Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun,

Davide Scaramuzza

Abstract — Dynamically changing environments, unreliable state estimation, and operation under severe resource constraints are fundamental challenges that limit the deployment of small autonomous drones. We address these challenges in the context of autonomous, vision-based drone racing in dynamic environments. A racing drone must traverse a track with possibly moving gates at high speed. We enable this functionality by combining the performance of a state-of-the-art planning and control system with the perceptual awareness of a convolutional neural network (CNN). The resulting modular system is both platform- and domain-independent: it is trained in simulation and deployed on a physical quadrotor without any fine-tuning. The abundance of simulated data, generated via domain randomization, makes our system robust to changes of illumination and gate appearance. To the best of our knowledge, our approach is the first to demonstrate *zero-shot sim-to-real* transfer on the task of agile drone flight. We extensively test the precision and robustness of our system, both in simulation and on a physical platform, and show significant improvements over the state of the art.

C.1 Introduction

Drone racing is a popular sport in which professional pilots fly small quadrotors through complex tracks at high speeds (Fig. C.1). Drone pilots undergo years of training to master the sensorimotor skills involved in racing. Such skills would also be valuable to autonomous systems in applications such as disaster response or structure inspection, where drones must be able to quickly and safely fly through complex dynamic environments [302].

Developing a fully autonomous racing drone is difficult due to challenges that span dynamics modeling, onboard perception, localization and mapping, trajectory generation, and optimal

Figure C.1 – The perception block of our system, represented by a convolutional neural network (CNN), is trained *only* with non-photorealistic simulation data. Due to the abundance of such data, generated with domain randomization, the trained CNN can be deployed on a physical quadrotor without any finetuning.

control. For this reason, autonomous drone racing has attracted significant interest from the research community, giving rise to multiple autonomous drone racing competitions [198, 199].

One approach to autonomous racing is to fly through the course by tracking a precomputed global trajectory. However, global trajectory tracking requires to know the race-track layout in advance, along with highly accurate state estimation, which current methods are still not able to provide [73, 227, 31]. Indeed, visual inertial odometry [73, 227] is subject to drift in estimation over time. SLAM methods can reduce drift by relocalizing in a previously-generated, globally-consistent map. However, enforcing global consistency leads to increased computational demands that strain the limits of on-board processing. In addition, regardless of drift, both odometry and SLAM pipelines enable navigation only in a predominantly-static world, where waypoints and collision-free trajectories can be statically defined. Generating and tracking a global trajectory would therefore fail in applications where the path to be followed cannot be defined *a priori*. This is usually the case for professional drone competitions, since gates can be moved from one lap to another.

In this paper, we take a step towards autonomous, vision-based drone racing in dynamic environments. Instead of relying on globally consistent state estimates, our approach deploys a convolutional neural network to identify waypoints in local body-frame coordinates. This eliminates the problem of drift and simultaneously enables our system to navigate through dynamic environments. The network-predicted waypoints are then fed to a state-of-the-art planner [201] and tracker [61], which generate a short trajectory segment and corresponding motor commands to reach the desired location. The resulting system combines the perceptual awareness of CNNs with the precision offered by state-of-the-art planners and controllers, getting the best of both worlds. The approach is both powerful and lightweight: all computations run fully onboard.

An earlier version of this work [139] (Best System Paper award at the Conference on Robotic Learning, 2018) demonstrated the potential of our approach both in simulation and on a physical platform. In both domains, our system could perform complex navigation tasks, such as seeking a moving gate or racing through a dynamic track, with higher performance than state-of-the-art, highly engineered systems. In the present paper, we extend the approach to generalize to environments and conditions not seen at training time. In addition, we evaluate the effect of

design parameters on closed-loop control performance, and analyze the computation-accuracy trade-offs in the system design.

In the earlier version [139], the perception system was track specific: it required a substantial amount of training data from the target race track. Therefore, significant changes in the track layout, background appearance, or lighting would hurt performance. In order to increase the generalization abilities and robustness of our perception system, we propose to use domain randomization [279]. The idea is to randomize during data collection all the factors to which the system must be invariant, i.e., illumination, viewpoint, gate appearance, and background. We show that domain randomization leads to an increase in closed-loop performance relative to our earlier work [139] when evaluated in environments or conditions not seen at training time. Specifically, we demonstrate performance increases of up to 300% in simulation (Fig. C.6) and up to 36% in real-world experiments (Fig. C.14).

Interestingly, the perception system becomes invariant not only to specific environments and conditions but also to the training domain. We show that after training purely in non-photorealistic simulation, the perception system can be deployed on a physical quadrotor that successfully races in the real world. On real tracks, the policy learned in simulation has comparable performance to one trained with real data, thus alleviating the need for tedious data collection in the physical world.

C.2 Related Work

Pushing a robotic platform to the limits of handling gives rise to fundamental challenges for both perception and control. On the perception side, motion blur, challenging lighting conditions, and aliasing can cause severe drift in vision-based state estimation [73, 206, 180]. Other sensory modalities, e.g. LIDAR or event-based cameras, could partially alleviate these problems [27, 238]. Those sensors are however either too bulky or too expensive to be used on small racing quadrotors. Moreover, state-of-the-art state estimation methods are designed for a predominantly-static world, where no dynamic changes to the environment occur.

From the control perspective, plenty of work has been done to enable high-speed navigation, both in the context of autonomous drones [190, 201, 200] and autonomous cars [154, 134, 140, 245]. However, the inherent difficulties of state estimation make these methods difficult to adapt for small, agile quadrotors that must rely solely on onboard sensing and computing. We will now discuss approaches that have been proposed to overcome the aforementioned problems.

C.2.1 Data-driven Algorithms for Autonomous Navigation

A recent line of work, focused mainly on autonomous driving, has explored data-driven approaches that tightly couple perception and control [56, 218, 234, 133]. These methods provide several interesting advantages, e.g. robustness against drifts in state estimation [56, 218] and

the possibility to learn from failures [133]. The idea of learning a navigation policy end-to-end from data has also been applied in the context of autonomous, vision-based drone flight [246, 177, 84]. To overcome the problem of acquiring a large amount of annotated data to train a policy, Loquercio et al. [177] proposed to use data from ground vehicles, while Gandhi et al. [84] devised a method for automated data collection from the platform itself. Despite their advantages, end-to-end navigation policies suffer from high sample complexity and low generalization to conditions not seen at training time. This hinders their application to contexts where the platform is required to fly at high speed in dynamic environments. To alleviate some of these problems while retaining the advantages of data-driven methods, a number of works propose to structure the navigation system into two modules: perception and control [100, 54, 36, 45, 202]. This kind of modularity has proven to be particularly important for transferring sensorimotor systems across different tasks [45, 54] and application domains [36, 202].

We employ a variant of this perception-control modularization in our work. However, in contrast to prior work, we enable high-speed, agile flight by making the output of our neural perception module compatible with fast and accurate model-based trajectory planners and trackers.

C.2.2 Drone Racing

The popularity of drone racing has recently kindled significant interest in the robotics research community. The classic solution to this problem is image-based visual servoing, where a robot is given a set of target locations in the form of reference images or patterns. Target locations are then identified and tracked with hand-crafted detectors [276, 66, 161]. However, the handcrafted detectors used by these approaches quickly become unreliable in the presence of occlusions, partial visibility, and motion blur. To overcome the shortcomings of classic image-based visual servoing, recent work proposed to use a learning-based approach for localizing the next target [131]. The main problem of this kind of approach is, however, limited agility. Image-based visual servoing is reliable when the difference between the current and reference images is small, which is not always the case under fast motion.

Another approach to autonomous drone racing is to learn end-to-end navigation policies via imitation learning [204]. Methods of this type usually predict low-level control commands, in the form of body-rates and thrust, directly from images. Therefore, they are agnostic to drift in state estimation and can potentially operate in dynamic environments, if enough training data is available. However, despite showing promising results in simulated environments, these approaches still suffer from the typical problems of end-to-end navigation: (i) limited generalization to new environments and platforms and (ii) difficulties in deployment to real platforms due to high computational requirements (desired inference rate for agile quadrotor control is much higher than what current on-board hardware allows).

To facilitate robustness in the face of unreliable state estimation and dynamic environments, while also addressing the generalization and feasibility challenges, we use modularization. On one

hand, we take advantage of the perceptual awareness of CNNs to produce navigation commands from images. On the other hand, we benefit from the high speed and reliability of classic control pipelines for generation of low-level controls.

C.2.3 Transfer from Simulation to Reality

Learning navigation policies from real data has a shortcoming: high cost of generating training data in the physical world. Data needs to be carefully collected and annotated, which can involve significant time and resources. To address this problem, a recent line of work has investigated the possibility of training a policy in simulation and then deploying it on a real system. Work on transfer of sensorimotor control policies has mainly dealt with manual grasping and manipulation [99, 299, 23, 127, 243, 247]. In driving scenarios, synthetic data was mainly used to train perception systems for high-level tasks, such as semantic segmentation and object detection [235, 129]. One exception is the work of Müller et al. [202], which uses modularization to deploy a control policy learned in simulation on a physical ground vehicle. Domain transfer has also been used for drone control: Sadeghi and Levine [246] learned a collision avoidance policy by using 3D simulation with extensive domain randomization.

Akin to many of the aforementioned methods, we use domain randomization [279] and modularization [202] to increase generalization and achieve sim-to-real transfer. Our work applies these techniques to drone racing. Specifically, we identify the most important factors for generalization and transfer with extensive analyses and ablation studies.

C.3 Method

We address the problem of robust, agile flight of a quadrotor in a dynamic environment. Our approach makes use of two subsystems: perception and control. The perception system uses a Convolutional Neural Network (CNN) to predict a goal direction in local image coordinates, together with a desired navigation speed, from a single image collected by a forward-facing camera. The control system uses the navigation goal produced by the perception system to generate a minimum-jerk trajectory [201] that is tracked by a low-level controller [61]. In the following, we describe the subsystems in more detail.

Perception system. The goal of the perception system is to analyze the image and provide a desired flight direction and navigation speed for the robot. We implement the perception system by a convolutional network. The network takes as input a 300×200 pixel RGB image, captured from the onboard camera, and outputs a tuple $\{\vec{x}, v\}$, where $\vec{x} \in [-1, 1]^2$ is a two-dimensional vector that encodes the direction to the new goal in normalized image coordinates, and $v \in [0, 1]$ is a normalized desired speed to approach it. To allow for onboard computing, we employ a modification of the DroNet architecture of Loquercio et al. [177]. In section C.4.3, we will present the details of our architecture, which was designed to optimize the trade-off between

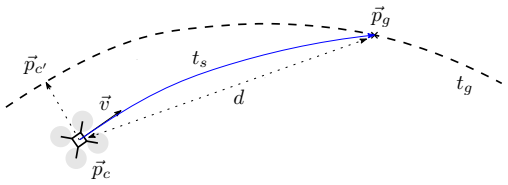


Figure C.2 – The pose $\vec{p}_c$ of the quadrotor is projected on the global trajectory t_g to find the point $\vec{p}_{c'}$. The point at distance d from the current quadrotor position $\vec{p}_c$, which belongs to t_g in the forward direction with respect to $\vec{p}_{c'}$, defines the desired goal position $\vec{p}_g$. To push the quadrotor towards the reference trajectory t_g, a short trajectory segment t_s is planned and tracked in a receding horizon fashion.

accuracy and inference time. With our hardware setup, the network achieves an inference rate of 15 frames per second while running concurrently with the full control stack. The system is trained by imitating an automatically computed expert policy, as explained in Section C.3.1.

Control system. Given the tuple $\{\vec{x}, v\}$, the control system generates low-level commands. To convert the goal position $\vec{x}$ from two-dimensional normalized image coordinates to three-dimensional local frame coordinates, we back-project the image coordinates $\vec{x}$ along the camera projection ray and derive the goal point at a depth equal to the prediction horizon d (see Figure C.2). We found setting d proportional to the normalized platform speed v predicted by the network to work well. The desired quadrotor speed v_{des} is computed by rescaling the predicted normalized speed v by a user-specified maximum speed v_{max}: $v_{des} = v_{max} \cdot v$. This way, with a single trained network, the user can control the aggressiveness of flight by varying the maximum speed. Once p_g in the quadrotor's body frame and v_{des} are available, a state interception trajectory t_s is computed to reach the goal position (see Figure C.2). Since we run all computations onboard, we use computationally efficient minimum-jerk trajectories [201] to generate t_s. To track t_s, i.e. to compute the low-level control commands, we employ the control scheme proposed by Faessler et al. [61].

C.3.1 Training Procedure

We train the perception system with imitation learning, using automatically generated globally optimal trajectories as a source of supervision. To generate these trajectories, we make the assumption that at training time the location of each gate of the race track, expressed in a common reference frame, is known. Additionally, we assume that at training time the quadrotor has access to accurate state estimates with respect to the latter reference frame. Note however that at test time no privileged information is needed and the quadrotor relies on image data only. The overall training setup is illustrated in Figure C.2.

Expert policy. We first compute a global trajectory t_g that passes through all gates of the track, using the minimum-snap trajectory implementation from Mellinger and Kumar [190]. To generate training data for the perception network, we implement an expert policy that follows the reference trajectory. Given a quadrotor position $\vec{p}_c \in \mathbb{R}^3$, we compute the closest point $\vec{p}_{c'} \in \mathbb{R}^3$ on the

global reference trajectory. The desired position $\vec{p}_g \in \mathbb{R}^3$ is defined as the point on the global reference trajectory the distance of which from $\vec{p}_c$ is equal to the prediction horizon $d \in \mathbb{R}$. We project the desired position $\vec{p}_g$ onto the image plane of the forward facing camera to generate the ground truth normalized image coordinates $\vec{x}_g$ corresponding to the goal direction. The desired speed v_g is defined as the speed of the reference trajectory at $\vec{p}_{c'}$ normalized by the maximum speed achieved along t_g.

Data collection. To train the network, we collect a dataset of state estimates and corresponding camera images. Using the global reference trajectory, we evaluate the expert policy on each of these samples and use the result as the ground truth for training. An important property of this training procedure is that it is agnostic to how exactly the training dataset is collected. We use this flexibility to select the most suitable data collection method when training in simulation and in the real world. The key consideration here is how to deal with the domain shift between training and test time. In our scenario, this domain shift mainly manifests itself when the quadrotor flies far from the reference trajectory t_g. In simulation, we employed a variant of DAgger [240], which uses the expert policy to recover whenever the learned policy deviates far from the reference trajectory. Repeating the same procedure in the real world would be infeasible: allowing a partially trained network to control a UAV would pose a high risk of crashing and breaking the platform. Instead, we manually carried the quadrotor through the track and ensured a sufficient coverage of off-trajectory positions.

Generating data in simulation. In our simulation experiment, we perform a modified version of DAgger [240] to train our flying policy. On the data collected through the expert policy (Section C.3.1) (in our case we let the expert policy fly for 40 s), the network is trained for 10 epochs on the accumulated data. In the following run, the trained network is predicting actions, which are only executed if they keep the quadrotor within a margin ϵ from the global trajectory. In case the network's action violates this constraint, the expert policy is executed, generating a new training sample. This procedure is an automated form of DAgger [240] and allows the network to recover when deviating from the global trajectory. After another 40 s of data generation, the network is retrained on all the accumulated data for 10 epochs. As soon as the network performs well on a given margin ϵ, the margin is increased. This process repeats until the network can eventually complete the whole track without help of the expert policy. In our simulation experiments, the margin ϵ was set to 0.5 m after the first training iteration. The margin was incremented by 0.5 m as soon as the network could complete the track with limited help from the expert policy (less than 50 expert actions needed). For experiments on the static track, 20k images were collected, while for dynamic experiments 100k images of random gate positions were generated.

Generating data in the real world. For safety reasons, it is not possible to apply DAgger for data collection in the real world. Therefore, we ensure sufficient coverage of the possible actions by manually carrying the quadrotor through the track. During this procedure, which we call *handheld* mode, the expert policy is constantly generating training samples. Due to the drift of onboard state estimation, data is generated for a small part of the track before the quadrotor

is reinitialized at a known position. For the experiment on the static track, 25k images were collected, while for the dynamic experiment an additional 15k images were collected for different gate positions. For the narrow gap and occlusion experiments, 23k images were collected.

Loss function. We train the network with a weighted MSE loss on point and velocity predictions:

$$L = \|\vec{x} - \vec{x}_g\|^2 + \gamma(v - v_g)^2, \tag{C.1}$$

where $\vec{x}_g$ denotes the groundtruth normalized image coordinates and v_g denotes the groundtruth normalized speed. By cross-validation, we found the optimal weight to be $\gamma = 0.1$, even though the performance was mostly insensitive to this parameter (see Appendix for details).

Dynamic environments. The described training data generation procedure is limited to static environments, since the trajectory generation method is unable to take the changing geometry into account. How can we use it to train a perception system that would be able to cope with dynamic environments? Our key observation is that training on multiple static environments (for instance with varying gate positions) is sufficient to operate in dynamic environments at test time. We collect data from multiple layouts generated by moving the gates from their initial position. We compute a global reference trajectory for each layout and train a network jointly on all of these. This simple approach supports generalization to dynamic tracks, with the additional benefit of improving the robustness of the system.

Sim-to-real transfer. One of the big advantages of perception-control modularization is that it allows training the perception block exclusively in simulation and then directly applying on the real system by leaving the control part unchanged. As we will show in the experimental section, thanks to the abundance of simulated data, it is possible to train policies that are extremely robust to changes in environmental conditions, such as illumination, viewpoint, gate appearance, and background. In order to collect diverse simulated data, we perform visual scene randomization in the simulated environment, while keeping the approximate track layout fixed. Apart from randomizing visual scene properties, the data collection procedure remains unchanged.

We randomize the following visual scene properties: (i) the textures of the background, floor, and gates, (ii) the shape of the gates, and (iii) the lighting in the scene. For (i), we apply distinct random textures to background and floor from a pool of 30 diverse synthetic textures (Figure C.3a). The gate textures are drawn from a pool of 10 mainly red/orange textures (Figure C.3c). For gate shape randomization (ii), we create 6 gate shapes of roughly the same size as the original gate. Figure C.3d illustrates four of the different gate shapes used for data collection. To randomize illumination conditions (iii), we perturb the ambient and emissive light properties of all textures (background, floor, gates). Both properties are drawn separately for background, floor, and gates from uniform distributions with support $[0, 1]$ for the ambient property and $[0, 0.3]$ for the emissive property.

While the textures applied during data collection are synthetic, the textures applied to background

(a) (b) (c) (d)

Figure C.3 – To test the generalization abilities of our approach, we randomize the visual properties of the environment (background, illumination, gate shape, and gate texture). This figure illustrates the random textures and shapes applied both at training (**a**) and test time(**b**). For space reasons, not all examples are shown. In total, we used 30 random backgrounds during training and 10 backgrounds during testing. We generated 6 different shapes of gates and used 5 of them for data generation and one for evaluation. Similarly, we used 10 random gate textures during training and a different one during evaluation. **a)** Random backgrounds used during training data generation. **b)** Random backgrounds used at test time. **c)** Gate textures. **d)** Selection of training examples illustrating the gate shapes and variation in illumination properties.

and floor at test time represent common indoor and outdoor environments (Figure C.3b). For testing we use held-out configurations of gate shape and texture not seen during training.

C.3.2 Trajectory Generation

Generation of global trajectory. Both in simulation and in real-world experiments, a global trajectory is used to generate ground truth labels. To generate the trajectory, we use the implementation of Mellinger and Kumar [190]. The trajectory is generated by providing a set of waypoints to pass through, a maximum velocity to achieve, as well as constraints on maximum thrust and body rates. Note that the speed on the global trajectory is not constant. As waypoints, the centers of the gates are used. Furthermore, the trajectory can be shaped by additional waypoints, for example if it would pass close to a wall otherwise. In both simulation and real-world experiments, the maximum normalized thrust along the trajectory was set to $18\,\mathrm{m\,s^{-2}}$ and the maximum roll and pitch rate to $1.5\,\mathrm{rad\,s^{-1}}$. The maximum speed was chosen based on the dimensions of the track. For the large simulated track, a maximum speed of $10\,\mathrm{m\,s^{-1}}$ was chosen, while on the smaller real-world track $6\,\mathrm{m\,s^{-1}}$.

Generation of trajectory segments. The proposed navigation approach relies on constant recomputation of trajectory segments t_s based on the output of a CNN. Implemented as state-interception trajectories, t_s can be computed by specifying a start state, goal state and a desired execution time. The velocity predicted by the network is used to compute the desired execution time of the trajectory segment t_s. While the start state of the trajectory segment is fully defined by the quadrotor's current position, velocity, and acceleration, the end state is only constrained by the goal position p_g, leaving velocity and acceleration in that state unconstrained. This is, however, not an issue, since only the first part of each trajectory segment is executed in a receding horizon fashion. Indeed, any time a new network prediction is available, a new state interception trajectory t_s is calculated.

The goal position p_g is dependent on the prediction horizon d (see Section C.3.1), which directly influences the aggressiveness of a maneuver. Since the shape of the trajectory is only constrained by the start state and end state, reducing the prediction horizon decreases the lateral deviation from the straight-line connection of start state and end state but also leads to more aggressive maneuvers. Therefore, a long prediction horizon is usually required on straight and fast parts of the track, while a short prediction horizon performs better in tight turns and in proximity of gates. A long prediction horizon leads to a smoother flight pattern, usually required on straight and fast parts of the track. Conversely, a short horizon performs more agile maneuvers, usually required in tight turns and in the proximity of gates.

The generation of the goal position p_g differs from training to test time. At training time, the quadrotor's current position is projected onto the global trajectory and propagated by a prediction horizon d_{train}. At test time, the output of the network is back-projected along the camera projection ray by a planning length d_{test}.

At training time, we define the prediction horizon d_{train} as a function of distance from the last gate and the next gate to be traversed:

$$d_{train} = \max\left(d_{min}, \min\left(\|s_{last}\|, \|s_{next}\|\right)\right) , \tag{C.2}$$

where $s_{last} \in \mathbb{R}^3$ and $s_{next} \in \mathbb{R}^3$ are the distances to the corresponding gates and d_{min} represents the minimum prediction horizon. The minimum distance between the last and the next gate is used instead of only the distance to the next gate to avoid jumps in the prediction horizon after a gate pass. In our simulated track experiment, a minimum prediction horizon of $d_{min} = 1.5\,\mathrm{m}$ was used, while for the real track we used $d_{min} = 1.0\,\mathrm{m}$.

At test time, since the output of the network is a direction and a velocity, the length of a trajectory segment needs to be computed. To distinguish the length of trajectory segments at test time from the same concept at training time, we call it *planning length* at test time. The planning length of trajectory segments is computed based on the velocity output of the network (computation based on the location of the quadrotor with respect to the gates is not possible at test time since we do not have knowledge about gate positions). The objective is again to adapt the planning length such that both smooth flight at high speed and aggressive maneuvers in tight turns are possible. We achieve this versatility by computing the planning length according to this linear function:

$$d_{test} = \min\left[d_{max}, \max\left(d_{min}, m_d v_{out}\right)\right] , \tag{C.3}$$

where $m_d = 0.6\,\mathrm{s}$, $d_{min} = 1.0\,\mathrm{m}$ and $d_{max} = 2.0\,\mathrm{m}$ in our real-world experiments, and $m_d = 0.5\,\mathrm{s}$, $d_{min} = 2.0\,\mathrm{m}$ and $d_{max} = 5.0\,\mathrm{m}$ in the simulated track.

(a) (b)

Figure C.4 – Illustration of the simulated tracks. The small track (**a**) consists of 4 gates and spans a total length of 43 meters. The large track (**b**) consists of 8 gates placed at different heights and spans a total length of 116 meters.

C.4 Experiments

We extensively evaluate the presented approach in a wide range of simulated and real scenarios. We first use a controlled, simulated environment to test the main building blocks of our system, i.e. the convolutional architecture and the perception-control modularization. Then, to show the ability of our approach to control real quadrotors, we perform a second set of experiments on a physical platform. We compare our approach to state-of-the-art methods, as well as to human drone pilots of different skill levels. We also demonstrate that our system achieves zero-shot simulation-to-reality transfer. A policy trained on large amounts of cheap simulated data shows increased robustness against external factors, e.g. illumination and visual distractors, compared to a policy trained only with data collected in the real world. Finally, we perform an ablation study to identify the most important factors that enable successful policy transfer from simulation to the real world.

C.4.1 Experimental Setup

For all our simulation experiments we use Gazebo as the simulation engine. Although non-photorealistic, we have selected this engine since it models with high fidelity the physics of a quadrotor via the RotorS extension [77].

Specifically, we simulate the AscTec Hummingbird multirotor, which is equipped with a forward-looking 300×200 pixels RGB camera.

The platform is spawned in a flying space of cubical shape with side length of 70 meters, which contains the experiment-specific race track. The flying space is bounded by background and floor planes whose textures are randomized in the simulation experiments of Section C.4.5.

The large simulated race track (Figure C.4b) is inspired by a real track used in international competitions. We use this track layout for all of our experiments, except the comparison against end-to-end navigation policies. The track is travelled in the same direction (clockwise or counterclockwise) at training and testing time. We will release all code required to run our simulation experiments upon acceptance of this manuscript.

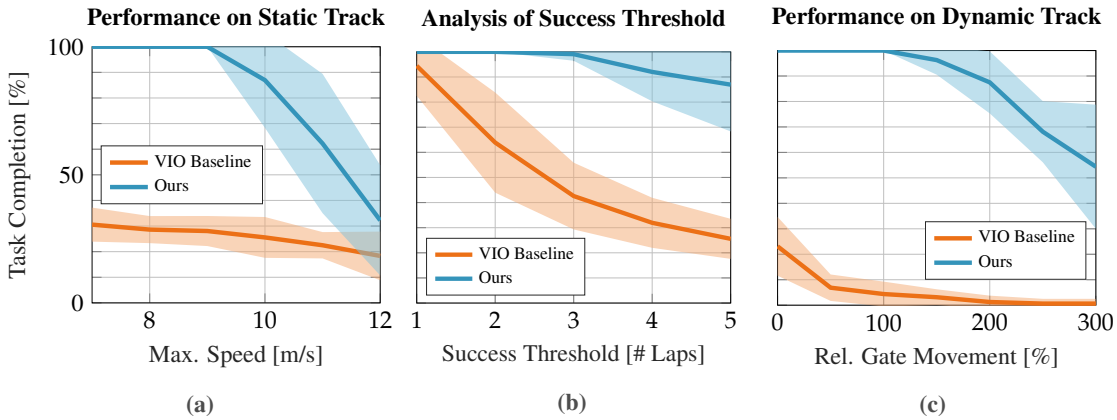


Figure C.5 – a) Results of simulation experiments on the large track with static gates for different maximum speeds. *Task completion rate* measures the fraction of gates that were successfully completed without crashing. A task completion rate of 100% is achieved if the drone can complete five consecutive laps without crashing. For each speed 10 runs were performed. **b)** Analysis of the influence of the choice of success threshold. The experimental setting is the same as in Figure C.5a, but the performance is reported for a fixed maximum speed of $10\,\mathrm{m\,s^{-1}}$ and different success thresholds. The y-axis is shared with Figure C.5a. **c)** Result of our approach when flying through a simulated track with moving gates. Every gate independently moves in a sinusoidal pattern with an amplitude proportional to its base size (1.3 m), with the indicated multiplier. For each amplitude 10 runs were performed. As for the static gate experiment, a task completion rate of 100% is achieved if the drone can complete five consecutive laps without crashing. Maximum speed is fixed to $8\,\mathrm{m\,s^{-1}}$. The y-axis is shared with Figure C.5a. Lines denote mean performance, while the shaded areas indicate one standard deviation. The reader is encouraged to watch the supplementary video to better understand the experimental setup and the task difficulty.

For real-world experiments, except for the ones evaluating sim-to-real transfer, we collected data in the real world. We used an in-house quadrotor equipped with an Intel UpBoard and a Qualcomm Snapdragon Flight Kit. While the latter is used for visual-inertial odometry, the former represents the main computational unit of the platform. The Intel UpBoard was used to run all the calculations required for flying, from neural network prediction to trajectory generation and tracking.

C.4.2 Experiments in Simulation

Using a controlled simulated environment, we perform an extensive evaluation to (i) understand the advantages of our approach with respect to end-to-end or classical navigation policies, (ii) test the system's robustness to structural changes in the environment, and (iii) analyze the effect of the system's hyper-parameters on the final performance.

Comparison to end-to-end learning approach. In our first scenario, we use a small track that consists of four gates in a planar configuration with a total length of 43 meters (Figure C.4a).

We use this track to compare the performance to a naive deep learning baseline that directly regresses body rates from raw images. Ground truth body rates for the baseline were provided by generating a minimum snap reference trajectory through all gates and then tracking it with a

low-level controller [61]. For comparability, this baseline and our method share the same network architecture. Our approach was always able to successfully complete the track. In contrast, the naive baseline could never pass through more than one gate. Training on more data (35K samples, as compared to 5K samples used by our method) did not noticeably improve the performance of the baseline. We believe that end-to-end learning of low-level controls [204] is suboptimal for the task of drone navigation when operating in the real world. Since a quadrotor is an unstable platform [208], learning the function that converts images to low-level commands has a very high sample complexity. Additionally, the network is constrained by computation time. In order to guarantee stable control, the baseline network would have to produce control commands at a higher frequency (typically 50 Hz) than the camera images arrive (30 Hz) and process them at a rate that is computationally infeasible with existing onboard hardware. In our experiments, since the low-level controller runs at 50 Hz, a network prediction is repeatedly applied until the next prediction arrives.

In order to allow on-board sensing and computing, we propose a modularization scheme which organizes perception and control into two blocks. With modularization, our approach can benefit from the most advanced learning based perceptual architectures and from years of study in the field of control theory [183]. Because body rates are generated by a classic controller, the network can focus on the navigation task, which leads to high sample efficiency. Additionally, because the network does not need to ensure the stability of the platform, it can process images at a lower rate than required for the low-level controller, which unlocks onboard computation. Given its inability to complete even this simple track, we do not conduct any further experiments with the direct end-to-end regression baseline.

Performance on a complex track. In order to explore the capabilities of our approach of performing high-speed racing, we conduct a second set of experiments on a larger and more complex track with 8 gates and a length of 116 meters (Figure C.4b). The quantitative evaluation is conducted in terms of average task completion rate over five runs initialized with different random seeds. For one run, the task completion rate linearly increases with each passed gate while 100% task completion is achieved if the quadrotor is able to successfully complete five consecutive laps without crashing. As a baseline, we use a pure feedforward setting by following the global trajectory t_g using state estimates provided by visual inertial odometry [73].

The results of this experiment are shown in Figure C.5a. We can observe that the VIO baseline, due to accumulated drift, performs worse than our approach. Figure C.5b illustrates the influence of drift on the baseline's performance. While performance is comparable when one single lap is considered a success, it degrades rapidly if the threshold for success is raised to more laps. On a static track (Figure C.5a), a SLAM-based state estimator [206, 227] would have less drift than a VIO baseline, but we empirically found the latency of existing open-source SLAM pipelines to be too high for closed-loop control. A benchmark comparison of latencies of monocular visual-inertial SLAM algorithms for flying robots can be found in [51].

Our approach works reliably up to a maximum speed of $9\,\mathrm{m\,s^{-1}}$ and performance degrades

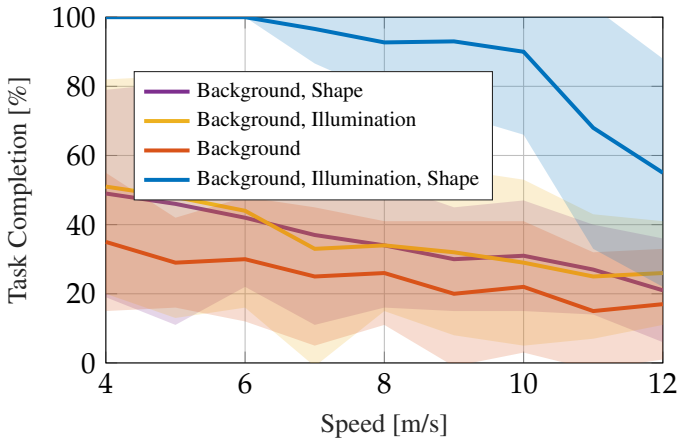


Figure C.6 – Generalization tests on different backgrounds after domain randomization. More comprehensive randomization increases the robustness of the learned policy to unseen scenarios at different speeds. Lines denote mean performance, while the shaded areas indicate one standard deviation. Background randomization has not been included in the analysis: without it the policy fails to complete even a single gate pass.

gracefully at higher velocities. The decrease in performance at higher speeds is mainly due to the higher body rates of the quadrotor that larger velocities inevitably entail. Since the predictions of the network are in the body frame, the limited prediction frequency (30 Hz in the simulation experiments) is no longer sufficient to cope with the large roll and pitch rates of the platform at high velocities.

Generalization to dynamic environments. The learned policy has a characteristic that the expert policy lacks of: the ability to cope with dynamic environments. To quantitatively test this ability, we reuse the track layout from the previous experiment (Figure C.4b), but dynamically move each gate according to a sinusoidal pattern in each dimension independently. Figure C.5c compares our system to the VIO baseline for varying amplitudes of gates' movement relative to their base size. We evaluate the performance using the same metric as explained in Section C.4.1. For this experiment, we kept the maximum platform velocity v_{max} constant at $8\,\mathrm{m\,s^{-1}}$. Despite the high speed, our approach can handle dynamic gate movements up to 1.5 times the gate diameter without crashing. In contrast, the VIO baseline cannot adapt to changes in the environment, and fails even for small gate motions up to 50% of the gate diameter. The performance of our approach gracefully degrades for gate movements larger than 1.5 times the gate diameter, mainly due to the fact that consecutive gates get too close in flight direction while being shifted in other directions. Such configurations require extremely sharp turns that go beyond the navigation capabilities of the system. From this experiment, we can conclude that the proposed approach reactively adapts to dynamic changes in the environment and generalizes well to cases where the track layout remains roughly similar to the one used to collect training data.

Generalization to changes in the simulation environment. In the previous experiments, we

have assumed a constant environment (background, illumination, gate shape) during data collection and testing. In this section, we evaluate the generalization abilities of our approach to environment configurations not seen during training. Specifically, we drastically change the environment background (Figure C.3b) and use gate appearance and illumination conditions held out at training time.

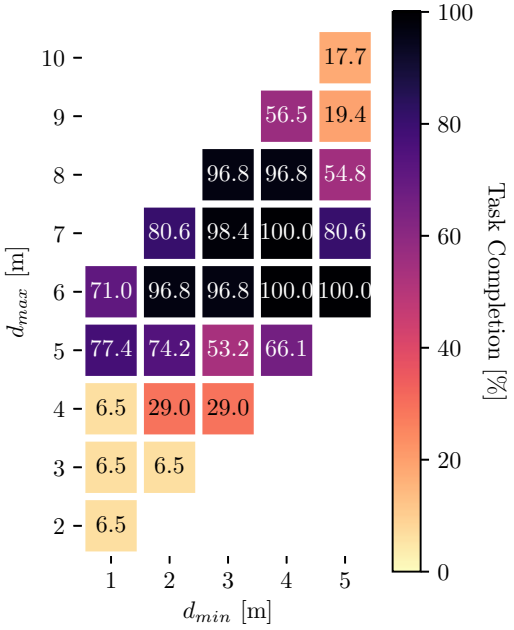


Figure C.7 – Sensitivity analysis of planning length parameters d_{min}, d_{max} on a simulated track. Maximum speed and (static) track layout are kept constant during the experiment.

Figure C.6 shows the result of this evaluation. As expected, if data collection is performed in a single environment, the resulting policy has limited generalization (red line). To make the policy environment-agnostic, we performed domain randomization while keeping the approximate track layout constant (details in Section C.3.1). Clearly, both randomization of gate shape and illumination lead to a policy that is more robust to new scenarios. Furthermore, while randomization of a single property leads to a modest improvement, performing all types of randomization simultaneously is crucial for good transfer. Indeed, the simulated policy needs to be invariant to all of the randomized features in order to generalize well.

Surprisingly, as we show below, the learned policy can not only function reliably in simulation, but is also able to control a quadrotor in the real world. In Section C.4.5 we present an evaluation of the real world control abilities of this policy trained in simulation, as well as an ablation study to identify which of the randomization factors presented above are the most important for generalization and knowledge transfer.

Sensitivity to planning length. We perform an ablation study of the *planning length* parameters d_{min}, d_{max} on a simulated track. Both the track layout and the maximum speed ($10.0\,\mathrm{m\,s^{-1}}$) are

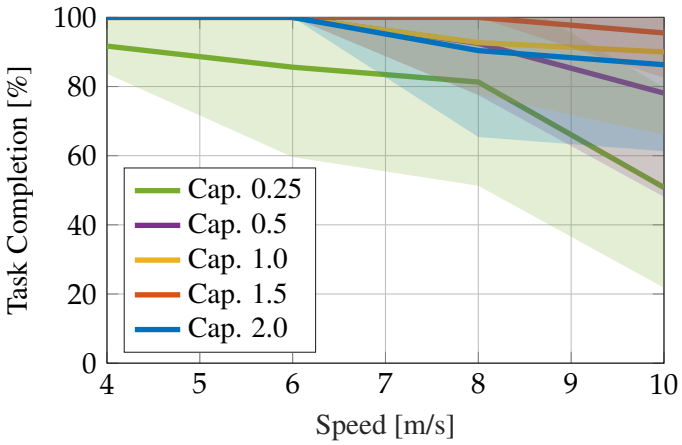


Figure C.8 – Comparison of different network capacities on different backgrounds after domain randomization.

kept constant in this experiment. We varied d_{min} between 1.0 m and 5.0 m and d_{max} between $(d_{min} + 1.0)$m and $(d_{min} + 5.0)$m. Figure C.7 shows the results of this evaluation. For each configuration the average *task completion rate* (Section C.4.1) over 5 runs is reported. Our systems performs well over a large range of d_{min}, d_{max}, with performance dropping sharply only for configurations with very short or very long planning lengths. This behaviour is expected, since excessively short planning lengths result in very aggressive maneuvers, while excessively long planning lengths restrict the agility of the platform.

C.4.3 Analysis of Accuracy and Efficiency

The neural network at the core of our perception system constitutes the biggest computational bottleneck of our approach. Given the constraints imposed by our processing unit, we can guarantee real-time performance only with relatively small CNNs. Therefore, we investigated the relationship between the capacity (hence the representational power) of a neural network and its performance on the navigation task. We measure performance in terms of both prediction accuracy on a validation set, and closed-loop control on a simulated platform, using, as above, completion rate as metric. The capacity of the network is controlled through a multiplicative factor on the number of filters (in convolutional layers) and number of nodes (in fully connected layers). The network with capacity 1.0 corresponds to the DroNet architecture [177].

Figure C.9 shows the relationship between the network capacity, its test loss (RMSE) on a validation set, and its inference time on an Intel UpBoard (our onboard processing unit). Given their larger parametrization, wider architectures have a lower generalization error but largely increase the computational and memory budget required for their execution. Interestingly, a lower generalization loss does not always correspond to a better closed-loop performance. This can be observed in Figure C.8, where the network with capacity 1.5 outperforms the one with capacity

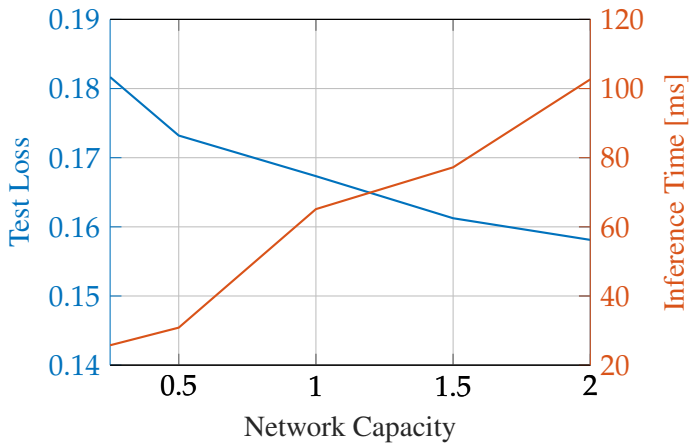


Figure C.9 – Test loss and inference time for different network capacity factors. Inference time is measured on the actual platform.

2.0 at high speeds. Indeed, as shown in Figure C.9, larger networks entail smaller inference rates, which result in a decrease in agility.

In our previous conference paper [139], we used a capacity factor of 1.0, which appears to have a good time-accuracy trade-off. However, in the light of this study, we select a capacity factor of 0.5 for all our new sim-to-real experiments to ease the computational burden. Indeed, the latter experiments are performed at a speed of $2\,\mathrm{m\,s^{-1}}$, where both 0.5 and 1.0 have equivalent closed-loop control performance (Figure C.8).

C.4.4 Experiments in the Real World

To show the ability of our approach to function in the real world, we performed experiments on a physical quadrotor. We compared our model to state-of-the-art classic approaches to robot navigation, as well as to human drone pilots of different skill levels.

Narrow gate passing. In the initial set of experiments the quadrotor was required to pass through a narrow gate, only slightly larger than the platform itself. These experiments are designed to test the robustness and precision of the proposed approach. An illustration of the setup is shown in Figure C.10. We compare our approach to the handcrafted window detector of Falanga et al. [66] by replacing our perception system with the handcrafted detector and leaving the control system unchanged.

Table C.1 shows a comparison between our approach and the baseline. We tested the robustness of both approaches to the initial position of the quadrotor by placing the platform at different starting angles with respect to the gate (measured as the angle between the line joining the center of gravity of the quadrotor and the gate, respectively, and the optical axis of the forward facing camera on the platform). We then measured the average success rate at passing the gate without

Figure C.10 – Setup of the narrow gap and occlusion experiments.

Relative Angle Range [°]	Handcrafted Detector	Network
$[0, 30]$	70%	100%
$[30, 70]$	0%	80%
$[70, 90]$*	0%	20%

Table C.1 – Success rate for flying through a narrow gap from different initial angles. Each row reports the average of ten runs uniformly spanning the range. The gate was completely invisible at initialization in the experiments marked with *.

crashing. The experiments indicate that our approach is not sensitive to the initial position of the quadrotor. The drone is able to pass the gate consistently, even if the gate is only partially visible. In contrast, the baseline sometimes fails even if the gate is fully visible because the window detector loses tracking due to platform vibrations. When the gate is not entirely in the field of view, the handcrafted detector fails in all cases.

In order to further highlight the robustness and generalization abilities of the approach, we perform experiments with an increasing amount of clutter that occludes the gate. Note that the learning approach has not been trained on such occluded configurations. Figure C.11 shows that our approach is robust to occlusions of up to 50% of the total area of the gate (Figure C.10), whereas the handcrafted baseline breaks down even for moderate levels of occlusion. For occlusions larger than 50% we observe a rapid drop in performance. This can be explained by the fact that the remaining gap was barely larger than the drone itself, requiring very high precision to successfully pass it. Furthermore, visual ambiguities of the gate itself become problematic. If just one of the edges of the window is visible, it is impossible to differentiate between the top and bottom part. This results in over-correction when the drone is very close to the gate.

Experiments on a race track. To evaluate the performance of our approach in a multi-gate scenario, we challenge the system to race through a track with either static or dynamic gates. The track is shown in Figure C.13. It is composed of four gates and has a total length of 21 meters.

To fully understand the potential and limitations of our approach, we compared to a number of baselines, such as a classic approach based on planning and tracking [171] and human pilots of

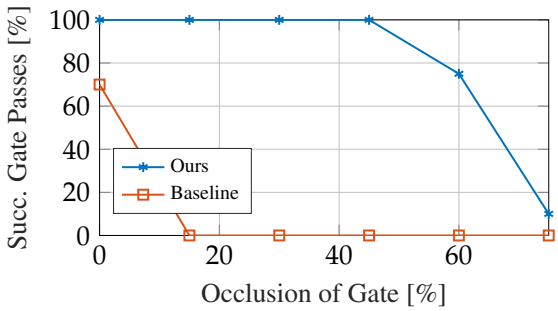


Figure C.11 – Success rate for different amounts of occlusion of the gate. Our method is much more robust than the baseline method that makes use of a hand-crafted window detector. Note that at more than 60% occlusion, the platform has barely any space to pass through the gap.

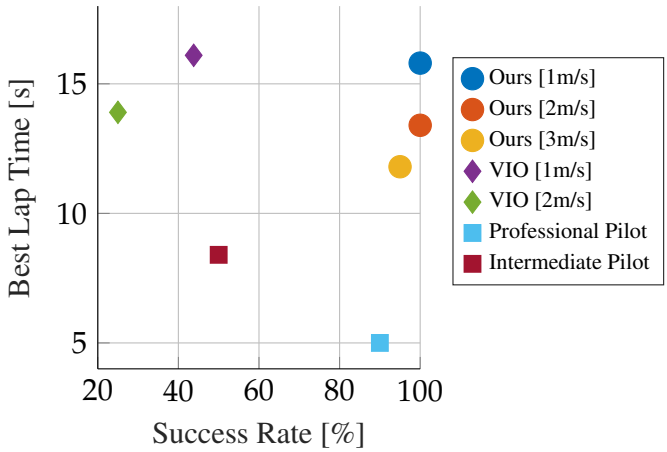


Figure C.12 – Results on a real race track composed of 4 gates. Our learning-based approach compares favorably against a set of baselines based on visual-inertial state estimation. Additionally, we compare against an intermediate and a professional human pilot. We evaluate success rate using the same metric as explained in Section C.4.1.

Figure C.13 – Track configuration used for the real world experiments.

different skill levels. Note that due to the smaller size of the real track compared to the simulated one, the maximum speed achieved in the real world experiments is lower than in simulation. For our baseline, we use a state-of-the-art visual-inertial odometry (VIO) approach [171] for state estimation in order to track the global reference trajectory.

Figure C.12 summarizes the quantitative results of our evaluation, where we measure success rate (completing five consecutive laps without crashing corresponds to 100%), as well as the best lap time. Our learning-based approach outperforms the VIO baseline, whose drift at high speeds inevitably leads to poor performance. In contrast, our approach is insensitive to state estimation drift, since it generates navigation commands in the body frame. As a result, it completes the track with higher robustness and speed than the VIO baseline.

In order to see how state-of-the-art autonomous approaches compare to human pilots, we asked a professional and an intermediate pilot to race through the track in first-person view. We allowed the pilots to practice the track for 10 laps before lap times and failures were measured (Table C.2). It is evident from Figure C.12 that both the professional and the intermediate pilots were able to complete the track faster than the autonomous systems. However, the high speed and aggressive flight by human pilots comes at the cost of increased failure rates. The intermediate pilot in particular had issues with the sharp turns present in the track, leading to frequent crashes. Compared with the autonomous systems, human pilots perform more agile maneuvers, especially

Method	Task Completion (Average)		Best lap time [s]	
	static	dynamic	static	dynamic
Ours	95%	95%	12.1	15.0
Professional Pilot	90%	80%	5.0	6.5

Table C.2 – Comparison of our approach with a professional human pilot on a static and a dynamic track. We evaluate the performance using the same metric as explained in Section C.4.1.

in sharp turns. Such maneuvers require a level of reasoning about the environment that our autonomous system still lacks.

Dynamically moving gates. We performed an additional experiment to understand the abilities of our approach to adapt to dynamically changing environments. In order to do so, we manually moved the gates of the race track (Figure C.13) while the quadrotor was navigating through it. Flying the track under these conditions requires the navigation system to reactively respond to dynamic changes. Note that moving gates break the main assumption of traditional high-speed navigation approaches [28, 76], specifically that the trajectory can be pre-planned in a static world. They could thus not be deployed in this scenario. Due to the dynamic nature of this experiment, we encourage the reader to watch the supplementary video available at http://youtu.be/8RILnqPxo1s. Table C.2 provides a comparison in term of task completion and lap time with respect to a professional pilot. Due to the gates' movement, lap times are larger than the ones recorded in static conditions. However, while our approach achieves the same performance with respect to crashes, the human pilot performs slightly worse, given the difficulties entailed by the unpredictability of the track layout. It is worth noting that training data for our policy was collected by changing the position of only a single gate, but the network was able to cope with movement of any gate at test time.

C.4.5 Simulation to Real World Transfer

We now attempt direct simulation-to-real transfer of the navigation system. To train the policy in simulation, we use the same process to collect simulated data as in Section C.4.1, i.e. randomization of illumination conditions, gate appearance, and background. The resulting policy, evaluated in simulation in Figure C.6, is then used without any finetuning to fly a real quadrotor. Despite the large appearance differences between the simulated environment (Figure C.3d) and the real one (Figure C.13), the policy trained in simulation via domain randomization has the ability to control the quadrotor in the real world. Thanks to the abundance of simulated data, this policy can not only be transferred from simulation to the real world, but is also more robust to changes in the environment than the policy trained with data collected on the real track. As can be seen in the supplementaty video, the policy learned in simulation can not only reliably control the platform, but is also robust to drastic differences in illumination and distractors on the track.

To quantitatively benchmark the policy learned in simulation, we compare it against a policy that was trained on real data. We use the same metric as explained in Section C.4.1 for this evaluation. All experiments are repeated 10 times and the results averaged. The results of this evaluation are shown in Figure C.14. The data that was used to train the "real" policy was recorded on the same track for two different illumination conditions, *easy* and *medium*. Illumination conditions are varied by changing the number of enabled light sources: 4 for the easy, 2 for the medium, and 1 for the difficult. The supplementary video illustrates the different illumination conditions.

The policy trained in simulation performs on par with the one trained with real data in experiments

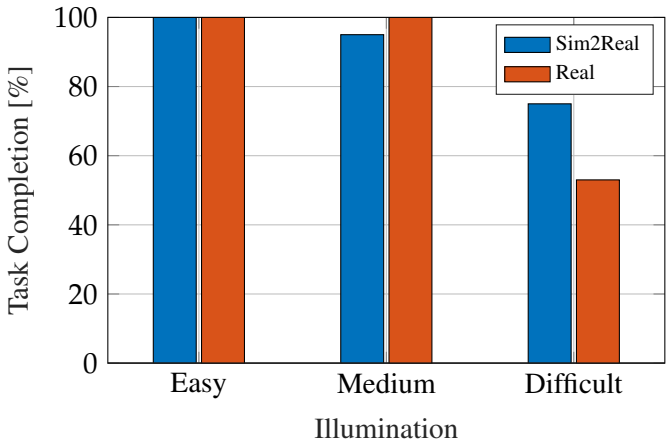


Figure C.14 – Performance comparison (measured with task completion rate) of the model trained in simulation and the one trained with real data. With *easy* and *medium* illumination (on which the real model was trained on), the approaches achieve comparable performance. However, with *difficult* illumination the simulated model outperforms the real one, since the latter was never exposed to this degree of illumination changes at training time. The supplementary video illustrates the different illumination conditions.

that have the same illumination conditions as the training data of the real policy. However, when the environment conditions are drastically different (i.e. with very challenging illumination) the policy trained with real data is outperformed by the one trained in simulation. Indeed, as shown by previous work [127], the abundance of simulated training data makes the resulting learning policy robust to environmental changes. We invite the reader to watch the supplementary video to understand the difficulty of this last set of experiments.

What is important for transfer? We conducted a set of ablation studies to understand what are the most important factors for transfer from simulation to the real world. In order to do so, we collected a dataset of real world images from both indoor and outdoor environments in different illumination conditions, which we then annotated using the same procedure as explained in Section C.3. More specifically, the dataset is composed of approximately 10K images and is collected from 3 indoor environments under different illumination conditions. Sample images of this dataset are shown in the appendix.

During data collection in simulation, we perform randomization of background, illumination conditions, and gate appearance (shape and texture). In this experiments, we study the effect of each of the randomized factors, except for the background which is well known to be fundamental for transfer [127, 279, 246]. We use as metric the Root Mean Square Error (RMSE) in prediction on our collected dataset. As shown in Figure C.15, illumination is the most important of the randomization factors, while gate shape randomization has the smallest effect. Indeed, while gate appearance is similar in the real world and in simulation, the environment appearance and illumination are drastically different. However, including more randomization is always beneficial for the robustness of the resulting policy (Figure C.6).

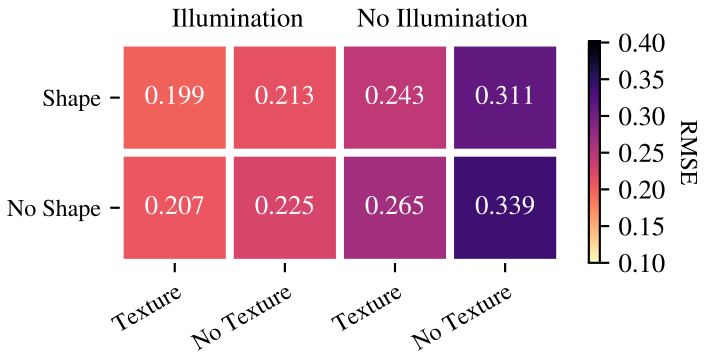


Figure C.15 – Average RMSE on testing data collected in the real world (lower is better). Headers indicate what is randomized during data collection.

C.5 Discussion and Conclusion

We have presented a new approach to autonomous, vision-based drone racing. Our method uses a compact convolutional neural network to continuously predict a desired waypoint and speed directly from raw images. These high-level navigation directions are then executed by a classic planning and control pipeline. As a result, the system combines the robust perceptual awareness of modern machine learning pipelines with the precision and speed of well-known control algorithms.

We investigated the capabilities of this integrated approach over three axes: precision, speed, and generalization. Our extensive experiments, performed both in simulation and on a physical platform, show that our system is able to navigate complex race tracks, avoids the problem of drift that is inherent in systems relying on global state estimates, and can cope with highly dynamic and cluttered environments.

Our previous conference work [139] required collecting a substantial amount of training data from the track of interest. Here instead we propose to collect diverse simulated data via domain randomization to train our perception policy. The resulting system can not only adapt to drastic appearance changes in simulation, but can also be deployed to a physical platform in the real world even if only trained in simulation. Thanks to the abundance of simulated data, a perception system trained in simulation can achieve higher robustness to changes in environment characteristics (e.g. illumination conditions) than a system trained with real data.

It is interesting to compare the two training strategies—on real data and sim-to-real—in how they handle ambiguous situations in navigation, for instance when no gate is visible or multiple gates are in the field of view. Our previous work [139], which was trained on the test track, could disambiguate those cases by using cues in the environment, for instance discriminative landmarks in the background. This can be seen as implicitly memorizing a map of the track in the network weights. In contrast, when trained only in simulation on multiple tracks (or randomized versions of the same track), our approach can no longer use such background cues to disambiguate the

flying direction and has instead to rely on a high-level map prior. This prior, automatically inferred from the training data, describes some common characteristics of the training tracks, such as, for instance, to always turn right when no gate is visible. Clearly, when ambiguous cases cannot be resolved with a prior of this type (e.g. an 8-shaped track), our sim-to-real approach would likely fail. Possible solutions to this problem are fine-tuning with data coming from the real track, or the use of a metric prior on the track shape to make decisions in ambiguous conditions [137].

Due to modularity, our system can combine model-based control with learning-based perception. However, one of the main disadvantages of modularity is that errors coming from each sub-module degrade the full system performance in a cumulative way. To overcome this problem, we plan to improve each component with experience using a reinforcement learning approach. This could increase the robustness of the system and improve its performance in challenging scenarios (e.g. with moving obstacles).

While our current set of experiments was conducted in the context of drone racing, we believe that the presented approach could have broader implications for building robust robot navigation systems that need to be able to act in a highly dynamic world. Methods based on geometric mapping, localization, and planning have inherent limitations in this setting. Hybrid systems that incorporate machine learning, like the one presented in this paper, can offer a compelling solution to this task, given the possibility to benefit from near-optimal solutions to different subproblems. However, scaling our proposed approach to more general applications, such as disaster response or industrial inspection, poses several challenges. First, due to the unknown characteristics of the path to be flown (layout, presence and type of landmarks, obstacles), the generation of a valid teacher policy would be impossible. This could be addressed with techniques such as *few-shot learning*. Second, the target applications might require extremely high agility, for instance in the presence of sharp turns, which our autonomous system still lacks of. This issue could be alleviated by integrating learning deeper into the control system [218].

C.6 Supplementary

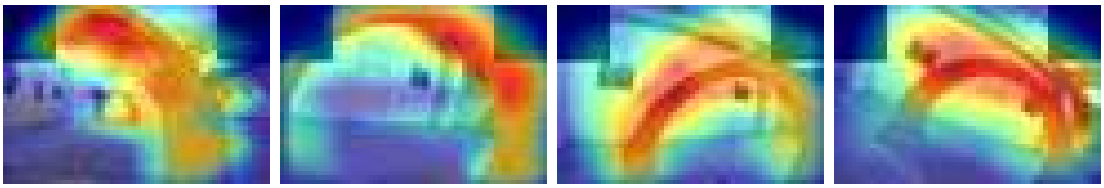


Figure C.16 – Visualization of network attention using the Grad-CAM technique [255]. Yellow to red areas correspond to areas of medium to high attention, while blue corresponds to areas of low attention. It is evident that the network learns to mostly focus on gates instead of relying on the background, which explains its capability to robustly handle dynamically moving gates. (Best viewed in color.)

Figure C.17 – Samples from dataset used in the ablation studies to quantify the importance of the randomization factors. (Best viewed in color.)

C.6.1 Gamma Evaluation

In this section, we examine the effect of the weighting factor γ in the loss function used to train our system (Eq. (C.1)). Specifically, we selected 7 values of γ in the range $[0.0001, 100]$ equispaced in logarithmic scale. Our network is then trained for 100 epochs on data generated from the static simulated track (Figure C.4b). After each epoch, performance is tested at a speed of $8\,\mathrm{m\,s^{-1}}$ according to the performance measure defined in C.4.1. Figure C.18 shows the results of this evaluation. The model is able to complete the track for all configurations after 80 epochs. Despite some values of γ lead to faster learning, we see that the system performance is not too sensitive to this weighting factor. Since $\gamma = 0.1$ proves to give the best results, we use it in all our experiments.

C.6.2 Network Architecture and Grad-CAM

We implement the perception system using a convolutional network. The input to the network is a 300×200 pixel RGB image, captured from the onboard camera at a frame rate of 30 Hz. After normalization in the $[0, 1]$ range, the input is passed through 7 convolutional layers, divided in 3 residual blocks, and a final fully connected layer that outputs a tuple $\{\vec{x}, v\}$. $\vec{x} \in [-1, 1]^2$ is a two-dimensional vector that encodes the direction to the new goal in normalized image coordinates and $v \in [0, 1]$ is a normalized desired speed to approach it.

To understand why the network is robust to previously unseen changes in the environment, we

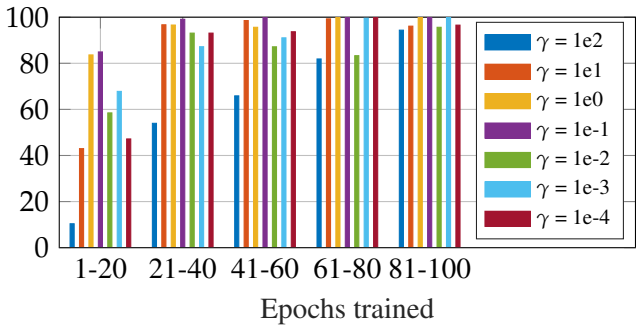


Figure C.18 – Success rate for different values of γ. For each γ, the network is trained up to 100 epochs. Performance is evaluated after each training epoch according to the performance criterion defined in C.4.1. For readability reasons, performance measurements are averaged over 20 epochs.

visualize the network's attention using the Grad-CAM technique [255] in Figure C.16. Grad-CAM visualizes which parts of an input image were important for the decisions made by the network. It becomes evident that the network bases its decision mostly on the visual input that is most relevant to the task at hand – the gates – while mostly ignoring the background.

C.6.3 Additional Evaluation Dataset

To quantify the performance of the policy trained in simulation to zero-shot generalization in real world scenarios, we collected a dataset of approximately 10k images from the real world. This dataset was collected from three indoor environments of different dimension and appearance. During data collection, illumination conditions differ either for intra-day variations in natural light or for the deployment of artificial light sources. To generate ground truth, we use the same annotation process as described in Section C.3. Some samples from this dataset are shown in Fig. C.17.

D A General Framework for Uncertainty Estimation in Deep Learning

Reprinted, with permission, from:

Antonio Loquercio, Mattia Segù, and Davide Scaramuzza. “A General Framework for Uncertainty Estimation in Deep Learning”. In: *IEEE Robotics Autom. Lett.* 5.2 (2020), pp. 3153–3160. DOI: 10.1109/LRA.2020.2974682

A General Framework for Uncertainty Estimation in Deep Learning

Antonio Loquercio, Mattia Segu and Davide Scaramuzza

Abstract — Neural networks predictions are unreliable when the input sample is out of the training distribution or corrupted by noise. Being able to detect such failures automatically is fundamental to integrate deep learning algorithms into robotics. Current approaches for uncertainty estimation of neural networks require changes to the network and optimization process, typically ignore prior knowledge about the data, and tend to make over-simplifying assumptions which underestimate uncertainty. To address these limitations, we propose a novel framework for uncertainty estimation. Based on Bayesian belief networks and Monte-Carlo sampling, our framework not only fully models the different sources of prediction uncertainty, but also incorporates prior data information, e.g. sensor noise. We show theoretically that this gives us the ability to capture uncertainty better than existing methods. In addition, our framework has several desirable properties: (i) it is *agnostic* to the network architecture and task; (ii) it does not require changes in the optimization process; (iii) it can be applied to *already trained* architectures. We thoroughly validate the proposed framework through extensive experiments on both computer vision and control tasks, where we outperform previous methods by up to 23% in accuracy.

Supplementary Material

The video available at https://youtu.be/X7n-bRS5vSM shows qualitative results of our experiments. The project's code is available at: https://tinyurl.com/s3nygw7

Figure D.1 – A neural network trained for steering angle prediction can be fully functional on a clean image (left) but generate unreliable predictions when processing a corrupted input (right). In this work, we propose a general framework to associate each network prediction with an uncertainty (illustrated above in red) that allows to detect such failure cases automatically.

D.1 Introduction

Robots act in an uncertain world. In order to plan and make decisions, autonomous systems can only rely on noisy perceptions and approximated models. Wrong decisions not only result in the failure of the mission but might even put human lives at risk, e.g., if the robot is an autonomous car (Fig. D.1). Under these conditions, deep learning algorithms can be fully integrated into robotic systems only if a measure of prediction uncertainty is available. Indeed, estimating uncertainties enables Bayesian sensor fusion and provides valuable information during decision making [275].

Prediction uncertainty in deep neural networks generally derives from two sources: *data* uncertainty and *model* uncertainty. The former arises because of noise in the data, usually caused by the sensors' imperfections. The latter instead is generated from unbalances in the training data distribution. For example, a rare sample should have higher model uncertainty than a sample which appears more often in the training data. Both components of uncertainty play an important role in robotic applications. A sensor can indeed never be assumed to be noise free, and training datasets cannot be expected to cover all the possible edge-cases.

Traditional approaches for uncertainty estimation model the network activations and weights by parametric probability distributions. However, these approaches are particularly difficult to train [111] and are rarely used in robotic applications. Another family of approaches estimate uncertainties through sampling [80]. Since they do not explicitly model data uncertainty, these methods generate over-confident predictions [123]. In addition, methods based on sampling generally disregard any relationship between data and model uncertainty, which increases the risk of underestimating uncertainties. Indeed, an input sample with large noise should have larger model uncertainty than the same sample with lower noise.

In this paper, we propose a novel framework for uncertainty estimation of deep neural network predictions. By combining Bayesian belief networks [75, 87, 24] with Monte-Carlo sampling, our framework captures prediction uncertainties better than state-of-the-art methodologies. In order to do so, we propose two main innovations with respect to previous works: the use of prior information about the data, e.g., sensor noise, to compute data uncertainty, and the modelling of the relationship between data and model uncertainty. We demonstrate both theoretically and experimentally that these two innovations allow our framework to produce higher quality uncertainty estimates than state-of-the-art methods. In addition, our framework has some desirable properties: (i) it is *agnostic* to the neural network architecture and task; (ii) it does not require any change in the optimization or learning process, and (iii) it can be applied to an *already trained* neural network. These properties make our approach an appealing solution to learning-based perception or control algorithms, enabling them to be better integrated into robotic systems.

To show the generality of our framework, we perform experiments on four challenging tasks: end-to-end steering angle prediction, obstacle future motion prediction, object recognition, and closed-loop control of a quadrotor. In these tasks, we outperform existing methodologies for uncertainty estimation by up to 23% in term of prediction accuracy. However, our framework is not limited to these problems and can be applied, without any change, to a wide range of tasks. Overall, our work makes the following contributions:

- We propose a general framework to compute uncertainties of neural networks predictions. Our framework is general in that it is agnostic to the network architecture, does not require changes in the learning or optimization process, and can be applied to already trained neural networks.
- We show mathematically that our framework can capture data and model uncertainty and use prior information about the data.
- We experimentally show that our approach outperforms existing methods for uncertainty estimation on a diverse set of tasks.

D.2 Related Work

In the following, we discuss the methods that have been proposed to estimate uncertainties and a series of approaches which have used this information in robotic systems.

D.2.1 Estimating Uncertainties in Neural Networks Predictions

A neural network is generally composed of a large number of parameters and non-linear activation functions, which makes the (multi-modal) posterior distribution of a network predictions intractable. To approximate the posterior, existing methods deploy different techniques, mainly based on Bayesian inference and Monte-Carlo sampling.

To recover probabilistic predictions, Bayesian approaches represent neural networks weights through parametric distributions, e.g., exponential-family [20, 75, 111, 292]. Consequently, networks' predictions can also be represented by the same distributions, and can be analytically computed using non-linear belief networks [75] or graphical models [271]. More recently, Wang et al. [292] propose natural parameter networks, which model inputs, parameters, nodes, and targets by Gaussian distributions. Overall, these family of approaches can recover uncertainties in a principled way. However, they generally increase the number of trainable parameters in a super-linear fashion, and require specific optimization techniques [111] which limits their impact in real-world applications.

In order to decrease the computational burden, Gast et al. [87] proposed to replace the network's input, activations, and outputs by distributions, while keeping network's weights deterministic. Similarly, probabilistic deep state space models retrieve data uncertainty in sequential data, and use it for learning-based filtering [74, 298]. However, disregarding weights uncertainty generally results in over-confident predictions, in particular for inputs not well represented in the training data.

Instead of representing neural networks parameters and activations by probability distributions, a second class of methods proposed to use Monte-Carlo (MC) sampling to estimate uncertainty. The MC samples are generally computed using an ensemble of neural networks. The prediction ensemble could either be generated by differently trained networks [143, 133, 156], or by keeping drop-out at test-time [80]. While this class of approaches can represent well the multi-modal posterior by sampling, it cannot generally represent data uncertainty, due for example to sensor noise. A possible solution is to tune the dropout rates [81], however it is always possible to construct examples where this approach would generate erroneous predictions [123].

To model data uncertainty, Kendall et al. [141] proposed to add to each output a "variance" variable, which is trained by a maximum-likelihood (a.k.a. heteroscedastic) loss on data. Combined with Monte-Carlo sampling, this approach can predict both the model and data uncertainty. However, this method requires to change the architecture, due to the variance addition, and to use the heteroscedastic loss for training, which is not always feasible.

Akin to many of the aforementioned methods, we use Monte-Carlo samples to predict model uncertainty. Through several experiments, we show why this type of uncertainty, generally ignored or loosely modelled by Bayesian methods [87], cannot be disregarded. In addition to Monte-Carlo sampling, our approach also computes the prediction uncertainty due to the sensors noise by using gaussian belief networks [75] and assumed density filtering [24]. Therefore, our approach can recover the full prediction uncertainty for any given (and possible already trained) neural network, without requiring any architectural or optimization change.

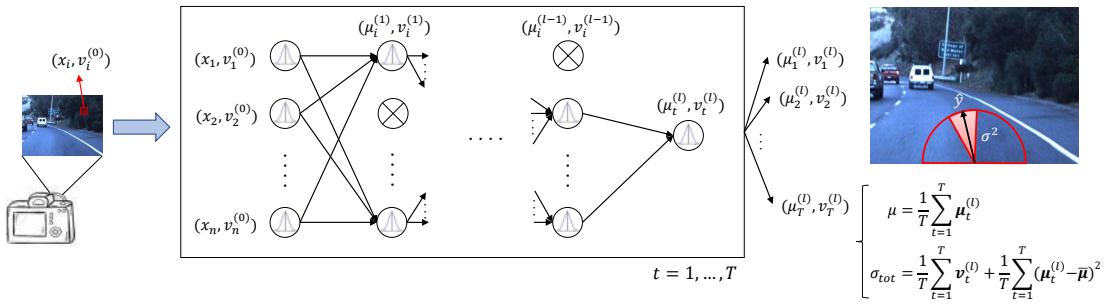


Figure D.2 – Given an input sample $\mathbf{x}$, associated with noise $\mathbf{v}^{(0)}$, and a trained neural network, our framework computes the confidence associated to the network output. In order to do so, it first transforms the given network into a Bayesian belief network. Then, it uses an ensemble of T such networks, created by enabling dropout at test time, to generate the final prediction $\boldsymbol{\mu}$ and uncertainty σ_{tot}.

D.2.2 Uncertainty Estimation in Robotics

Given the paramount importance of safety, autonomous driving research has allocated a lot of attention to the problem of uncertainty estimation, from both the perception [68, 210] and the control side [143, 133]. Feng. et al. [68] showed an increase in performance and reliability of a 3D Lidar vehicle detection system by adding uncertainty estimates to the detection pipeline. Predicting uncertainty was also shown to be fundamental to cope with sensor failures in autonomous driving [143], and to speed-up the reinforcement learning process on a real robot [133].

For the task of autonomous drone racing, Kaufmann et al. [137] demonstrated the possibility to combine optimal control methods to a network-based perception system by using uncertainty estimation and filtering. Also for the task of robot manipulation, uncertainty estimation was shown to play a fundamental role to increase the learning efficiency and guarantee the manipulator safety [269, 41].

In order to fully integrate deep learning into robotics, learning systems should reliably estimate the uncertainty in their predictions [275]. Our framework represents a minimally invasive solution to this problem: we do not require any architectural changes or re-training of existing models.

D.3 Methodology

Due to the large number of (possibly non-linear) operations required to generate predictions, the posterior distribution $p(\mathbf{y}|\mathbf{x})$, where $\mathbf{y}$ are output predictions and $\mathbf{x}$ input samples, is intractable. Formally, we define the total prediction uncertainty as $\sigma_{tot} = \mathrm{Var}_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y})$. This uncertainty comes from two sources: data and model uncertainty. In order to estimate σ_{tot}, we derive a tractable approximation of $p(\mathbf{y}|\mathbf{x})$. In the following, we present the derivation of this approximation by using Bayesian inference, and the resulting algorithm to predict σ_{tot} (illustrated in Fig. D.2).

D.3.1 The data uncertainty

Sensors' observations, e.g. images, are generally corrupted by noise. Therefore, what a neural network processes as input is $\mathbf{z}$, a noisy version of the "real" input $\mathbf{x}$. We assume that the sensor has known noise characteristic $\mathbf{v}$, which can be generally acquired by system identification or hardware specifications. Given $\mathbf{v}$, we assume the input data distribution $q(\mathbf{z}|\mathbf{x})$ to be:

$$q(\mathbf{z}|\mathbf{x}) \sim \mathrm{N}\left(\mathbf{z}; \mathbf{x}, \mathbf{v}\right). \tag{D.1}$$

The output uncertainty resulting from this noise is generally defined as *data* (or aleatoric) uncertainty.

In order to compute data uncertainty, we forward-propagate sensor noise through the network via Assumed Density Filtering (ADF) [24]. This approach, initially applied to neural networks by Gast et al. [87], replaces each network activation, including input and output, by probability distributions. Specifically, the joint density of all activations in a network with l layers is:

$$p(\mathbf{z}^{(0:l)}) = p(\mathbf{z}^{(0)}) \prod_{i=1}^{l} p(\mathbf{z}^{(i)}|\mathbf{z}^{(i-1)}) \tag{D.2}$$

$$p(\mathbf{z}^{(i)}|\mathbf{z}^{(i-1)}) = \delta[\mathbf{z}^{(i)} - \mathbf{f}^{(i)}(\mathbf{z}^{(i-1)})] \tag{D.3}$$

where $\delta[\cdot]$ is the Dirac delta and $\mathbf{f}^{(i)}$ the i-th network layer. Since this distribution is intractable, ADF approximates it with:

$$p(\mathbf{z}^{(0:l)}) \approx q(\mathbf{z}^{(0:l)}) = q(\mathbf{z}^{(0)}) \prod_{i=1}^{l} q(\mathbf{z}^{(i)}) \tag{D.4}$$

where $q(\mathbf{z})$ is normally distributed, with all components independent:

$$q(\mathbf{z}^{(i)}) \sim \mathrm{N}\left(\mathbf{z}^{(i)}; \boldsymbol{\mu}^{(i)}, \mathbf{v}^{(i)}\right) = \prod_{j} \mathrm{N}\left(\mathbf{z}_j^{(i)}; \boldsymbol{\mu}_j^{(i)}, \mathbf{v}_j^{(i)}\right). \tag{D.5}$$

The activation $\mathbf{z}^{(i-1)}$ is then processed by the (possibly non-linear) i-th layer function, $\mathbf{f}^{(i)}$, which transforms it into the (not necessarily normal) distribution:

$$\hat{p}(\mathbf{z}^{(0:i)}) = p(\mathbf{z}^{(i)}|\mathbf{z}^{(i-1)}) q(\mathbf{z}^{(0:i-1)}). \tag{D.6}$$

The goal of ADF is then to find the distribution $q(\mathbf{z}^{(0:i)})$ which better approximates $\hat{p}(\mathbf{z}^{(0:i)})$ under some measure, e.g. Kullback-Leibler divergence:

$$q(\mathbf{z}^{(0:i)}) = \underset{\hat{q}(\mathbf{z}^{(0:i)})}{\arg\min}\, \mathrm{KL}(\hat{q}(\mathbf{z}^{(0:i)}) \parallel \hat{p}(\mathbf{z}^{(0:i)})) \tag{D.7}$$

Minka et al. [195] showed that the solution to (D.7) requires matching the moments of the two

distributions. Under the normality assumptions, this is equivalent to:

$$\boldsymbol{\mu}^{(i)} = \mathbb{E}_{q(\mathbf{z}^{(i-1)})}[\,\mathbf{f}^{(i)}(z^{(i-1)})] \tag{D.8}$$

$$\mathbf{v}^{(i)} = \mathbb{V}_{q(\mathbf{z}^{(i-1)})}[\,\mathbf{f}^{(i)}(z^{(i-1)})] \tag{D.9}$$

where $\mathbb{E}$ and $\mathbb{V}$ are the first and second moment of the distribution. The solution of Eq. (D.8) and Eq. (D.9) can be computed analytically for the majority of functions used in neural networks, e.g. convolution, de-convolutions, relu, etc, and has an approximated solution for max-pooling. This results in a recursive formula to compute the activations mean and uncertainty, $(\boldsymbol{\mu}^{(i)}, \mathbf{v}^{(i)})$, given the parameters of the previous activations distribution $q(\mathbf{z}^{(i-1)})$. We refer the reader to [87, 75, 49] for the details of the propagation formulas.

In summary, ADF modifies the forward pass of a neural network to generate not only output predictions $\boldsymbol{\mu}^{(l)}$, but also their respective data uncertainties $\mathbf{v}^{(l)}$. In order to do so, ADF propagates the input uncertainty $\mathbf{v} = \mathbf{v}^{(0)}$, which, in a robotics scenario, corresponds to the sensor noise characteristics.

D.3.2 The model uncertainty

Model (or epistemic) uncertainty refers to the confidence a model has about its prediction. Similarly to Bayesian approaches [53, 181, 209, 268, 80], we represent this uncertainty by placing a distribution over the neural network weights, $\boldsymbol{\omega}$. This distribution depends on the training dataset $\mathrm{D} = \{\mathbf{X}, \mathbf{Y}\}$, where $\mathbf{X}, \mathbf{Y}$ are training samples and labels, respectively. Therefore, the weight distribution after training can be written as $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$.

Except in trivial cases, $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ is intractable. In order to approximate this distribution, Monte-Carlo based approaches collect weights samples by using dropout at test time [268, 80, 141]. Formally, this entails to approximate:

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \approx q(\boldsymbol{\omega}; \Phi) = Bern(\boldsymbol{\omega}; \Phi) \tag{D.10}$$

where Φ are the Bernoulli (or dropout) rates on the weights. Under this assumption, the model uncertainty is the variance of T Monte-Carlo samples, i.e. [80]:

$$\mathrm{Var}^{model}_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y}) = \sigma_{model} = \frac{1}{T}\sum_{t=1}^{T}(\mathbf{y}_t - \bar{\mathbf{y}})^2 \tag{D.11}$$

where $\{\mathbf{y}_t\}_{t=1}^{T}$ is a set of T sampled outputs for weights instances $\boldsymbol{\omega}^t \sim q(\boldsymbol{\omega}; \Phi)$ and $\bar{\mathbf{y}} = {}^1/_T \sum_t \mathbf{y}_t$.

Eq. D.11 has an intuitive explanation: Due to over-parametrization, a network develops redundant representations of samples frequently observed in the training data. Because of the redundancy, predictions for those samples will remain approximately constant when a part of the network is

switched off with dropout. Consequently, these samples will receive a low model uncertainty. In contrast, for rare samples the network is not able to generate such redundancies. Therefore, it will associate them with high model uncertainty.

D.3.3 Model uncertainty of an already trained network

The optimal dropout rates Φ needed to compute σ_{model} are the minimizers of the distance between the real and the hypothesis weight distribution:

$$\Phi = \arg\min_{\hat{\Phi}} \mathrm{KL}(p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y}) \parallel q(\boldsymbol{\omega};\hat{\Phi})). \tag{D.12}$$

Previous works showed that the best Φ corresponds to the training dropout rates [268, 80, 141]. This, however, hinders the computation of model uncertainty for networks trained without dropout.

Since re-training a given network with a specific rate Φ is not always possible for several applications, we propose to find the best Φ *after training* by minimizing the negative log-likelihood between predicted and ground-truth labels. This is justified by the following lemma:

Lemma D.3.1. *The dropout rates Φ minimizing Eq. (D.12), under normality assumption on the output, are equivalent to:*

$$\Phi = \arg\min_{\hat{\Phi}} \sum_{d\in D} \frac{1}{2}\log(\sigma_{tot}^{d}) + \frac{1}{2\sigma_{tot}^{d}}(\mathbf{y}_{gt}^{d} - \mathbf{y}_{pred}^{d}(\hat{\Phi}))^2. \tag{D.13}$$

Proof. A successfully trained network $p_{net}(\mathbf{y}|\mathbf{x},\boldsymbol{\omega})$ can very well predict the ground-truth, *i.e.*:

$$p_{gt}(\mathbf{y}|\mathbf{x}) \approx p_{pred}(\mathbf{y}|\mathbf{x}) = \int_{\boldsymbol{\omega}} p_{net}(\mathbf{y}|\mathbf{x},\boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y}). \tag{D.14}$$

By approximating $p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})$ by $q(\boldsymbol{\omega}|\Phi)$, *i.e.*, putting dropout only at test time, the real predicted distribution is actually:

$$\hat{p}_{pred}(\mathbf{y}|\mathbf{x};\Phi) = \int_{\boldsymbol{\omega}} p_{net}(\mathbf{y}|\mathbf{x},\boldsymbol{\omega}) q(\boldsymbol{\omega}|\Phi). \tag{D.15}$$

Since $p_{net}(\cdot)$ in Eq. (D.14) and Eq. (D.15) are the same, minimizing $\mathrm{KL}(p_{gt}(\mathbf{y}|\mathbf{x}) \parallel \hat{p}_{pred}(\mathbf{y}|\mathbf{x};\Phi))$ is equivalent to minimizing Eq. (D.12). Assuming that both p_{net} and p_{gt} are normal, and that $\sigma_{gt} \to 0$ (i.e. ground-truth is quasi-deterministic), the distance between the predicted and ground-truth distribution is equivalent to Eq. (D.13). □

Practically, Φ is found by grid-search on a log-range of 20 possible rates in the range $[0,1]$.

D.3.4 The total uncertainty

Section D.3.1 shows that ADF can be used to propagate sensor uncertainties to the network outputs. This is equivalent to model the output distribution $p(\mathbf{y}|\mathbf{x}) \approx p(\mathbf{y}|\mathbf{z}, \boldsymbol{\omega})p(\mathbf{z}|\mathbf{x})$, where $\boldsymbol{\omega}$ are deterministic network parameters and $p(\mathbf{z}|\mathbf{x})$ the sensor noise characteristics. Instead, Section D.3.2 shows that model uncertainty can be computed by putting a distribution on the network weights $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$. The total uncertainty σ_{tot} results from the combination of the model and data uncertainty. It can be computed through a stochastic version of ADF, as presented in the following lemma.

Lemma D.3.2. *The total variance of a network output* $\mathbf{y}$ *for an input sample* $\mathbf{x}$ *corrupted by noise* $\mathbf{v}^{(0)}$ *is:*

$$\sigma_{tot} = Var_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y}) = \frac{1}{T}\sum_{t=1}^{T} \mathbf{v}_t^{(l)} + (\boldsymbol{\mu}_t^{(l)} - \bar{\boldsymbol{\mu}})^2 \tag{D.16}$$

where $\{\boldsymbol{\mu}_t^{(l)}, \mathbf{v}_t^{(l)}\}_{t=1}^{T}$ *is a set of* T *outputs from the ADF network with weights* $\boldsymbol{\omega}^t \sim q(\boldsymbol{\omega}; \Phi)$ *and* $\bar{\boldsymbol{\mu}} = 1/T \sum_t \boldsymbol{\mu}_t^l$.

Its proof can be found in the supplementary material. Intuitively, Eq. (D.16) generates the total uncertainty by summing the two components of data and model uncertainty. Note the difference between Eq. (D.11) and our model uncertainty, $1/T \sum_{t=1}^{T}(\boldsymbol{\mu}_t^{(l)} - \bar{\boldsymbol{\mu}})^2$. Differently from Eq. (D.11), the prediction ensemble used to calculate the model variance is not generated with network outputs $\mathbf{y}_t$, but with ADF predictions $\boldsymbol{\mu}_t^{(l)}$. Consequently, the model uncertainty also depends on the input sensor noise $\mathbf{v}^{(0)}$. Indeed, this is a very intuitive result: even though a sample has been frequently observed in the training data, it should have large model uncertainty if corrupted by high noise. From Lemma D.3.2 we derive a computationally feasible algorithm to compute, at the same time, predictions and total uncertainties. Illustrated in Fig. D.2, the algorithm is composed of three main steps: (i) transforming a neural network into its ADF version (which does not require re-training), (ii) collect T samples by forwarding $(\mathbf{x}, \mathbf{v}^{(0)})$ to the network with $\boldsymbol{\omega}^t \sim q(\boldsymbol{\omega}; \Phi)$ and (iii) compute output predictions and variances according to lemma D.3.2.

It is interesting to draw a connection between Eq. (D.16) and the total uncertainty formulas used by previous works. Gast et al. [80], for example, do not use ADF networks to collect Monte-Carlo samples, and substitutes to the data uncertainty $\mathbf{v}^{(l)}$ a user-defined constant σ_d. Using a constant for the data uncertainty is nevertheless problematic, since different input samples might have different noise levels (due to, for example, temperature or pressure changes). Manually tuning this constant is generally difficult in practice, since it is not possible to use prior information about sensor noise characteristics. This makes it less attractive to robotics applications, where this information is either available or can be retrieved via identification.

In order to increase adaptation, Kendall et al. [141] proposed to learn the data uncertainty from the data itself. However, this comes at the cost of modifying the architecture and the training process, which hinders its application to already trained models and generally results in performance

Figure D.3 – On well illuminated frames where the car is driving straight, a network trained to predict steering angles is very certain about its output (left). In contrast, for poorly illuminated, ambiguous frames, the network is highly uncertain about its predictions (right).

drops. Moreover, it considers the model and data uncertainty to be completely independent, which is a overly-restrictive assumption in many cases. For example, high sensor noise can result in large model uncertainty, in particular if the model was never exposed, at training time, to such kind of noise levels. In contrast, our approach can model this dependence, since it uses ADF samples to compute model uncertainty. We refer the reader to the proof of Lemma D.3.2 for the formal justification of the previous statements.

D.4 Experiments

We validate our framework to compute uncertainties on several computer vision and robotic tasks. Specifically, as demonstrators we select end-to-end steering angle prediction, object future motion prediction, object recognition, and model-error compensation for autonomous drone flight. These demonstrators encompass the most active areas of research in mobile robotics, from computer vision to learning-based control. For each application, we compare against state-of-the-art methods for uncertainty estimation both qualitatively and quantitatively. All training details are reported in the appendix.

D.4.1 Demonstrators

End-to-End Steering Angle Prediction: Neural networks trained to predict steering angles from images have been used in several robotics applications, e.g. autonomous driving [21, 202] and flying [177]. In order to account for the safety of the platform, however, previous works showed the importance of quantifying uncertainties [133, 143]. In this section, we show that our framework can be used to estimate uncertainty without losing prediction performance on steering prediction.

To predict steering angles, we use the DroNet architecture of Loquercio et al. [177], since it was

Method	Re-train	RMSE	EVA	NLL
Gal et al. [80]	Yes	0.09	**0.83**	-0.72
Gast et al. [87]	Yes	0.10	0.79	-0.89
Kendall et al. [141]	Yes	0.11	0.75	**-1.1**
Ours	No	**0.09**	0.81	-1.0

Table D.1 – Benchmark comparison against state-of-the-art methods for variance estimation on the end-to-end steering angle prediction task.

shown to allow closed-loop control on several unmanned aerial platforms [217, 139]. Differently from DroNet, however, we add a dropout layer after each convolution or fully connected layer. This is indeed necessary to extract Monte-Carlo samples. In order to show that our approach can estimate uncertainties from already trained networks, dropout is only activated *at test time*. We train this architecture on the Udacity dataset [177], which provides labelled images collected from a car in a large set of environments, illumination and traffic conditions. As it is the standard for the problem [21, 177], we train the network with mean-squared-error loss $\left\|y_{gt} - y_{pred}\right\|^2$, where y_{gt} and y_{pred} are the ground-truth and estimated steerings. For evaluation, we measure performance with Explained Variance (EVA) and Root Mean Squared Error (RMSE), as in [177]. Since there is no ground-truth for variances, we quantify their accuracy with negative log-likelihood (NLL) $1/2 \log(\sigma_{tot}) + 1/2\sigma_{tot}(y_{gt} - y_{pred})^2$ [80, 141, 87].

We compare our approach against state-of-the-art methods for uncertainty estimation. For a fair comparison, all methods share the same network architecture. For all the methods using sampling, we keep the number of samples fixed to $T = 20$, as it allows real-time performance (see Sec D.4.2). Our approach additionally assumes an input noise variance $\mathbf{v} = 2$ grayscale levels, which is typical for the type of camera used in the Udacity dataset.

Table D.1 summarizes the results of this experiment. Unsurprisingly, the method of Kendall et al. [141], trained to minimize the NLL loss, can predict good variances, but loses prediction quality due to the change of training loss and architecture. The approach of Gast et al. [87], trained under the same NLL objective, performs analogously in terms of RMSE and EVA. However, it performs worse in term of NLL, since this baseline only accounts for data uncertainty. In contrast to the previous baselines, the method of Gal et al. [80] predicts more precise steering angles due to ensembling, but generates poorer uncertainty estimates. With our framework, it is not necessary to make compromises: we can both make accurate predictions and have high quality uncertainty estimates, without changing or re-training the network.

Fig. D.3 shows some qualitative results of our approach. As expected, our approach assigns very low variance to well-illuminated images with $y_{gt} \approx 0$. These are indeed the most frequent samples in the training dataset, and contain limited image noise. In contrast, our method predicts high uncertainties for images with large light gradients. This is expected, since those samples can be ambiguous and have a smaller signal to noise ratio. For more qualitative experiments, we refer

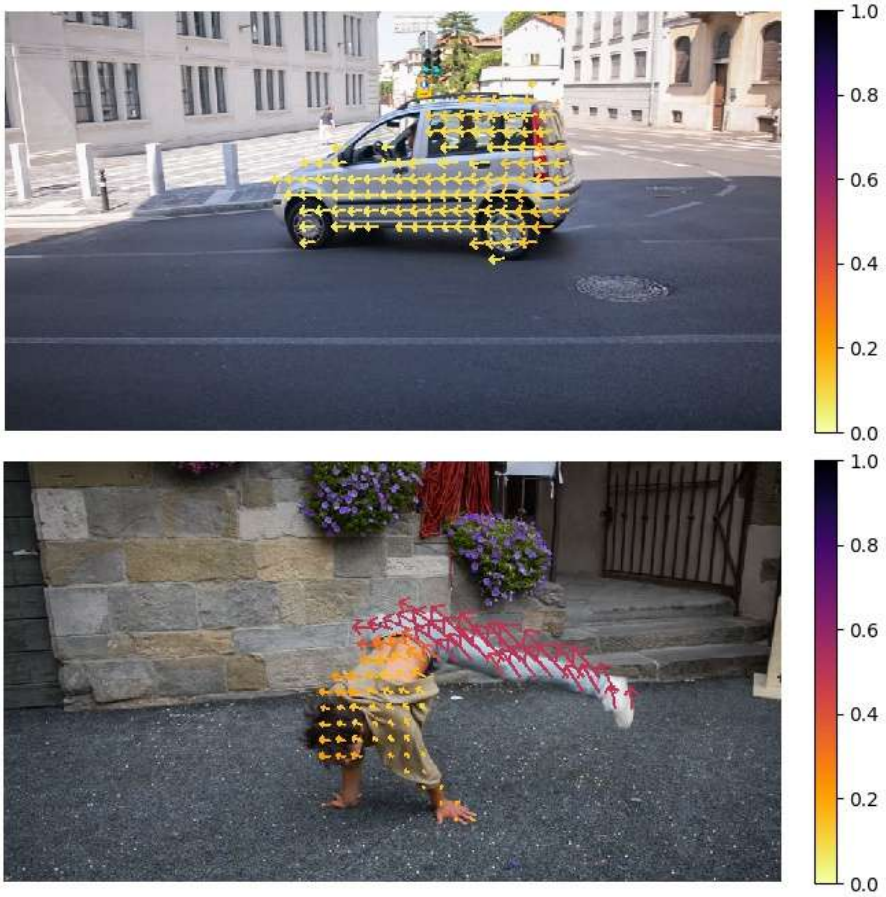


Figure D.4 – Qualitative evaluation on the task of object future motion prediction. Future motion predictions are indicated by arrows, while uncertainties are color-coded. For objects whose motion is easily predictable, e.g car on the top, our framework produces good predictions with low uncertainties. In contrast, for object whose motion is not easily predictable, e.g. the dance at the bottom, predictions are associated with higher variance.

the reader to the supplementary video.

Object Future Motion Prediction: In this section, we train a neural network to predict the motion of an object in the future. Endowing robots with such an ability is important for several navigation tasks, e.g. path planning and obstacle avoidance. More specifically, the task is equivalent to predict the position of an object at time $t + \Delta t$, assuming to have the video of the object moving from $t = 0, \ldots, t$. For the sake of simplicity, we predict motion only in the image plane, or, equivalently, the future 2D optical flow of the object.

In order to predict future flows we use the Flownet2S architecture [124], as it represents a good trade-off between performance and computational cost. The input to the network consists of a concatenation of the current and past frames I_t, I_{t-1}, and the object mask M_t. Its output is instead the optical flow *on the object* between the current and the (unobserved) future frame I_t, I_{t+1}. Ground-truth for this task is generated by the Flownet2 architecture [124], which is

Method	Re-train	EPE	KL	NLL
Gal et al. [80]	Yes	5.99	56.7	6.96
Gast et al. [87]	Yes	6.12	50.1	5.74
Kendall et al. [141]	Yes	6.79	52.5	5.28
Ours	No	**5.91**	**45.1**	**4.07**

Table D.2 – Benchmark comparison against state-of-the-art on the task of object future motion prediction.

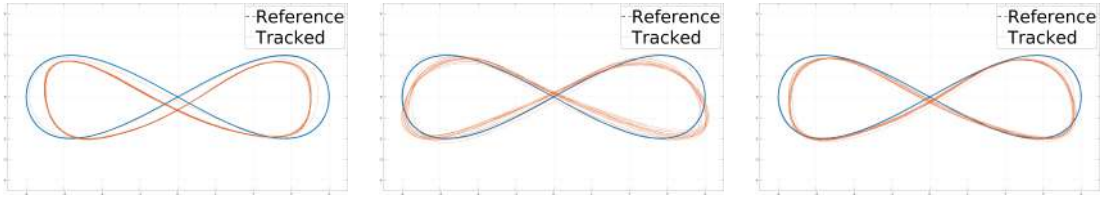


Figure D.5 – Qualitative comparison between different control models in a lemniscate trajectory. Due to drag effects, the nominal model (left) cannot accurately track the reference trajectory. Linear model compensation [61] (middle) decreases tracking errors, but still provides suboptimal results. Our approach (right) providing non-linear compensations to the model only if the prediction uncertainty is within a confidence bound, achieves the best tracking performance.

instead provided with the current and future frames I_t, I_{t+1}.

We perform experiments on the Davis 2016 dataset [223], which has high quality videos with moving objects, as well as pixel-level object masks annotations. Also for this experiment, it is assumed an input noise variance $\mathbf{v}^{(0)}$ of 2 pixels, which is compatible with the type of camera used to collect the dataset.

We again compare our framework for uncertainty estimation to state-of-the-art approaches [80, 87, 141]. To quantitatively evaluate the optical flow predictions, we use the standard end-point error (EPE) metric [124]. This metric is, however, 'local', since it does not evaluate the motion prediction *as a whole* but just as average over pixels. In order to better understand if our approach can predict the motion of the entire object correctly, we fit a Gaussian to both the sets of predicted and ground-truth flows. The KL distance between these two distributions represents our second metric. Finally, we use the standard negative log-likelihood metric (NLL) to evaluate the uncertainty estimations.

Table D.2 summarizes the results of this experiment. Our method outperforms all baselines on every metric. Interestingly, even though our network has not been specifically trained to predict variances as in Kendall et al. [141], it estimates uncertainty 23% better than the runner-up method. At the same time, being the network specifically trained for the task, it makes accurate predictions, outperforming the approach from Gal et al. [80] by 2% in terms of RMSE and 20% on the KL metric.

Qualitative results in Fig. D.4 show that our framework captures an intuitive behaviour of the

prediction network. Whenever the motion of the object is highly predictable, e.g. a car driving on the road, future optical flow vectors are accurately estimated and a low uncertainty is assigned to them. In contrast, if the object moves unpredictably, e.g. a dancer, the network is more uncertain about its predictions, particularly for the parts of the person which quickly change velocity.

Object Recognition: In this section, we investigate the performance of our framework on a classic computer vision task: object recognition. In order to do that, we evaluate our framework on the CIFAR-10 Dataset. We use two metrics to evaluate the performance of our approach: the average classification accuracy, and the average of per-class negative log-likelihood. Results of this evaluation are reported in Table D.3. Similarly to previous tasks, variance estimation in object recognition benefits from considering both model and data uncertainty. The aforementioned result table does not include the baseline of Kendall et al. [141], since its training procedure is specifically designed for regression problems, and it failed to converge in our classification experiments.

Closed-Loop Control of a Quadrotor: In this last experiment, we demonstrate that our framework can be used to fully integrate a deep learning algorithm into a robotics system. In order to do so, we consider the task of real-time, closed-loop control of a simulated quadrotor. For this task, we deploy a multi-layer perceptron (MLP) to learn compensation terms of a forward-dynamics quadrotor model. These compensations generally capture model inaccuracies, due to, e.g., rotor or fuselage drag [61].

We define the common model of a quadrotor, e.g. the one of Mellinger et al. [190], as the *nominal* model. As proposed by previous work [61], we add to the linear and angular acceleration of the nominal model $\dot{p}, \dot{\omega}$ two compensations e_{lin}, e_{ang}. However, while previous work [61] predicts e_{lin}, e_{ang} as a linear function of the platform linear and angular speed v, ω, we propose to predict them as a function of the entire system state $s = (v, \dot{v}, \omega, \dot{\omega})$ via an MLP. Except for the modification of the function approximator (MLP instead of linear), we keep the training and testing methodology of Faessler et al. [61] unchanged.

We collect annotated training data to predict e_{lin}, e_{ang} in simulation [77]. Similarly to [61], we use a set of circular and lemniscate trajectories at different speeds to generate this data. The annotated data is then used to train our MLP, which takes as input s and outputs a 6 dimensional vector e_{lin}, e_{ang}.

We use our framework to compute the MLP's prediction uncertainty. At each time-step, if the uncertainty is larger than a user-defined threshold, the compensations will not be applied. This

Method	Re-train	Accuracy	NLL
Gal et al. [80]	Yes	93.2	4.79
Gast et al. [87]	Yes	93.7	15.2
Ours	No	**94.0**	**2.65**

Table D.3 – Benchmark comparison against state-of-the-art on the task of object recognition.

stops the network to compensate when uncertain, avoiding platform instabilities. Specifically, we set this threshold as five times the mean prediction uncertainty in an held-out testing set.

We perform closed-loop experiments on two types of trajectories: a horizontal circular with 4mt radius and an horizontal lemniscate (see Fig. D.5). Both maneuvers, not observed at training time, were performed with a maximum speed of $7^m/_s$. Following previous work [61], we use the RMSE metric between the reference and actual trajectory for quantitative analysis. The results of this evaluation are presented in Table D.4. Obviously, the high maneuvers' speed introduces significant drag effects, limiting the accuracy of the nominal model. Adding a linear compensation model, as in Faessler et al. [61], improves performance on the simple circular maneuver, but fails to generalize to the more involved lemniscate trajectory. Substituting the linear with a non-linear compensation model (MLP) improves generalization and boosts performance in the latter maneuver. However, applying the compensation only if the network is certain about its predictions (MLP with σ_{tot}) additionally increases tracking performance by 2% and 8% on the circular and lemniscate trajectories, respectively. Indeed, these maneuvers, unobserved at training time, contain states for which compensation is highly uncertain. Finally, Fig. D.5 shows a qualitative comparison on the tracking performance of the different methods. Thanks to the non-linearities and the uncertainty estimation, our approach appears to be closer to the reference trajectory, hence minimizing tracking errors.

Method	Circular	Lemniscate
Nominal model	0.271	0.299
Faessler et al. [61]	0.086	0.298
MLP (Ours)	0.086	0.255
MLP with σ_{tot} (Ours)	**0.084**	**0.234**

Table D.4 – Quantitative comparison on trajectory tracking, closed-loop experiments.

D.4.2 Practical Considerations

Run-time Analysis: We perform a run-time analysis of our framework to study the trade-off between inference time and estimation accuracy. Similar to all methods based on Monte-Carlo sampling, our approach requires multiple forward passes of each image to estimate uncertainty. The larger the number of samples, the better the estimates [80]. As it can be observed in Fig. D.6, the quality of variance estimation, measured in terms of NLL, plateaus for $T \geq N$ samples. In our experiments, we selected $T = 20$ as it allows processing at $\approx 10Hz$, which is acceptable for closed-loop control [217, 139].

Feed-forward vs Recurrent Models: Since our derivations are agnostic to the architecture, the proposed framework can be applied to both feed-forward and recurrent models. However, while recurrent models can improve performance on sequential tasks, their computational cost for extracting uncertainty is significantly higher: for each Monte-Carlo sample, the entire temporal sequence needs to be re-processed.

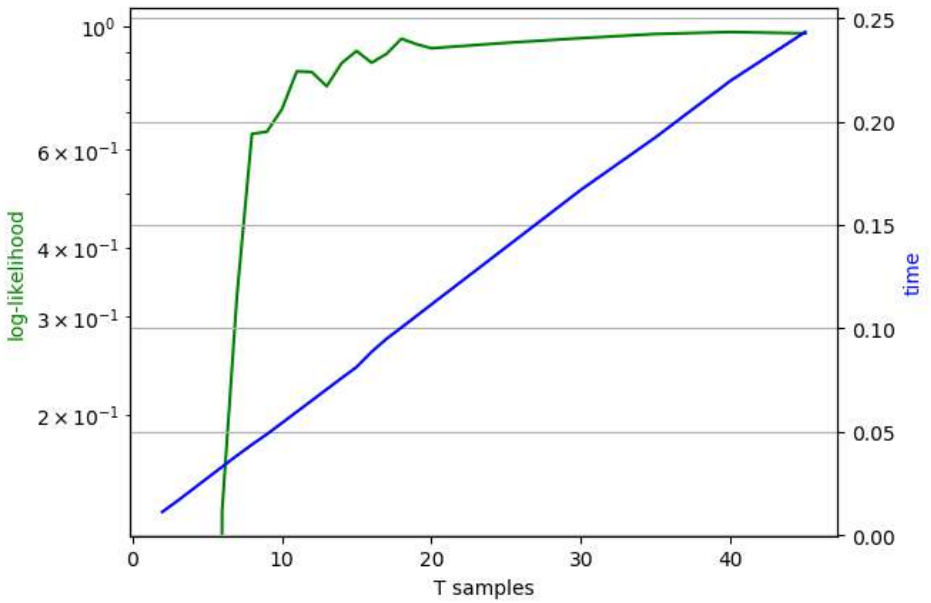


Figure D.6 – As typical for sampling based approaches, a higher number of samples improves uncertainty estimates. In order to obtain real-time estimates, it is necessary to trade-off performance for speed. For example, in the task of end-to-end steering angle prediction, $T = 20$ samples is enough to have good uncertainty estimates and $\approx$ 10Hz inference rate.

D.5 Conclusion

In this work, we present a general framework for uncertainty estimation of neural network predictions. Our framework is general in the sense that it is agnostic to the architecture, the learning procedure, and the training task. Inspired by Bayesian inference, we mathematically show that our approach tightly couples the sources of prediction uncertainty. To demonstrate the flexibility of our approach, we test it on several control and vision tasks. On each task we outperform state-of-the-art methods for uncertainty estimation, without compromising prediction accuracy.

Similarly to all sampling-based methods [80, 141], the main limitation of our approach is that, in order to generate σ_{tot}, we need several network forward passes for each input. This is particularly problematic for recurrent models, which need to unroll the entire temporal sequence for each sample. Although we show that this does not hinder real time performance (see Fig. D.6), it still represents the main bottleneck of our framework. We believe that finding alternative solutions to compute model uncertainty, using, e.g., information theory [2], is a very interesting venue for future work.

D.6 Supplementary

D.6.1 Proof of Lemma III.2

Consider the probabilistic model of the ADF network and the probabilistic distribution over the input:

$$\begin{aligned} p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) &= p(\mathbf{y}|\mathbf{z}, \boldsymbol{\omega}) p(\mathbf{z}|\mathbf{x}) \\ p(\mathbf{z}|\mathbf{x}) &\sim \mathrm{N}\left(\mathbf{z}; \mathbf{x}, \mathbf{v}_t^{(0)}\right) \end{aligned} \tag{D.17}$$

Let's now place a posterior distribution $p\left(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}\right)$ over network weights given the training data $\mathbf{D} = \{\mathbf{X}, \mathbf{Y}\}$. Consequently, the full posterior distribution of the Bayesian ADF network can be parametrized as

$$\begin{aligned} p\left(\mathbf{y}, \mathbf{z}|\mathbf{x}, \mathbf{X}, \mathbf{Y}\right) &= \left(\int p(\mathbf{y}|\mathbf{z}, \boldsymbol{\omega}) \cdot p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) d\boldsymbol{\omega} \right) \cdot p(\mathbf{z}|\mathbf{x}) \\ &= \int p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) \cdot p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) d\boldsymbol{\omega} \end{aligned} \tag{D.18}$$

where $p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) = p(\mathbf{y}|\mathbf{z}, \boldsymbol{\omega}) \cdot p(\mathbf{z}|\mathbf{x}) \sim \mathrm{N}(\widehat{\mathbf{y}}_{\omega}, \mathbf{v}_t^{(l)} I_D)$ for each model weights realization $\boldsymbol{\omega}$. Also, we approximate the intractable posterior over network weights as

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) \approx q(\boldsymbol{\omega}) = \mathrm{Bern}(\mathbf{z}_1) \cdots \mathrm{Bern}(\mathbf{z}_L) \tag{D.19}$$

where $\mathrm{Bern}(\mathbf{z}_i)$ is a Bernoullian distribution over the activation of the i-th layer. Thus,

$$p\left(\mathbf{y}, \mathbf{z}|\mathbf{x}, \mathbf{X}, \mathbf{Y}\right) \approx \int p(\mathbf{y}, \mathbf{z}|\mathbf{x}, \boldsymbol{\omega}) \cdot q(\boldsymbol{\omega}) d\boldsymbol{\omega} = q(\mathbf{y}, \mathbf{z}|\mathbf{x}) \tag{D.20}$$

We will now prove that our framework actually recovers the total variance by plugging multiple stochastic forward passes with MC dropout in Lemma III.2.

Proof.

$$
\begin{aligned}
&\mathbb{E}_{q(\mathbf{y},\mathbf{z}|\mathbf{x})}\left(\mathbf{y}\mathbf{y}^T\right) \\
&\overset{(1)}{=} \int \left(\int \mathbf{y}\mathbf{y}^T \cdot p(\mathbf{y},\mathbf{z}|\mathbf{x},\boldsymbol{\omega})\mathrm{d}\mathbf{y} \right) q(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \\
&\overset{(2)}{=} \int \Bigg(\mathrm{Cov}_{p(\mathbf{y},\mathbf{z}|\mathbf{x},\boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}}) \\
&\qquad + \mathbb{E}_{p(\mathbf{y},\mathbf{z}|\mathbf{x},\boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}})\mathbb{E}_{p(\mathbf{y},\mathbf{z}|\mathbf{x},\boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}})^T \Bigg) \cdot q(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \\
&\overset{(3)}{=} \int \Big(\mathrm{Cov}_{p(\mathbf{y},\mathbf{z}|\mathbf{x},\boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}}) + \mathbb{E}_{p(\mathbf{y},\mathbf{z}|\mathbf{x},\boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}})\mathbb{E}_{p(\mathbf{y},\mathbf{z}|\mathbf{x},\boldsymbol{\omega})}(\mathbf{y}_{\boldsymbol{\omega}})^T \Big) \\
&\qquad \cdot \mathrm{Bern}(\mathbf{z}_1)\cdots\mathrm{Bern}(\mathbf{z}_L)\mathrm{d}\mathbf{z}_1\cdots\mathrm{d}\mathbf{z}_L \\
&\overset{(4)}{=} \int \Big(\mathbf{v}_t^{(l)} I_D + \widehat{\mathbf{y}}(\mathbf{x},\mathbf{z}_1,...,\mathbf{z}_L)\widehat{\mathbf{y}}(\mathbf{x},\mathbf{z}_1,...,\mathbf{z}_L)^T \Big) \\
&\qquad \cdot \mathrm{Bern}(\mathbf{z}_1)\cdots\mathrm{Bern}(\mathbf{z}_L)\mathrm{d}\mathbf{z}_1\cdots\mathrm{d}\mathbf{z}_L \\
&\overset{(5)}{\approx} \frac{1}{T}\sum_{t=1}^{T}\mathbf{v}_t^{(l)} + \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}(\mathbf{x},\widehat{\mathbf{z}}_{1,t},...,\widehat{\mathbf{z}}_{L,t})\widehat{\mathbf{y}}(\mathbf{x},\widehat{\mathbf{z}}_{1,t},...,\widehat{\mathbf{z}}_{L,t})^T
\end{aligned}
$$

(1) follows by the definition of expected value.
(2) follows by the definition of covariance:

$$\mathrm{Cov}(\mathbf{y}) = \mathbb{E}(\mathbf{y}\mathbf{y}^T) - \mathbb{E}(\mathbf{y})\,\mathbb{E}(\mathbf{y})^T$$

(3) follows from Equation D.19.
(4) since $p(\mathbf{y},\mathbf{z}|\mathbf{x},\boldsymbol{\omega}) = \mathrm{N}\Big(\mathbf{y};\widehat{\mathbf{y}}(\mathbf{x},\mathbf{z}_1,...,\mathbf{z}_L),\mathbf{v}_t^{(l)} I_D\Big)$.
(5) approximation by Monte Carlo integration.

Consequently, from the result just obtained and by the definition of variance, it can be easily shown that the total variance can be computed as:

$$
\begin{aligned}
\mathrm{Var}_{q(\mathbf{y}|\mathbf{x})}(\mathbf{y}) &= \mathbb{E}_{q(\mathbf{y},\mathbf{z}|\mathbf{x})}\left(\mathbf{y}\mathbf{y}^T\right) - \mathbb{E}_{q(\mathbf{y},\mathbf{z}|\mathbf{x})}(\mathbf{y})\,\mathbb{E}_{q(\mathbf{y},\mathbf{z}|\mathbf{x})}(\mathbf{y})^T \\
&\approx \frac{1}{T}\sum_{t=1}^{T}\widehat{\mathbf{y}}(\mathbf{x},\widehat{\mathbf{z}}_{1,t},...,\widehat{\mathbf{z}}_{L,t})\widehat{\mathbf{y}}(\mathbf{x},\widehat{\mathbf{z}}_{1,t},...,\widehat{\mathbf{z}}_{L,t})^T \\
&\quad - \mathbb{E}_{q(\mathbf{y},\mathbf{z}|\mathbf{x})}(\mathbf{y})\,\mathbb{E}_{q(\mathbf{y},\mathbf{z}|\mathbf{x})}(\mathbf{y})^T + \frac{1}{T}\sum_{t=1}^{T}\mathbf{v}_t^{(l)} \\
&= \frac{1}{T}\sum_{t=1}^{T}\mathbf{v}_t^{(l)} + (\boldsymbol{\mu}_t^{(l)} - \bar{\boldsymbol{\mu}})^2
\end{aligned}
$$

The total variance of a network output $\mathbf{y}$ for an input sample $\mathbf{x}$ corrupted by noise $\mathbf{v}^{(0)}$ is:

$$\sigma_{tot} = \mathrm{Var}_{p(\mathbf{y}|\mathbf{x})}(\mathbf{y}) = \frac{1}{T}\sum_{t=1}^{T}\mathbf{v}_t^{(l)} + (\boldsymbol{\mu}_t^{(l)} - \bar{\boldsymbol{\mu}})^2 \tag{D.21}$$

which indeed amounts to the sum of the sample variance of T MC samples *(model uncertainty)* and the average of the corresponding *data variances* $\mathbf{v}_t^{(l)}$ returned by the ADF network. □

In conclusion, the final output of our framework is $[\mathbf{y}^*, \boldsymbol{\sigma}_{tot}]$, where $\mathbf{y}^*$ is the mean of the mean predictions $\{\hat{\mathbf{y}}_t\}_{t=1}^{T}$ collected over T stochastic forward passes.

D.7 Training Details

D.7.1 Implementation

Our framework for uncertainty estimation is implemented in Pytorch, and included in the supplementary material. It will be publicly released upon acceptance. For training and testing, we use a desktop computer equipped with an NVIDIA-RTX 2080.

D.7.2 End-to-End Steering Angle Prediction

The NN architecture used for the task *End-to-End Steering Angle Prediction* is a shallow ResNet that takes inspiration from the DroNet architecture by Loquercio et al. [177]. The network was trained on the Udacity dataset [177], containing approximately $70,000$ images captured from a car and distributed over 6 different experiment settings, 5 for training and 1 for testing. A validation set is held out from the data of the first experiment. For every experiment, time-stamped images are stored from 3 cameras (left, central, right) with the associated data: IMU, GPS, gear, brake, throttle, steering angles and speed. For our purpose, only images from the forward-looking camera and their associated steering angles are used. The network is trained for 100 epochs with Adam and an initial learning rate of $1e-3$. The loss used for training is an L2-loss, which is also computed on the validation set at every epoch to select the best model. The total training time on our aforementioned hardware amounts to 6 hours .

D.7.3 Object Future Motion Prediction

For the task *Object Future Motion Prediction* we employ a FlowNet2S architecture to predict the future optical flow. Given the frames at time $t-1$ and t, it predicts the future optical flow between frame t and $t+1$. We use the publicly available weights pre-trained on FlyingChairs

and FlyingThings3D [187] datasets for the task of optical flow regression to initialize our model, which is then specifically trained for our task for 2000 epochs (around 15 hours on our hardware) on the DAVIS 2016 dataset [223]. We used Adam with a learning rate of $1e-3$ and the loss used is the multi-scale L1 metrics. This multi-scale L1 loss is computed only on the moving object, identified by the segmentation mask. As input to the network, we pass the frames at time $t-1$ and t, stacked together with the segmentation mask corresponding to frame t. The optical flow between image t and $t+1$ is used as ground-truth. Since DAVIS dataset does not provide optical flow annotations, we use the state-of-the-art FlowNet2 [124] architecture to collect optical flow annotations for this dataset. Instead, the segmentation masks used as auxiliary input are already provided along with the DAVIS dataset. At test time, to prove the efficacy of the proposed method, the object mask M_t is generated by the state-of-the-art object detector AGS [297]. This indeed provides a good indicator of the network performance 'in the wild', where no ground-truth object mask is available.

D.7.4 Model Error Compensation

The Multi Layer Perceptron (MLP) used for model error compensation in the task *Closed-Loop Control of a Quadrotor* was trained for 100 epochs on a dataset consisting of data collected by a drone. We used Adam with $1e-4$ as learning rate, together with an L1 loss. Training took approximately 2 hours on our hardware. As input we used 24 features, each of them being collected at the same time step. These features are quaternion odometry, linear velocity odometry, angular velocity odometry, and thrusts. The network was trained to learn the linear and angular model error. The data were collected from three different kind of trajectories: circular, lemniscate and random. The circular and lemniscate trajectories were generated with a fixed radius of $4m$ and different velocities. Eventually, the training dataset is composed of almost one million datapoints, consisting of six circular trajectories, six lemniscate trajectories and random trajectories, generated interpolating randomly generated points in the space. Each of these trajectories is generated with a fixed velocity per trajectory ranging from 1 to $8m/s$. One tenth of the data was held out for testing and parameter tuning.

D.8 Sensitivity to Sensor Noise Estimates

One of our framework's input consists of the sensor noise variance $\mathbf{v}^{(0)}$, which is propagated through the CNN to recover data uncertainty on output predictions. The value of $\mathbf{v}^{(0)}$ is usually available from the sensor data sheet, or estimated via system identification. In this section, we study the sensitivity of our framework to the precision of the $\mathbf{v}^{(0)}$ estimates. In order to do so, we perform a controlled experiment for the task of steering angle prediction, where each image is corrupted by known Gaussian noise $\mathrm{N}\left(0;\mathbf{v}^{(0)}\right)$. In this experiment, our framework has an estimate of the noise variance $\hat{\mathbf{v}}^{(0)}$, which does not necessarily coincides with the real $\mathbf{v}^{(0)}$.

Real Sensor Noise $\mathbf{v}^{(0)}$	0.01	0.05	0.1
Gal et al. [80]	0.67	0.35	0.03
Gast et al. [87]	0.74	0.37	0.04
Kendall et al. [141]	**0.99**	0.3	-0.11
Ours ($\hat{\mathbf{v}}^{(0)} = 0.01$)	0.95	**0.41**	**0.12**

Table D.5 – Log-likelihood (LL) score for increasing sensor noise. Higher is better.

Specifically, we keep $\hat{\mathbf{v}}^{(0)} = 0.01$, while we let $\mathbf{v}^{(0)}$ change. The results of this evaluation are reported in Table D.5. Perhaps unsurprisingly, performance peaks when the assumed sensor noise coincides with the real input noise magnitude. However, as the difference between the real and assumed noise variance increases, performance gracefully drops for our approach, indicating the robustness of our method to wrong sensor noise estimates. Interestingly, Table D.5 also shows that our approach deals better than the baselines to increasing magnitudes of the noise. This is due to the coupling of data and model uncertainty enforced by our framework.

E Learning Depth with Very Sparse Supervision

Reprinted, with permission, from:

Antonio Loquercio, A. Dosovitskiy, and D. Scaramuzza. “Learning Depth With Very Sparse Supervision”. In: *IEEE Robotics and Automation Letters* 5 (2020), pp. 5542–5549

Learning Depth with Very Sparse Supervision

Antonio Loquercio, Alexey Dosovitskiy and Davide Scaramuzza

Abstract — Motivated by the astonishing capabilities of natural intelligent agents and inspired by theories from psychology, this paper explores the idea that perception gets coupled to 3D properties of the world via interaction with the environment. Existing works for depth estimation require either massive amounts of annotated training data or some form of hard-coded geometrical constraint. This paper explores a new approach to learning depth perception requiring neither of those. Specifically, we propose a novel global-local network architecture that can be trained with the data observed by a robot exploring an environment: images and extremely sparse depth measurements, down to even a single pixel per image. From a pair of consecutive images, the proposed network outputs a latent representation of the camera's and scene's parameters, and a dense depth map. Experiments on several datasets show that, when ground truth is available even for just one of the image pixels, the proposed network can learn monocular dense depth estimation up to 22.5% more accurately than state-of-the-art approaches. We believe that this work, in addition to its scientific interest, lays the foundations to learn depth with extremely sparse supervision, which can be valuable to all robotic systems acting under severe bandwidth or sensing constraints.

E.1 Introduction

Understanding the three-dimensional structure of the world is crucial for the functioning of robotic systems: for instance, it supports path planning and navigation, as well as motion planning and object manipulation. Animals, including humans, obtain such three-dimensional understanding naturally, without any specialized training. By observing the environment and interacting with it [91], they learn to estimate (possibly non-metric) distances to objects using stereopsis and a variety of monocular cues [118, 94], including motion parallax, perspective, defocus, familiar object sizes. Could a robotic system acquire a metric understanding of its surrounding from a

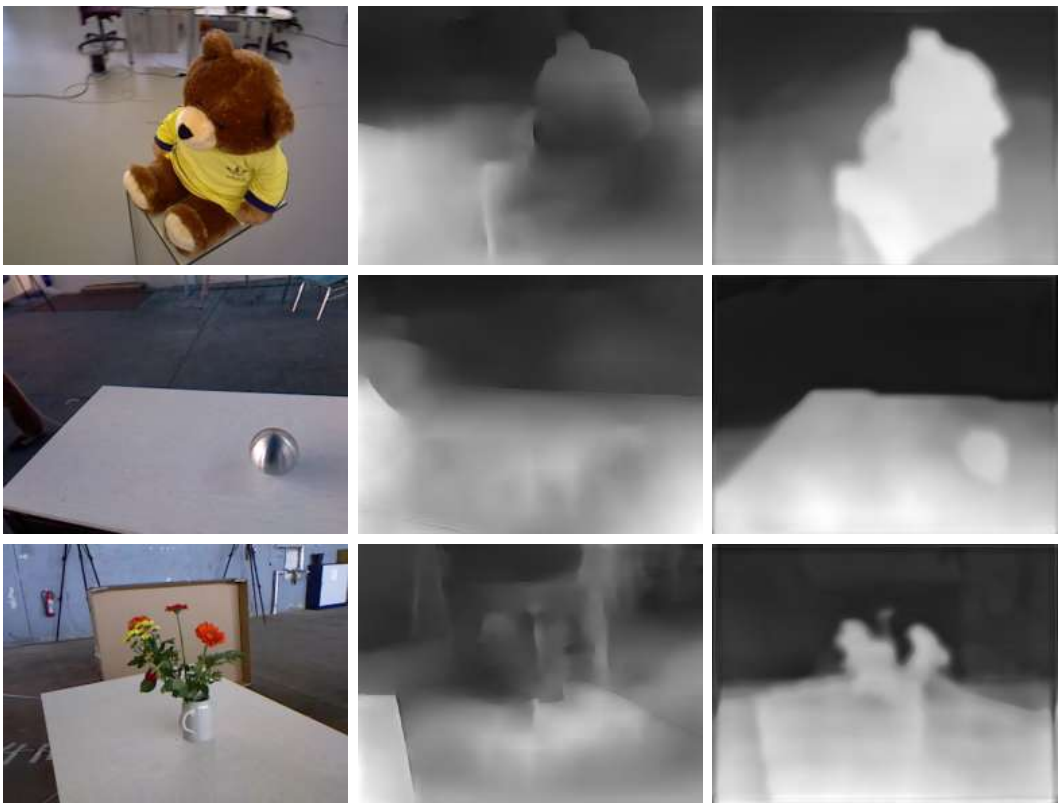


Figure E.1 – We train a depth perception system with what would be available to a robot interacting with the environment: images and very sparse depth measurements. When trained in such sparse regimes (up to a single pixel per image), our approach learns to perceive depth with higher accuracy than state-of-the-art methods.

similar feedback?

Endowing robots with such an ability would be valuable for several applications. Consider for example a swarm of nano aerial vehicles, whose task is to explore a previously unseen environment [217, 188]. Constrained by the size and battery life, each robot can only carry limited sensing, *e.g.* a camera and a 1D distance sensor. Learning to estimate dense metric depth maps from such on-board sensors during operation would allow efficient exploration and obstacle avoidance [42].

Classically, multi-view geometry methods are used to reconstruct the 3D coordinates of points given their corresponding projections in multiple images. These geometric approaches, carefully engineered over decades, demonstrate impressive results in a variety of settings and applications [114]. One downside of this class of approaches is that they are using only some of the depth cues (mainly stereo and motion parallax), but typically do not exploit more subtle monocular cues, such as perspective, defocus or known object size. Unsupervised learning approaches to depth estimation [86, 314, 33] combine geometry with deep learning, with the hope that deep

networks can learn to utilize the cues not used by the classic methods. In these approaches, depth estimators are trained from monocular or stereo video streams, using photometric consistency between different images as a loss function. Unsupervised learning approaches are remarkably successful in many cases, but they are fundamentally based on hard-coded geometry equations, which makes them potentially sensitive to the camera model and parameters, as well as difficult to tune.

E.1.1 Contributions

The present work is motivated by the following question: can a three dimensional perception system be trained with the data that a robot would observe interacting with the environment? To make the problem tractable, we make two assumptions. First, motivated by the extensive evidence from psychology and neuroscience on the fundamental importance of motion perception [207, 149], we provide pre-computed optical flow as an input to the depth estimation system. Optical flow estimation can be learned either from synthetic data [124, 273] or from real data in an unsupervised fashion [189]. Second, we assume at training time the availability of images and only very sparse depth ground truth (just few pixels per image), similar to what a robot might collect with a 1-D distance sensor while navigating an environment.

To learn a depth estimator from such assumptions, we design a specialized global-local deep architecture consisting of two modules, global and local. The global module takes as input two images and optical flow between them and outputs a compact latent vector of "global parameters". We expect those parameters to encode information about the observer's motion, the camera's intrinsics, and scene's features, *e.g.*, planarity. The local module then generates a compact fully convolutional network, conditioned on the "global parameters", and applies it to the optical flow field to generate the final depth estimate. The global and the local modules are trained jointly end-to-end with the available sparse depth labels and without camera's pose or instrinsics ground truth.

Since the method is inspired by learning via interaction, we evaluate it on several indoor scenarios. We compare against generic deep architectures, unsupervised depth estimation approaches, and classic geometry-based methods. Standard convolutional networks show good results when trained with dense depth ground truth, but their performance degrades dramatically in the very sparse data regime. Unsupervised learning-methods and classic triangulation approaches are generally strong, but their performance in the challenging monocular two-frame indoor scenario suffers from suboptimal correspondence estimation due to homogeneous surfaces and occlusions. The proposed approach outperforms all these baselines, thanks to its ability to effectively train with very sparse depth labels and its robustness to imperfections in optical flow estimates.

E.2 Related Work

The problem of recovering the three-dimensional structure of a scene from its two-dimensional projections has been long studied in computer vision [172, 106, 105, 285]. Classic methods are based on multi-view projective geometry [104]. The standard approach is to first find correspondences between images and then use these together with geometric constraints to estimate the camera motion between the images (for instance, with the eight-point algorithm [105]) and the 3D coordinates of the points (e.g., via triangulation [106]). Numerous advanced variations of this basic pipeline have been proposed [113, 211, 206, 253], improving or modifying various its elements. However, key characteristics of these classic methods are that they crucially rely on projective geometry, require laborious hand-engineering, and are not able to exploit non-motion-related depth cues.

To make optimal use of all depth cues, machine learning methods can either be integrated into the classic pipeline, or replace it altogether. The challenge for supervised learning methods is the collection of training data: obtaining ground truth camera poses and geometry for large realistic scenes can be extremely challenging. An alternative is to train on simulated data, but then generalization to diverse real-world scenes can become an issue. Therefore, while supervised learning methods have demonstrated impressive results [57, 229, 289, 313], it is desirable to develop algorithms that function in the absence of large annotated datasets.

Unsupervised (or self-supervised) learning provides an attractive alternative to the label-hungry supervised learning. The dominant approach is inspired by classic 3D reconstruction techniques and makes use of projective geometry and photometric consistency across frames. Existing works use various depth representations for this task: voxel grids [287], point clouds [165], triangular meshes [136] or depth maps [86, 314, 93, 182]. In this work we focus on the depth map representation. Among the methods for learning depth maps, some operate in the stereo setup (given a dataset of images recorded by a stereo pair of cameras) [86, 93], while others address the more challenging monocular setup, where the training data consists of monocular videos with arbitrary camera motions between the frames [314, 182]. Reprojection-based approaches can often yield good results in driving scenarios, but they crucially rely on geometric equations and precisely known camera parameters (one notable exception being the recent work in [95], which learns the camera parameters automatically) and enough textured views. In contrast, we do not require knowing the camera parameters in advance and are robust in low-textured indoor scenarios.

Several works, similar to ours, aim to learn 3D representations without explicitly applying geometric equations [277, 232, 60]. A scene, represented by one or several images, is encoded by a deep network into a latent vector, from which, given a target camera pose, a decoder network can generate new views of the scene. A downside of this technique is that the 3D representation is implicit and therefore cannot be directly used for downstream tasks such as navigation or motion planning. Moreover, at training time it requires knowing camera pose associated with each image. Our method, in contrast, does not require camera poses, and grounds its predictions

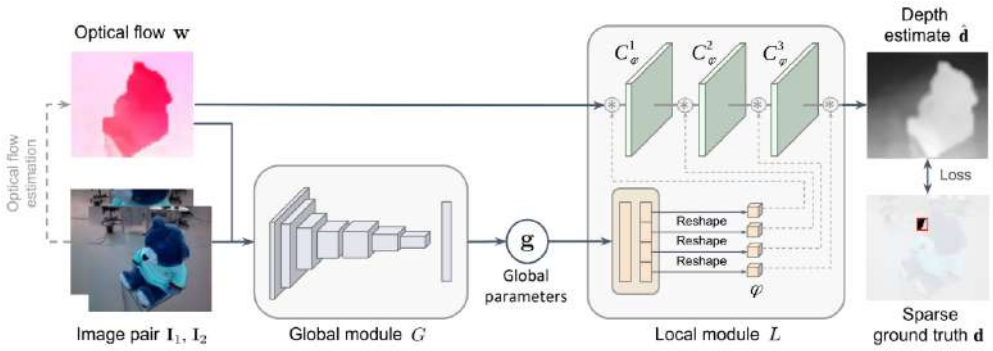


Figure E.2 – Global-local model architecture. An image pair and an estimated flow field are first fed through the global module that estimates the "global parameters" vector $\mathbf{g}$, representing the camera motion. From these global parameters, the local module generates three convolutional filter banks and applies them to the optical flow field. The output of the local module is then processed by a convolution to generate the final depth estimate.

in the physical world via very sparse depth supervision. This allows us to learn an explicit 3D representation in the form of depth maps.

E.3 Methodology

E.3.1 Model architecture

Given two monocular RGB images $\mathbf{I}_1$, $\mathbf{I}_2$, with unknown camera parameters and relative pose, as well as the optical flow $\mathbf{w}$ between them, we aim to estimate a dense depth map corresponding to the first image. We assume to have an artificial agent equipped with a range sensor, which navigates through an indoor environment. By doing so, it collects a training dataset of image pairs, with depth ground truth $\mathbf{d}$ available only for extremely few pixels. Using this sparsely annotated dataset, we train a deep network $F_\theta(\mathbf{I}_1, \mathbf{I}_2, \mathbf{w})$, with parameters $\boldsymbol{\theta}$, that predicts a dense depth map $\hat{\mathbf{d}}$ over the whole image plane. We now describe the network architecture in detail.

An overview of the global-local network architecture is provided in Figure E.2. The system operates on an image pair $\mathbf{I}_1$, $\mathbf{I}_2$ and the optical flow (dense point correspondences) $\mathbf{w}$ between them. In this work, we estimate the flow field with an off-the-shelf optical flow estimation algorithm, which is neither trained nor tuned on our data.

The rest of the model is composed of two modules: a global module G that processes the whole image and outputs a compact vector of "global parameters" and a local module L that applies a compact fully convolutional network, conditioned on the global parameters, to the optical flow field. This design is motivated both by classic 3D reconstruction methods and by machine learning considerations. Establishing an analogy with classic pipelines, the global module corresponds to the relative camera pose estimation, while the local module corresponds to triangulation – estimation of depth given the image correspondences and the camera motion. These connections

	Abs-Inv	Abs-Rel	S-RMSE
Formula	$\frac{1}{\lvert\Omega\rvert}\sum_\Omega \left\lvert \frac{1}{d} - \frac{1}{\hat{d}} \right\rvert$	$\frac{1}{\lvert\Omega\rvert}\sum_\Omega \frac{\lvert d-\hat{d}\rvert}{d}$	$\sqrt{\frac{1}{\lvert\Omega\rvert}\sum_\Omega E_{\log}^2 - \left(\frac{1}{\lvert\Omega\rvert}\sum_\Omega E_{\log}\right)^2}$

Table E.1 – Metrics for quantitative evaluation of depth accuracy. d is the ground truth depth, $\hat{d}$ is the prediction. For convenience we denote $E_{\log} \doteq \log(d/\hat{d}) = \log d - \log \hat{d}$. For all metrics, lower is better.

are described in more detail in the supplement. From the learning point of view, we aim to train a generalizable network with few labels, and therefore need to avoid overfitting. The local module is very compact and operates on a transferable representation – optical flow. The global network is bigger and takes raw images as input, but it communicates with the rest of the model only via the low-dimensional bottleneck of global parameters, which prevents potential overfitting.

The "global module" G is implemented by a convolutional encoder with global average pooling at the end. The network outputs a low-dimensional vector of "global parameters" $\mathbf{g} = G(\mathbf{I}_1, \mathbf{I}_2, \mathbf{w})$. The idea is that the vector represents the motion of the observer, although no explicit supervision is provided to enforce this behavior. While the optical flow alone is in principle sufficient for ego-motion estimation, we also feed the raw image pair to the network to supply it with additional cues.

The "local module" L takes as input the generated global parameters $\mathbf{g}$, as well as the optical flow field. First, the global parameter vector is processed by a linear perceptron that outputs several convolutional filters banks, collectively denoted by $\boldsymbol{\varphi} = LP(\mathbf{g})$. Then, these filter banks are stacked into a small fully convolutional network $C_{\boldsymbol{\varphi}}$ that is applied to the optical flow field. We append two channels of x- and y- image coordinates to the input $\mathbf{w}$ of $C_{\boldsymbol{\varphi}}$, as in CoordConv [167]. The output of $C_{\boldsymbol{\varphi}}$ is the final depth prediction $\hat{\mathbf{d}} = C_{\boldsymbol{\varphi}}(\mathbf{w})$.

This design of the local module is motivated by classic geometric methods: for estimating the depth of a point it is sufficient to know its displacement between the two images, its image plane coordinates, and the camera motion. In contrast to this standard formulation of triangulation, we intentionally make the receptive field of the network larger than 1×1 pixel, so that the network has the opportunity to correct for small inaccuracies or outliers in the optical flow input.

E.3.2 Loss function

Similarly to previous work [289, 229], we define the loss on the inverse depth $\hat{\mathbf{z}} \doteq \hat{\mathbf{d}}^{-1}$. This is a common representation in computer vision and robotics [43, 211], which allows to naturally handle points and their uncertainty over a large range of depths. We use the L_1 loss on the inverse depth, averaged over the subset P of the pixels that have associated ground truth inverse depth $\mathbf{z}$:

$$\mathcal{L}_{\text{depth}} = \frac{1}{|P|}\sum_{i\in P} |\hat{z}^i - z^i|. \tag{E.1}$$

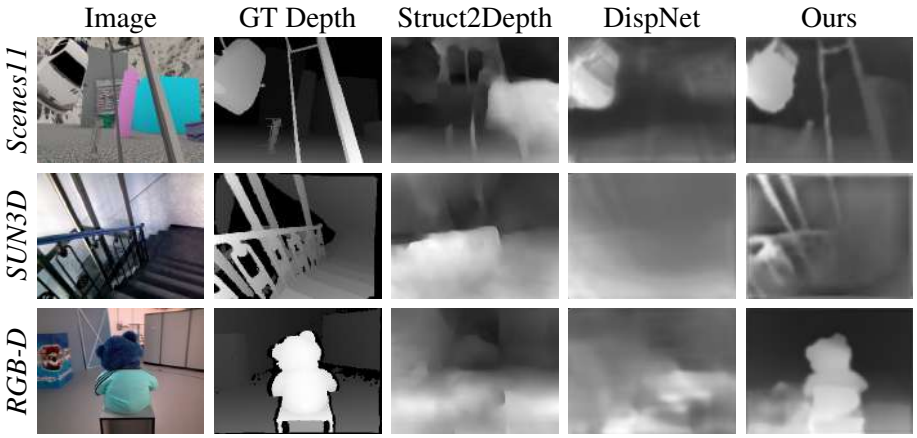


Figure E.3 – Qualitative comparison of the depth maps generated with the baselines and our approach. Overall, Struc2Depth's predictions are generally poor in homogeneous and repetitive regions, while DispNet tends to over-smooth depth maps. In contrast, our method can predict fine details of the scene geometry.

To encourage the local smoothness of the predicted depth maps, we add an L_1 regularization penalty on the gradient $\nabla\hat{\mathbf{z}} = (\partial_x\hat{\mathbf{z}}, \partial_y\hat{\mathbf{z}})$ of the estimated inverse depth. Similarly to classic structure from motion methods and unsupervised depth learning literature [93], we modulate this penalty according to the image gradients $\partial\mathbf{I}_1$, allowing depth discontinuities to be larger at points with large $\partial\mathbf{I}_1$:

$$\mathcal{L}_{\text{smooth}} = \frac{1}{|\Omega|}\sum_{i\in\Omega} |\partial_x\hat{z}^i|\, e^{-|\partial_x I_1^i|} + |\partial_y\hat{z}^i|\, e^{-|\partial_y I_1^i|}, \tag{E.2}$$

with Ω representing the full image plane. The full training loss of our network is a weighted sum of these two terms $\mathcal{L}_{\text{total}} = \lambda_p\mathcal{L}_{\text{depth}} + \lambda_s\mathcal{L}_{\text{smooth}}$.

E.3.3 Model details

In all our experiments the input images have resolution 256×192 pixels. Unless mentioned otherwise, ground truth depth is provided for a single pixel of each image, but we also experiment with denser ground-truth signals. We use a pre-trained PWC-Net [273] for optical flow estimation.

We use the Leaky ReLU non-linearity in all networks. The global module is implemented by a 5-layer convolutional encoder with the number of channels growing from 16 to 256, with stride 2 in the first 4 layers. The last 256-channel hidden layer is followed by a convolution with 6 channels and global average pooling, resulting in the 6-dimensional predicted global parameter vector $\mathbf{g}$. The local module consists of a single linear perceptron which transforms $\mathbf{g}$ linearly to a 3.9K vector. Empirically, we did not find an advantage in utilizing a multi-layer non-linear perceptron in place of the linear operation. The resulting vector is split into three parts, which are reshaped into filter banks with kernel size 3×3 and number of output channels 20, 10, 20, respectively.

Method	Scenes11			SUN3D			RGB-D		
	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE
Eigen [57]	0.045	0.57	0.77	0.072	0.82	0.38	0.046	0.54	0.37
DispNet [187]	0.038	0.51	0.70	0.041	0.49	0.33	0.038	0.45	0.36
FCRN [155]	0.041	0.52	0.74	0.047	0.44	0.30	0.042	0.45	0.35
Small Enc-Dec	0.046	0.66	0.83	0.064	0.73	0.45	0.049	0.58	0.46
Struct2Depth [33]	0.058	0.95	0.81	0.037	0.44	0.27	0.037	0.44	0.48
Struct2Depth [33] + Flow	0.056	0.94	0.79	0.036	0.42	0.27	0.035	0.42	0.45
Ours	**0.031**	**0.43**	**0.61**	**0.035**	**0.37**	**0.25**	**0.033**	**0.37**	**0.33**

Table E.2 – In the sparse training regime, our method can efficiently learn to predict depth from single point supervision, outperforming significantly both standard architectures and unsupervised depth estimation systems. For all error metrics, lower is better.

These filter banks, with Leaky ReLUs in between, constitute the compact fully-convolutional depth estimation network C_{φ}. The 20-channel network output is then processed by a single 3×3 convolutional layer to shrink the channels to 1. The design of this compact fully convolutional network has been inspired by the refinement layer used by previous works on supervised depth estimation [289, 229].

We train the model with the Adam optimizer [148] with an initial learning rate of 10^{-4} for a total of approximately 94K iterations with a mini-batch size of 16. We apply data augmentation during training. Further details are provided in the supplement.

E.4 Experiments

Navigating a physical agent in the real world to collect interaction data is challenging due to problem spanning perception, planning, and control. Therefore, in order to isolate the contribution of our proposed method, we simulate distance observations from a navigation agent, *e.g.* nano-drones [188], by masking out all depth ground truth except for a single one on several depth estimation benchmarks. Specifically, we test the approach on three datasets collected in cluttered indoor environments, either real or simulated: Scenes11, Sun3D, and RGB-D. Scenes11 [289] is a large synthetic dataset with randomly generated scenes composed of objects from ShapeNet [35] against diverse backgrounds composed of simple geometric shapes. SUN3D [300] is a large collection of RGB-D indoor videos collected with a Kinect sensor. RGB-D SLAM [270] is another RGB-D dataset collected with Kinect in indoor spaces.

For all datasets, we use the splits proposed by [289]. As commonly done in two-view depth estimation methods and in structure-from-motion methods [114, 289], we resolve the inherent scale ambiguity by normalizing the depth values such that the norm of the translation vector between the two views is equal to 1. To quantitatively evaluate the generated depth maps, we adopt three standard error metrics summarized in Table E.1. We compare to both standard convolutional neural networks, unsupervised methods, and classic structure from motion methods. Additional quantitative and qualitative comparisons are provided in the supplementary material.

E.4.1 Learning from very sparse ground truth

We compare the proposed global-local architecture to strong generic deep models – the encoder-decoder architecture of Eigen et al. [57], the popular fully convolutional architecture DispNet [187], and the multi-scale encoder-decoder of Laina et al. (FCRN) [155]. For a fair comparison with our method, we provide all the baselines with both the image pair and the optical flow field. Specifically, we generate input samples by concatenating the image sequence and the flow on the last dimension. We additionally tune the models to reach best performance on our task. The details of the tuning process are reported in the appendix. We also compare to a reduced-sized DispNet [187] (Small Enc-Dec), that has a number of parameters similar to our model (including both the global and the local module). Its encoder consists of 4 convolutions with $(16, 32, 54, 128)$ filters, with sizes of $(7, 5, 3, 3)$, and stride 2. Its decoder is composed of 4 up-convolutions with $(128, 64, 32, 16)$ filters of size 3 and stride 1. Encoder and decoder layers are connected through skip connections. Finally, we compare against Struct2Depth [33], current state-of-the-art system for unsupervised depth estimation.

As shown in Table E.2, our approach outperforms all the baselines in the sparse supervision regime. Specifically, we outperform the architecture of Eigen et al. [57] on average by 53%, the architecture of Laina et al. [155] by 22.5%, and the fully convolutional DispNet by 20%. Indeed, due to over-parametrization, these baselines tend to overfit to the training points, failing to generalize to unobserved images and locations.

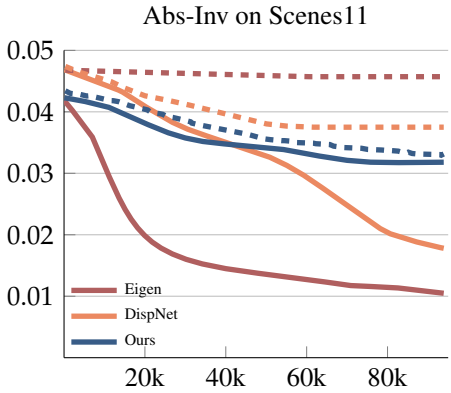


Figure E.4 – For large networks, the loss on training points (solid lines) is significantly higher than the validation loss (dashed lines). In contrast, our global-local architecture learns generalizible representations.

This is empirically demonstrated in Fig. E.4, where we plot the depth loss on training points as a function of the number of iterations. Decreasing the size of the architecture to address overfitting does not however solve the problem: the Small Enc-Dec, with number of parameters similar to our network, achieves poor results, mainly due to its limited capacity.

Our approach also achieves on average 24% better error than the unsupervised depth estimation baseline [33] over all datasets and metrics. Indeed, the considered datasets represent a challenge for geometry-based methods given the presence of large homogeneous regions, occlusions, and

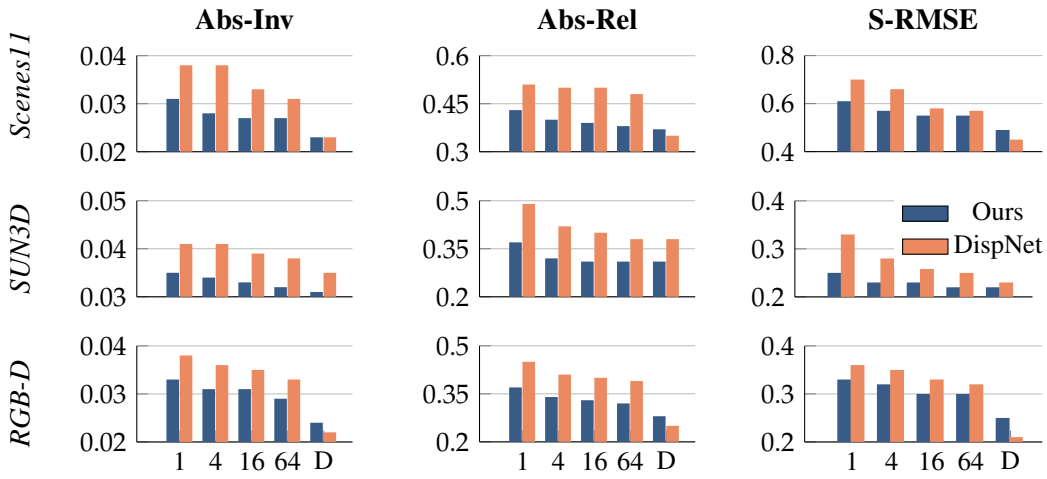


Figure E.5 – Depth estimation errors with increasing number of training pixels per image and dense supervision (D). When supervision gets sparser, our method's performance degrades more gracefully than the baseline.

small baselines between views, which are typical factors encountered in indoor scenes. Noticeably, the performance of Struct2Depth on the SUN3D dataset is relatively good, boosted by the larger baseline between views and the abundance of features. Interestingly, providing optical flow to the unsupervised baseline only increases performance of 3.5% on average. We hypothesize that this is due to the fact that unsupervised methods already estimate correspondences internally, and providing flow as input gives redundant information.

Fig. E.5 analyzes the performance of our and the DispNet architectures (our strongest baseline) as a function of the number of observed ground-truth pixels per image. Unsurprisingly, both methods learn to predict accurate depth maps when dense annotations (D) are available. Decreasing the amount of supervision obviously leads to performance drops. However, for our method the error increases on average by only 5% when going to sparser supervision, compared to 12% for the baseline, which leads to a large advantage over the baseline in the single-pixel supervision regime. This shows that the global-local architecture provides an appropriate inductive bias for learning from extremely sparse depth ground-truth.

E.4.2 Robustness to Dynamically Changing Camera Parameters

In practical applications camera internal parameters, such as focal length, may change through time. Indeed, environmental changes like temperature, humidity and pressure could cause severe variations to their nominal value. Due to these variations, methods based on projective geometry, which are sensitive to the accuracy of calibration parameters, can experience large performance drops. Although the problem could be alleviated by automatic re-calibration, these changes would have to be detected in the first place and would require either collecting multiple views of an object [225, 237] or additional sensing [122].

Method	Scenes11			SUN3D			RGB-D		
	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE
Struct2Depth [33]	0.062	2.19	0.87	0.045	0.52	0.25	0.050	0.54	0.46
DispNet [187]	0.039	0.57	0.70	0.041	0.46	0.27	0.046	0.56	0.38
Ours	**0.034**	**0.51**	**0.61**	**0.034**	**0.43**	**0.24**	**0.036**	**0.40**	**0.33**

Table E.3 – Depth estimation errors with camera intrinsics varying up to 20% of their nominal value between views. Based on projective geometry, unsupervised methods suffer the most from parameter uncertainty.

We empirically study the robustness of our method and the baselines to dynamically changing camera intrinsics. In particular, we randomly change, for each image pair, the horizontal and vertical focal lengths, as well as the center of projection, by up to 20% of their nominal value. The unsupervised depth estimation baseline suffers the most from the uncertainty in the camera intrinsics. Indeed, its estimation error increases on average by 26% with respect to the case in which camera parameters are correctly set. In contrast, as our approach does not explicitly rely on projective geometry, it does not exhibit such sensitivity to the camera parameters. Indeed, it experiences only a small decrease in performance, of approximately 5% with respect to the case where the intrinsics are fixed, since the learning problem becomes more challenging.

E.4.3 Global parameters and the camera motion

According to the intuition behind our model, the global parameters should have information about the observer's ego-motion between the frames, and as such should be related to the actual metric camera motion. Here we study this relation empirically, by training a camera pose predictor on the output of our global module, in supervised fashion. Note that this is done for analysis purposes only, after our full model has been trained: at training time the model has no access to the ground truth camera poses. Specifically, we add a small two-layer MLP with 256 hidden units on top of the global module that is either pre-trained with our method or randomly initialized. We then either train the full network or only the appended small MLP to predict the camera motion in supervised fashion (details of the training process are provided in the supplement).

Results in Table E.4 show that the global parameters indeed contain information about the camera pose. In both training setups pre-trained network substantially outperforms the random initialization: 17% to 64% error reduction across datasets and metrics when only tuning the MLP and up to 11% error reduction when training the full system. Interestingly, our method is also competitive against classic state-of-the-art baselines for motion estimation [289].

E.4.4 Robustness to Optical Flow Outliers

Optical flow estimation plays a central role in our learning procedure. In this section, we study the impact of correspondences' errors to the quality of the predicted depth map. Specifically, we compute the per-pixel Abs-Inv metric as a function of the optical flow's error, normalized

	Scenes11		SUN3D		RGB-D	
Method	rot	trans	rot	trans	rot	trans
Scratch-MLP	1.3	74.4	3.6	55.5	5.3	78.4
Pretrained-MLP	0.9	26.7	2.7	32.5	4.4	51.5
Scratch-Full	**0.7**	10.3	1.8	25.0	**3.2**	30.5
Pretrained-Full	**0.7**	**9.2**	**1.7**	**22.4**	**3.2**	**28.7**
KLT [289]	0.9	14.6	5.9	32.3	12.8	49.6
8-point FF [289]	1.3	19.4	3.7	33.3	4.7	46.1

Table E.4 – Estimation of camera motion based on the global parameters estimated by our model. We initialize the global module either randomly (Scratch) or as trained with our approach (Pretrained). We then append a small MLP and train supervised camera motion prediction by tuning either just the MLP (MLP) or the full network (Full). As a reference, we also report the performance of two classic approaches. We report rotation (rot) and translation (trans) errors in degrees (since the translation vector is normalized to 1, see § E.4). Lower is better.

for the image size. The results of this study are shown in Fig. E.6 for the SUN3D dataset. Results for other datasets are available in the appendix. Our approach (red line) can correct for outliers in the correspondences significantly better than the traditional triangulation approach. Interestingly, the main reason behind this behaviour does not consist in the precision of the camera motion estimation. Indeed, doing triangulation with either ground truth camera motion or the motion estimated from the global parameters (see Sec. E.4.3) result in approximately the same performance. Conversely, we hypothesize that our approach can cope with outliers in correspondences because of the information encoded in the global parameters. Such information not only includes the relative motion between views, but also specifics about the scene (e.g. planarity) and the camera intrinsics. To retrieve this information, the global network uses monocular cues like perspective, focus, or parallax. Conditioned on this scene knowledge and exploiting regularities in the data, the local network can adapt to the observed scene and filter out outliers. Such high-level reasoning is absent in traditional triangulation.

E.4.5 Ablation study

Our architecture is based on several design choices that we now validate through an ablation study. In particular, we ablate the following components: (i) the use of optical flow as an intermediate representation, (ii) the estimation of global variables to generate convolutional filters, (iii) the use of coordinate convolution in the fully convolutional network and (iv) the use of the image pair, in addition to optical flow, for the estimation of global parameters.

The results in Table E.5 show that all components are important and some have larger impact than others. The use of optical flow and coordinate convolution are crucial since they both provide essential cues for depth estimation. However, a basic encoder-decoder architecture (*i.e.* without global variables or coordinate convolutions) underperforms even when provided with optical flow. Unsurprisingly, the least important factor is providing the image pair to the global module,

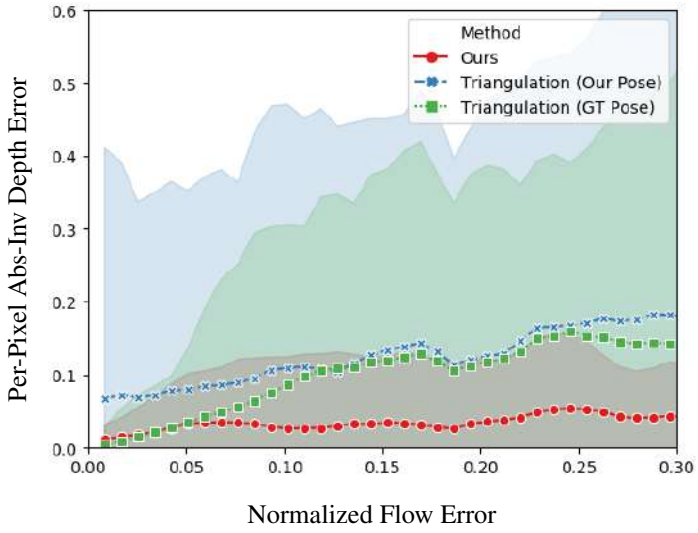


Figure E.6 – Relation between error in correspondences and depth error for the local-global network and classic triangulation, either with perfect pose (GT Pose) or with the pose provided by our fine-tuned global network (see Sec. E.4.3). Standard deviations of the errors are shadowed. Our approach learns to filter out errors in correspondences by exploiting its receptive field larger than one and regularities of those errors in the data.

	Abs-Inv	Abs-Rel	S-RMSE
Full Model	**0.033**	**0.43**	**0.61**
– Image Pair	0.033	0.45	0.62
– CoordConv	0.038	0.52	0.71
– Glob. Mod.	0.041	0.55	0.73
– Flow	0.052	0.73	0.81

Table E.5 – Ablation study on the Scenes11 dataset.

since, when camera parameters are fixed, the optical flow is a sufficient statistics of the observer's ego-motion.

E.5 Discussion

Motivated by the way natural agents learn to predict depth, we propose an approach for training a dense depth estimator from two unconstrained images given only very sparse supervision at training time and without the explicit use of geometry. We show that in cluttered indoor environments our global-local model outperforms state-of-the-art architectures for depth estimation by up to 22.5% in the sparse data regime.

Our methodology comes with some advantages and limitations with respect to previous work. One of the strongest advantage with respect to learning-based methods consists in its ability to learn from extremely sparse data. In addition, our method performs well in the cases where

camera parameters are unknown or corrupted by noise. However, one limitation that our approach shares with supervised and unsupervised methods is the lack of generalization between visually different environments. This limitation can be softened by training in multiple indoor and outdoor environments. With respect to traditional geometry, our method can better cope with outliers in correspondences, given its ability to adapt to the scene characteristics. However, similarly to classic methods, our approach suffers in the case of pure rotation, where correspondences are not informative for depth. This limitation can be overcome by adding memory to the neural network through a recurrent connection.

In the future, we plan to address the aforementioned limitations to increase the network performance. In addition, we believe that a very exciting venue for future work to be the extension of our algorithm to a more strictly interactive procedure on a physical platform.

E.6 Appendix

E.6.1 Connection with Two-View Triangulation

The problem of triangulation consists of computing the 3D coordinates of a point given its (noisy) projections on two or more views and the camera parameters of the views. Hence, it is a geometric problem. Following the usual formalism of homogeneous coordinates [104], the perspective projection of a 3D point M on two cameras with projection matrices P_1, P_2 (comprising the intrinsic and extrinsic parameters of both views) is given by $\lambda_1 m_1 = P_1 M$ and $\lambda_2 m_2 = P_2 M$, where λ_1, λ_2 are the projective scaling factors.

Given P_1, P_2, m_1, m_2, the linear triangulation algorithm [104, Sec.12.2], which tackles the triangulation problem in its most general setting (projective cameras), computes the 3D point M by minimizing the Rayleigh quotient $\|AM\| / \|M\|$, where A is the matrix

$$A(P_1, P_2, m_1, m_2) = \begin{pmatrix} [m_1]_\times P_1 \\ [m_2]_\times P_2 \end{pmatrix} \tag{E.3}$$

and $[u]_\times$ is the cross product matrix (such that $[u]_\times v = u \times v$, for all v).

In case of multiple point correspondences $\{m_1^i \leftrightarrow m_2^i\}$ for $i = 1, \ldots, N$, the camera matrices P_1, P_2 appear in the triangulation equations (E.3) of all of them, and hence, are "global" variables. If the camera matrices are known, then (i) every 3D point M^i can be triangulated independently from the rest, and (ii) the triangulated point is a function of the point correspondences $m_1^i \leftrightarrow m_2^i$.

$$M^i = f(m_1^i, m_2^i; P_1, P_2). \tag{E.4}$$

This intuition inspired the design of our modular architecture: First, a neural network regresses global variables which depend only on the two views, and then those global variables are used by a local module to generate a fully-convolutional net which transforms point correspondences (optical flow) into depth.

E.6.2 Training Process

We train our model from scratch on the Scenes11 dataset for approximately 150K steps using Adam as optimizer with an initial learning rate of $1e-4$. We normalize all losses with the number of points used to compute them. The loss weights for depth and smoothness λ_p, λ_s are 5.0 and 2.0 respectively. To increase generalization, we perform data augmentation at training time by mirroring pairs on the x-axis and rotating them 180 degrees, both 50% probability. For a fair comparison, we trained all baselines with exactly the same strategy and hyper-parameters on a desktop PC equipped with an NVIDIA-GeForce 940MX. On our hardware, the global-

local network inference takes approximately 4 milliseconds, with the global and local network requiring 4 and 1 milliseconds, respectively. Conversely, the prediction of optical flow with PWCNet requires 45 milliseconds per image pair.

For the pose experiments in Sec. 4.4, we trained a 2 hidden layer MLP with 20 nodes and leaky ReLU activation function to predict relative camera motion between frames from the global parameters estimated by our global network. For all datasets and all variations, we trained with an L_1 loss between estimated and real camera poses for 50K steps.

Tuning of Baselines

To fairly study the sparse training setting, we tuned the baseline to reach the best performance on the sparse training task. First, we changed the input layer of each baseline architecture. Exactly as for ours, the baselines' input consists of the concatenation of the image pair and the optical flow on the last channel. Given the very sparse supervision signal, we noticed that the ReLu activation function generated extremely sparse and noisy gradients. Therefore, we modified the original activation function of DispNet [187], FCRN [155] and Eigen [57] from ReLu to LeakyRelu. This change improved the performance of the baselines of up to 50% on average over metrics.

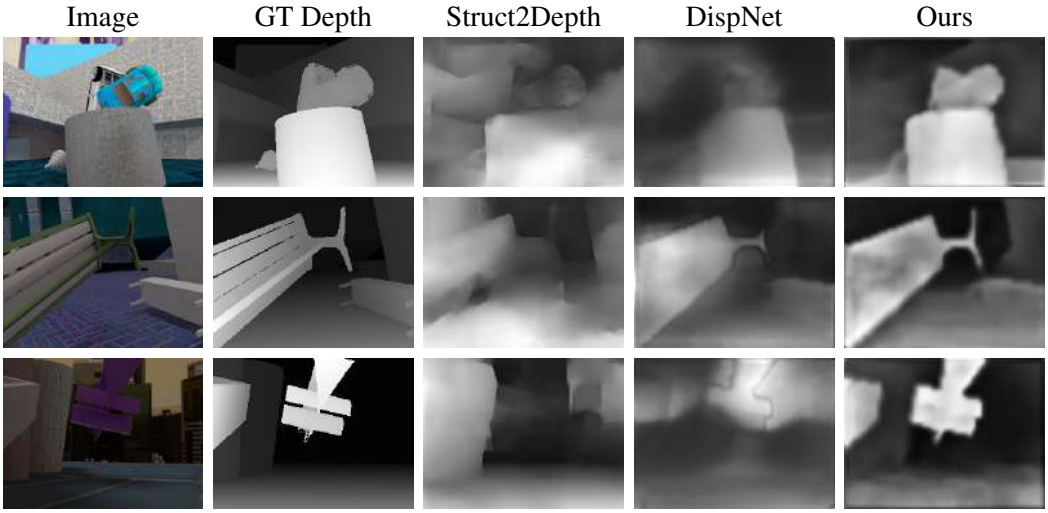


Figure E.7 – Qualitative results on the simulated Scenes11 dataset. Given the availability of noise-free depth maps for training and high quality optical flow, our approach can learn extremely sharp depth maps, comparable to the ones learned by an encoder-decoder with dense supervision.

E.6.3 Comparison with structure from motion methods

Dense Structure from Motion (SfM) methods [211, 113] can recover the depth map of a scene from two or more views using projective geometry [104]. We now compare our global-local

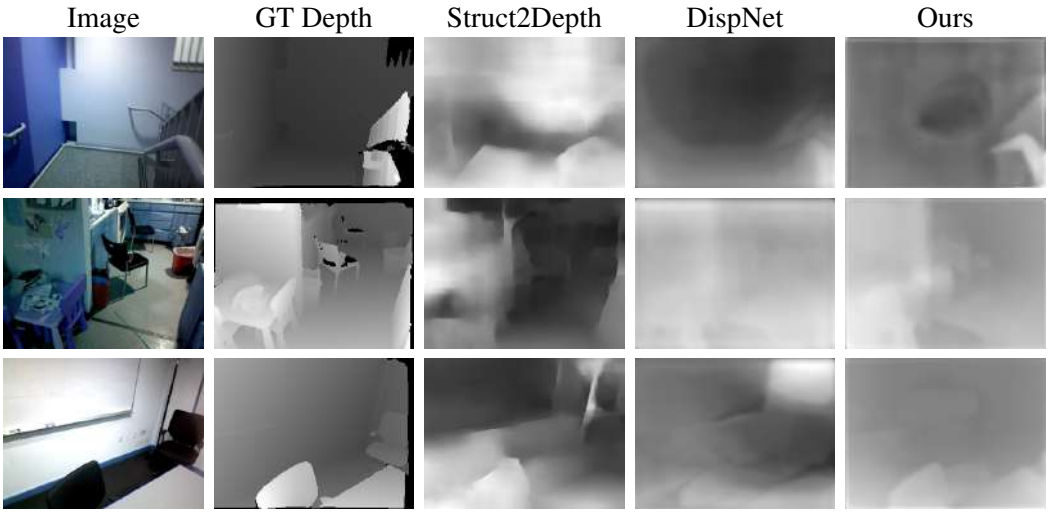


Figure E.8 – Qualitative results on the SUN3D: Given the very noisy optical flow estimated by PWCNet and the presence of noise in the training depth maps, the performance of all methods drops in this dataset. However, our approach is still able to learn smooth depth maps. Interestingly, our model can pick up details which are not present in the ground-truth depth (top-row).

architecture to two SfM baselines proposed by Ummenhofer et al. [289]: one that computes correspondences between images by matching SIFT keypoints (SfM-SIFT) and another that uses optical flow instead [12] (SfM-Flow). Given the correspondences, the essential matrix is estimated with the normalized 8-point algorithm and RANSAC [104], and further refined by minimizing the reprojection error with the *ceres* library. Finally, the depth maps are computed by plane sweep stereo and optimized with the variational approach of Hirshmueller et al. [114].

Results in Table E.6 show that our approach achieves on average 63% better error than SfM-SIFT and 43% better error than SfM-Flow over all datasets and metrics. Indeed, the considered datasets represent a challenge for geometry-based methods given the presence of large homogeneous regions, occlusions, and small baselines between views, which are typical factors encountered in indoor scenes. In addition, geometry-based methods are known to be subject to correspondence errors, which the aforementioned factors generally worsen. Nonetheless, our method remains competitive against the geometry-based techniques, outperforming them on two out of three metrics. Furthermore, as we show in the next section, our method is much more robust to errors in the optical flow estimates than classic triangulation.

E.6.4 Experiments with ground truth

The main intuition driving the design of our architecture consists of the fact that the relative pose between two images control the conversion of correspondences to depth. Since depth can be computed analytically given the correspondences (in non-degenerate cases), can also our model learn this relation when correspondences are perfect or the relative pose between cameras is

Dataset	Scenes11			SUN3D			RGB-D		
	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE
Dataset-Mean	0.069	0.771	0.940	0.081	0.730	0.378	0.062	0.695	0.475
SfM-SIFT [289]	0.051	1.027	0.900	0.029	0.286	0.290	0.050	0.703	0.577
SfM-Flow [289]	0.038	0.776	0.793	**0.029**	**0.297**	0.284	0.045	0.613	0.548
Ours	**0.033**	**0.43**	**0.61**	0.045	0.48	**0.23**	**0.040**	**0.44**	**0.33**

Table E.6 – Comparison to Structure from Motion (SfM) baselines. Our approach outperforms SfM methods on all datasets and metrics, except for two. This shows that learning from sparse supervision can be competitive with classic geometry-based techniques.

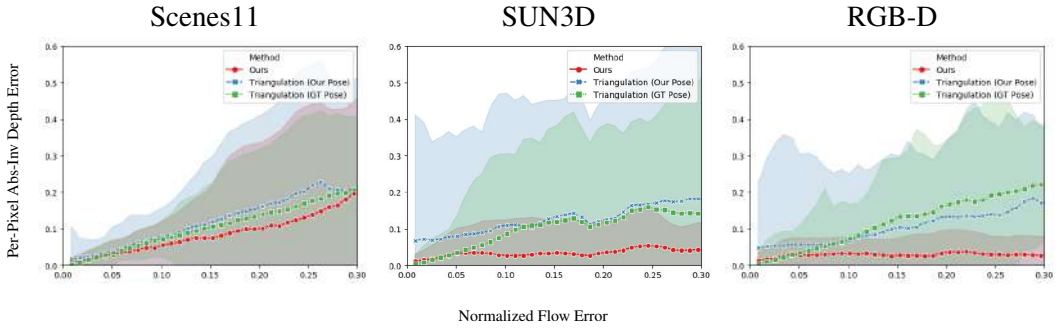


Figure E.9 – Relation between per pixel flow and depth error for our method and triangulation, either with perfect pose (GT Pose) or with the pose provided by our fine-tuned global network (see Sec. E.4.3). Our approach learns to filter out errors in correspondences by exploiting its receptive field larger than one and regularities of those errors in the data.

given? How does it compare to the triangulation equations in term of performance and ability to handle false correspondences?

To answer these questions, we trained our network with either perfect flow or given pose. As baselines, we used both the simple triangulation equation and the SfM-pipeline presented in Sec. 4.3, but provided with ground-truth camera motion. In addition, we also compare our approach to linear triangulation with the pose predicted by our global parameter network after supervised refinement on ground-truth poses (see Sec. 4.4). The results of this analysis are reported in Table E.7.

Performances on Scenes11 show that, when the optical flow generated by PWCNet are very precise, naive triangulation performs very competitively, even outperforming the more complex SfM method. Our network perform comparably to this baseline, showing indeed its ability to learn the mathematical relation between flow and depth. However, triangulation is very sensitive to its inputs: when the precision of the extrinsic parameters or of the correspondences decreases, its performance drastically drops. This can be observed from the results on the SUN3D and RGB-D datasets (Table E.7), where the optical flow generated from PWCNet is significantly worse than the one on Scenes11. In contrast, as we show in Fig. E.9, our network can cope against these inaccuracies, outperforming triangulation in the real datasets. However, our approach is still not competitive with the SfM pipeline with given extrinsic parameters on the SUN3D and

Dataset	Scenes11			SUN3D			RGB-D		
	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE
Triang.-P. Pose	0.061	0.48	0.65	0.65	0.78	0.52	0.50	0.52	0.73
Triang.-GT Pose	0.020	0.25	0.48	0.14	0.33	0.44	0.091	0.36	0.49
SfM-GT Pose [289]	0.023	0.35	0.62	**0.020**	**0.22**	0.24	**0.026**	**0.34**	0.398
Ours-GT Flow	**0.015**	**0.24**	**0.46**	0.026	0.26	**0.20**	0.040	0.35	**0.32**
Ours-GT Pose	0.021	0.28	0.54	0.038	0.37	0.32	0.043	0.44	0.38
Ours	0.033	0.43	0.61	0.045	0.48	0.23	0.040	0.44	0.33

Table E.7 – Comparison of our approach with different input modalities to triangulation with perfect pose, triangulation with pose estimated by finetuning our global net as in Sec. 4.4 (P. Pose), and the SfM pipeline with ground-truth pose. Although slightly outperformed by the SfM baseline with pose information, our approach is significantly better than naive triangulation on the real datasets, where correspondences are generally very noisy, indicating that our approach learns to filter out these correspondence errors. Providing ground-truth relative pose between images or perfect correspondences generally increases the network performance, showing the ability of our model to learn the relationship between these two modalities in ideal conditions.

Dataset	Scenes11			SUN3D			RGB-D		
	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE	Abs-Inv	Abs-Rel	S-RMSE
PWC [273](*)	0.046	0.63	0.81	0.081	0.87	0.37	0.047	0.51	0.42
Ours (finetune PWC)	0.038	0.48	0.70	0.042	0.40	0.25	0.046	0.41	0.35
Ours (no finetune)	0.033	0.43	0.61	0.045	0.48	0.23	0.040	0.44	0.33

Table E.8 – Fine-tuning the parameters of PWCNet with a very sparse depth loss performs worse than fixing them. (*) Indicates that a small convolutional head has been added to the PWCNet architecture.

RGB-D datasets: this is an indicator of the difficulty coming from estimating good global and local parameters when both training flows and depths are noisy, but not of our network ability to learn the relationship between these two modalities. Indeed, when the network is provided with perfect correspondences, performance generally increases.

E.6.5 Fine-tuning Optical Flow

For all our experiments above, we have always assumed a pre-computed optical flow field is provided as input. This optical flow, generated by the off-the-shelf PWCNet architecture [273], was fixed throughout training. In this section, we study the case when also the parameters of PwCNet are fine-tuned during training with the sparse depth loss. Table E.8 shows the results of this evaluation. Interestingly, finetuning the parameters of PWCNet performs worse than fixing them. Such finding can be explained by the fact that the sparse depth loss is not sufficient to train the large number of PWCNet parameters. Indeed, we noticed a decrease in the training error of approximately 10%, indicating an over-fitting to the observed training points. As an additional baseline, we add to PWCNet a small convolutional head to convert flow to depth and train everything with the sparse depth loss. The convolutional head consists of three convolution

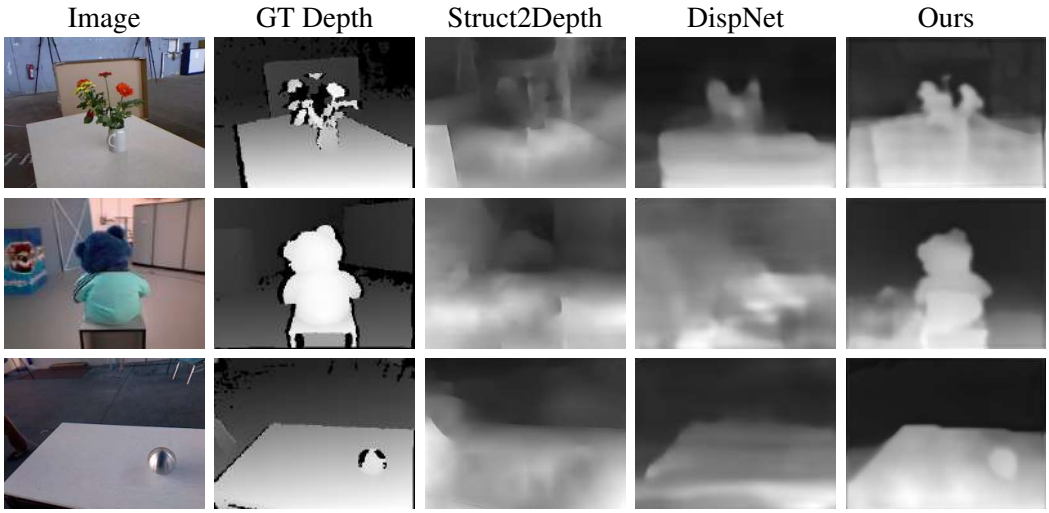


Figure E.10 – Qualitative results on RGB-D. Also this dataset represents a challenge for all methods, given the large baseline between views, noisy correspondences and noisy training depth maps. Nonetheless, our approach is still able to estimate sharp depth maps, sometimes capturing fine details which even an encoder-decoder trained with dense supervision fails to catch (bottom-row).

with (32,16,1) number of filters, stride 1 and filter size 3. The result of this approach, presented in the first row of Table E.8, confirms that the sparse loss does not provide enough feedback to fine-tune correspondences.

E.6.6 Qualitative Results

We show more qualitative results of depth estimation on the test set of our evaluation datasets in Fig. E.7, Fig. E.8 and Fig. E.10. Despite being trained with very sparse supervision, our approach learns to predict smooth depth maps with sharp edges, comparable to the ones an encoder-decoder architecture learns with dense supervision. In contrast, an encoder-decoder architecture fails to learn smooth depths when trained with sparse supervision.

Fig. E.11 shows some of the filters produced by the local network to convert optical flow into depth. Since converting flow to depth depends on the relative transformation between the two views, those filters are input-dependent. Generally, filters are different for each image pair. However, when the relative transformation between the input views is similar, filters also tend to be similar (first and second row of Fig. E.11). In contrast, when the relative transformation between views is completely different, filters tend to acquire a dissimilar pattern (first and third row of Fig. E.11).

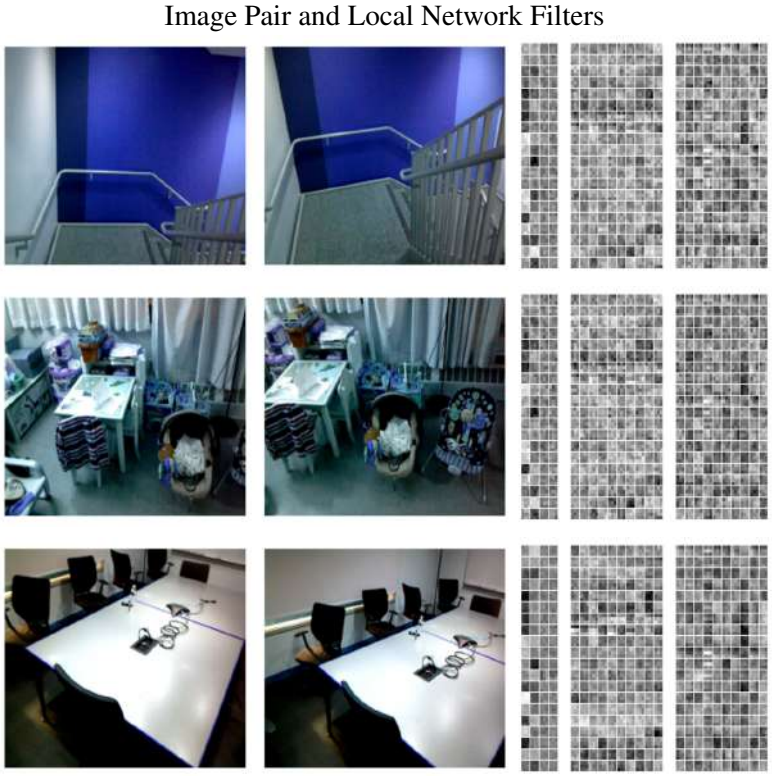


Figure E.11 – Local network filters generated by several image pairs. Generally, filters are different for each image pair. However, when the relative transformation between the two views is similar, filters also tend to be similar (first and second row). In contrast, when the relative transformation is completely different, filters tend to have a dissimilar pattern (first and third row).

F Deep Drone Acrobatics

The version presented here is reprinted, with permission, from:

Elia Kaufmann*, Antonio Loquercio*, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Deep Drone Acrobatics”. In: *RSS: Robotics, Science, and Systems* (2020)

Deep Drone Acrobatics

Elia Kaufmann*, Antonio Loquercio*, Rene Ranftl, Matthias Mueller, Vladlen Koltun

and Davide Scaramuzza

Abstract — Performing acrobatic maneuvers with quadrotors is extremely challenging. Acrobatic flight requires high thrust and extreme angular accelerations that push the platform to its physical limits. Professional drone pilots often measure their level of mastery by flying such maneuvers in competitions. In this paper, we propose to learn a sensorimotor policy that enables an autonomous quadrotor to fly extreme acrobatic maneuvers with only onboard sensing and computation. We train the policy entirely in simulation by leveraging demonstrations from an optimal controller that has access to privileged information. We use appropriate abstractions of the visual input to enable transfer to a real quadrotor. We show that the resulting policy can be directly deployed in the physical world without any fine-tuning on real data. Our methodology has several favorable properties: it does not require a human expert to provide demonstrations, it cannot harm the physical system during training, and it can be used to learn maneuvers that are challenging even for the best human pilots. Our approach enables a physical quadrotor to fly maneuvers such as the Power Loop, the Barrel Roll, and the Matty Flip, during which it incurs accelerations of up to 3g.

Supplementary Material

A video demonstrating acrobatic maneuvers is available at https://youtu.be/2N_wKXQ6MXA. Code can be found at https://github.com/uzh-rpg/deep_drone_acrobatics.

Figure F.1 – A quadrotor performs a Barrel Roll (left), a Power Loop (middle), and a Matty Flip (right). We safely train acrobatic controllers in simulation and deploy them with no fine-tuning (*zero-shot transfer*) on physical quadrotors. The approach uses only onboard sensing and computation. No external motion tracking was used.

F.1 Introduction

Acrobatic flight with quadrotors is extremely challenging. Human drone pilots require many years of practice to safely master maneuvers such as power loops and barrel rolls[1]. Existing autonomous systems that perform agile maneuvers require external sensing and/or external computation [179, 1, 29]. For aerial vehicles that rely only on onboard sensing and computation, the high accelerations that are required for acrobatic maneuvers together with the unforgiving requirements on the control stack raise fundamental questions in both perception and control. Therefore, they provide a natural benchmark to compare the capabilities of autonomous systems against trained human pilots.

Acrobatic maneuvers represent a challenge for the actuators, the sensors, and all physical components of a quadrotor. While hardware limitations can be resolved using expert-level equipment that allows for extreme accelerations, the major limiting factor to enable agile flight is reliable state estimation. Vision-based state estimation systems either provide significantly reduced accuracy or completely fail at high accelerations due to effects such as motion blur, large displacements, and the difficulty of robustly tracking features over long time frames [31]. Additionally, the harsh requirements of fast and precise control at high speeds make it difficult to tune controllers on the real platform, since even tiny mistakes can result in catastrophic outcomes for the platform.

The difficulty of agile autonomous flight led previous work to mostly focus on specific aspects of the problem. One line of research focused on the control problem, assuming near-perfect state estimation from external sensors [179, 1, 121, 29]. While these works showed impressive examples of agile flight, they focused purely on control. The issues of reliable perception and state estimation during agile maneuvers were cleverly circumvented by instrumenting the environment with sensors (such as Vicon and OptiTrack) that provide near-perfect state estimation to the platform at all times. Recent works addressed the control and perception aspects in an integrated way via techniques like perception-guided trajectory optimization [67, 66, 260] or training end-to-end visuomotor agents [308]. However, acrobatic performance of high-acceleration maneuvers

[1] https://www.youtube.com/watch?v=T1vzjPa5260

with only onboard sensing and computation has not yet been achieved.

In this paper, we show for the first time that a vision-based autonomous quadrotor with only onboard sensing and computation is capable of autonomously performing agile maneuvers with accelerations of up to 3g, as shown in Fig. F.1. This contribution is enabled by a novel simulation-to-reality transfer strategy, which is based on abstraction of both visual and inertial measurements. We demonstrate both formally and empirically that the presented abstraction strategy decreases the simulation-to-reality gap with respect to a naive use of sensory inputs. Equipped with this strategy, we train an end-to-end sensimotor controller to fly acrobatic maneuvers *exclusively* in simulation. Learning agile maneuvers entirely in simulation has several advantages: (i) Maneuvers can be simply specified by reference trajectories in simulation and do not require expensive demonstrations by a human pilot, (ii) training is safe and does not pose any physical risk to the quadrotor, and (iii) the approach can scale to a large number of diverse maneuvers, including ones that can only be performed by the very best human pilots.

Our sensorimotor policy is represented by a neural network that combines information from different input modalities to directly regress thrust and body rates. To cope with different output frequencies of the onboard sensors, we design an asynchronous network that operates independently of the sensor frequencies. This network is trained in simulation to imitate demonstrations from an optimal controller that has access to privileged state information.

We apply the presented approach to learning autonomous execution of three acrobatic maneuvers that are challenging even for expert human pilots: the Power Loop, the Barrel Roll, and the Matty Flip. Through controlled experiments in simulation and on a real quadrotor, we show that the presented approach leads to robust and accurate policies that are able to reliably perform the maneuvers with only onboard sensing and computation.

F.2 Related Work

Acrobatic maneuvers comprehensively challenge perception and control stacks. The agility that is required to perform acrobatic maneuvers requires carefully tuned controllers together with accurate state estimation. Compounding the challenge, the large angular rates and high speeds that arise during the execution of a maneuver induce strong motion blur in vision sensors and thus compromise the quality of state estimation.

The complexity of the problem has led early works to only focus on the control aspect while disregarding the question of reliable perception. Lupashin et al. [179] proposed iterative learning of control strategies to enable platforms to perform multiple flips. Mellinger et al. [191] used a similar strategy to autonomously fly quadrotors through a tilted window [191]. By switching between two controller settings, Chen et al. [38] also demonstrated multi-flip maneuvers. Abbeel et al. [1] learned to perform a series of acrobatic maneuvers with autonomous helicopters. Their algorithm leverages expert pilot demonstrations to learn task-specific controllers. While

these works proved the ability of flying machines to perform agile maneuvers, they did not consider the perception problem. Indeed, they all assume that near-perfect state estimation is available during the maneuver, which in practice requires instrumenting the environment with dedicated sensors.

Aggressive flight with only onboard sensing and computation is an open problem. The first attempts in this direction were made by Shen et al. [260], who demonstrated agile vision-based flight. The work was limited to low-acceleration trajectories, therefore only accounting for part of the control and perception problems encountered at high speed. More recently, Loianno et al. [171] and Falanga et al. [66] demonstrated aggressive flight through narrow gaps with only onboard sensing. Even though these maneuvers are agile, they are very short and cannot be repeated without re-initializing the estimation pipeline. Using perception-guided optimization, Falanga et al. [67] and Lee et al. [158] proposed a model-predictive control framework to plan aggressive trajectories while minimizing motion blur. However, such control strategies are too conservative to fly acrobatic maneuvers, which always induce motion blur.

Abolishing the classic division between perception and control, a recent line of work proposes to train visuomotor policies directly from data. Similarly to our approach, Zhang et al. [308] trained a neural network from demonstrations provided by an MPC controller. While the latter has access to the full state of the platform and knowledge of obstacle positions, the network only observes laser range finder readings and inertial measurements. Similarly, Li et al. [162] proposed an imitation learning approach for training visuomotor agents for the task of quadrotor flight. The main limitation of these methods is in their sample complexity: large amounts of demonstrations are required to fly even straight-line trajectories. As a consequence, these methods were only validated in simulation or were constrained to slow hover-to-hover trajectories.

Our approach employs *abstraction* of sensory input [202] to reduce the problem's sample complexity and enable *zero-shot sim-to-real transfer*. While prior work has demonstrated the possibility of controlling real-world quadrotors with zero-shot sim-to-real transfer [176, 246], our approach is the first to learn an end-to-end sensorimotor mapping – from sensor measurements to low-level controls – that can perform high-speed and high-acceleration acrobatic maneuvers on a real physical system.

F.3 Overview

In order to perform acrobatic maneuvers with a quadrotor, we train a sensorimotor controller to predict low-level actions from a history of onboard sensor measurements and a user-defined reference trajectory. An observation $\boldsymbol{o}[k] \in \mathbb{O}$ at time $k \in [0, \dots, T]$ consists of a camera image $\mathcal{I}[k]$ and an inertial measurement $\phi[k]$. Since the camera and IMU typically operate at different frequencies, the visual and inertial observations are updated at different rates. The controller's output is an action $\boldsymbol{u}[k] = [c, \boldsymbol{\omega}^\top]^\top \in \mathbb{U}$ that consists of continuous mass-normalized collective thrust c and bodyrates $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^\top$ that are defined in the quadrotor body frame.

The controller is trained via *privileged learning* [37]. Specifically, the policy is trained on demonstrations that are provided by a privileged expert: an optimal controller that has access to privileged information that is not available to the sensorimotor student, such as the full ground-truth state of the platform $\boldsymbol{s}[k] \in \mathbb{S}$. The privileged expert is based on a classic optimization-based planning and control pipeline that tracks a reference trajectory from the state $\boldsymbol{s}[k]$ using MPC [67].

We collect training demonstrations from the privileged expert in simulation. Training in simulation enables synthesis and use of unlimited training data for any desired trajectory, without putting the physical platform in danger. This includes maneuvers that stretch the abilities of even expert human pilots. To facilitate zero-shot simulation-to-reality transfer, the sensorimotor student does not directly access raw sensor input such as color images. Rather, the sensorimotor controller acts on an *abstraction* of the input, in the form of feature points extracted via classic computer vision. Such abstraction supports sample-efficient training, generalization, and simulation-to-reality transfer [202, 312].

The trained sensorimotor student does not rely on any privileged information and can be deployed directly on the physical platform. We deploy the trained controller to perform acrobatic maneuvers in the physical world, with no adaptation required.

The next section presents each aspect of our method in detail.

F.4 Method

We define the task of flying acrobatic maneuvers with a quadrotor as a discrete-time, continuous-valued optimization problem. Our task is to find an end-to-end control policy $\pi\colon \mathbb{O} \rightarrow \mathbb{U}$, defined by a neural network, which minimizes the following finite-horizon objective:

$$\min_{\pi} \qquad J(\pi) = \mathbb{E}_{\rho(\pi)}\left[\sum_{k=0}^{k=T} \mathcal{C}(\boldsymbol{\tau}_r[k], \boldsymbol{s}[k])\right], \tag{F.1}$$

where $\mathcal{C}$ is a quadratic cost depending on a reference trajectory $\boldsymbol{\tau}_r[k]$ and the quadrotor state $\boldsymbol{s}[k]$, and $\rho(\pi)$ is the distribution of possible trajectories $\{(\boldsymbol{s}[0], \boldsymbol{o}[0], \boldsymbol{u}[0]), \ldots, (\boldsymbol{s}[T], \boldsymbol{o}[T], \boldsymbol{u}[T])\}$ induced by the policy π.

We define the quadrotor state $\boldsymbol{s}[k] = [\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\omega}]$ as the platform position $\boldsymbol{p}$, its orientation quaternion $\boldsymbol{q}$, and their derivatives. Note that the agent π does not directly observe the state $\boldsymbol{s}[k]$. We further define the reference trajectory $\boldsymbol{\tau}_r[k]$ as a time sequence of reference states which describe the desired trajectory. We formulate the cost $\mathcal{C}$ as

$$\mathcal{C}(\boldsymbol{\tau}_r[k], \boldsymbol{s}[k]) = \boldsymbol{x}[k]^\top \mathcal{L} \boldsymbol{x}[k], \tag{F.2}$$

where $\boldsymbol{x}[k] = \boldsymbol{\tau}_r[k] - \boldsymbol{s}[k]$ denotes the difference between the state of the platform and the corresponding reference at time k, and $\mathcal{L}$ is a positive-semidefinite cost matrix.

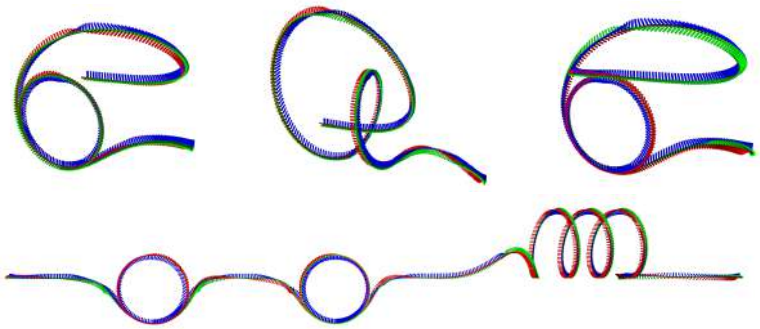


Figure F.2 – Reference trajectories for acrobatic maneuvers. Top row, from left to right: Power Loop, Barrel Roll, and Matty Flip. Bottom row: Combo.

F.4.1 Reference Trajectories

Both the privileged expert and the learned policy assume access to a reference trajectory $\boldsymbol{\tau}_r[k]$ that specifies an acrobatic maneuver. To ensure that such reference is dynamically feasible, it has to satisfy constraints that are imposed by the physical limits and the underactuated nature of the quadrotor platform. Neglecting aerodynamic drag and motor dynamics, the dynamics of the quadrotor can be modelled as

$$\begin{aligned} \dot{\boldsymbol{p}}_{\mathrm{WB}} &= \boldsymbol{v}_{\mathrm{WB}} \\ \dot{\boldsymbol{v}}_{\mathrm{WB}} &= {}_{\mathrm{W}}\boldsymbol{g} + \boldsymbol{q}_{\mathrm{WB}} \odot \boldsymbol{c}_{\mathrm{B}} \\ \dot{\boldsymbol{q}}_{\mathrm{WB}} &= \frac{1}{2}\Lambda\left(\boldsymbol{\omega}_{\mathrm{B}}\right) \cdot \boldsymbol{q}_{\mathrm{WB}} \\ \dot{\boldsymbol{\omega}}_{\mathrm{B}} &= \boldsymbol{J}^{-1} \cdot \left(\boldsymbol{\eta} - \boldsymbol{\omega}_{\mathrm{B}} \times \boldsymbol{J} \cdot \boldsymbol{\omega}_{\mathrm{B}}\right) \ , \end{aligned} \tag{F.3}$$

where $\boldsymbol{p}_{\mathrm{WB}}$, $\boldsymbol{v}_{\mathrm{WB}}$, $\boldsymbol{q}_{\mathrm{WB}}$ denote the position, linear velocity, and orientation of the platform body frame with respect to the world frame. The gravity vector ${}_{\mathrm{W}}\boldsymbol{g}$ is expressed in the world frame and $\boldsymbol{q}_{\mathrm{WB}} \odot \boldsymbol{c}_{\mathrm{B}}$ denotes the rotation of the mass-normalized thrust vector $\boldsymbol{c}_{\mathrm{B}} = (0, 0, c)^{\top}$ by quaternion $\boldsymbol{q}_{\mathrm{WB}}$. The time derivative of a quaternion $\boldsymbol{q} = (q_w, q_x, q_y, q_z)^{\top}$ is given by $\dot{\boldsymbol{q}} = \frac{1}{2}\Lambda(\boldsymbol{\omega}) \cdot \boldsymbol{q}$ and $\Lambda(\boldsymbol{\omega})$ is a skew-symmetric matrix of the vector $(0, \boldsymbol{\omega}^{\top})^{\top} = (0, \omega_x, \omega_y, \omega_z)^{\top}$. The diagonal matrix $\boldsymbol{J} = \mathrm{diag}(J_{xx}, J_{yy}, J_{zz})$ denotes the quadrotor inertia, and $\boldsymbol{\eta} \in \mathbb{R}^3$ are the torques acting on the body due to the motor thrusts.

Instead of directly planning in the full state space, we plan reference trajectories in the space of *flat outputs* $\boldsymbol{z} = [x, y, z, \psi]^{\top}$ proposed in [190], where x, y, z denote the position of the quadrotor and ψ is the yaw angle. It can be shown that any smooth trajectory in the space of flat outputs can be tracked by the underactuated platform (assuming reasonably bounded derivatives).

The core part of each acrobatic maneuver is a circular motion primitive with constant tangential velocity v_l. The orientation of the quadrotor is constrained by the thrust vector the platform needs to produce. Consequently, the desired platform orientation is undefined when there is

no thrust. To ensure a well-defined reference trajectory through the whole circular maneuver, we constrain the norm of the tangential velocity v_l to be larger by a margin ε than the critical tangential velocity that would lead to free fall at the top of the maneuver:

$$\|v_l\| > \varepsilon\sqrt{rg}, \tag{F.4}$$

where r denotes the radius of the loop, $g = 9.81\,\mathrm{m\,s^{-2}}$, and $\varepsilon = 1.1$.

While the circular motion primitives form the core part of the agile maneuvers, we use constrained polynomial trajectories to enter, transition between, and exit the maneuvers. A polynomial trajectory is described by four polynomial functions specifying the independent evolution of the components of z over time:

$$z_i(t) = \sum_{j=0}^{j=P_i} a_{ij} \cdot t^j \quad \text{for} \quad i \in \{0,1,2,3\}\ . \tag{F.5}$$

We use polynomials of order $P_i = 7$ for the position components $(i = \{0,1,2\})$ of the flat outputs and $P_i = 2$ for the yaw component $(i = 3)$. By enforcing continuity for both start $(t = 0)$ and end $(t = t_m)$ of the trajectory down to the 3rd derivative of position, the trajectory is fully constrained. We minimize the execution time t_m of the polynomial trajectories, while constraining the maximum speed, thrust, and body rates throughout the maneuver.

Finally, the trajectories are concatenated to the full reference trajectory, which is then converted back to the full state-space representation $\tau_r(t)$ [190]. Subsequent sampling with a frequency of 50 Hz results in the discrete-time representation $\tau_r[k]$ of the reference trajectory. Some example trajectories are illustrated in Figure F.2.

F.4.2 Privileged Expert

Our privileged expert π^* consists of an MPC [67] that generates collective thrust and body rates via an optimization-based scheme. The controller operates on the simplified dynamical model of a quadrotor proposed in [201]:

$$\begin{aligned} \dot{p}_{\mathrm{WB}} &= v_{\mathrm{WB}} \\ \dot{v}_{\mathrm{WB}} &= {}_{\mathrm{W}}g + q_{\mathrm{WB}} \odot c_{\mathrm{B}} \\ \dot{q}_{\mathrm{WB}} &= \frac{1}{2}\Lambda\left(\omega_B\right) \cdot q_{\mathrm{WB}} \end{aligned} \tag{F.6}$$

In contrast to the model (F.3), the simplified model neglects the dynamics of the angular rates. The MPC repeatedly optimizes the open-loop control problem over a receding horizon of N time steps and applies the first control command from the optimized sequence. Specifically, the action

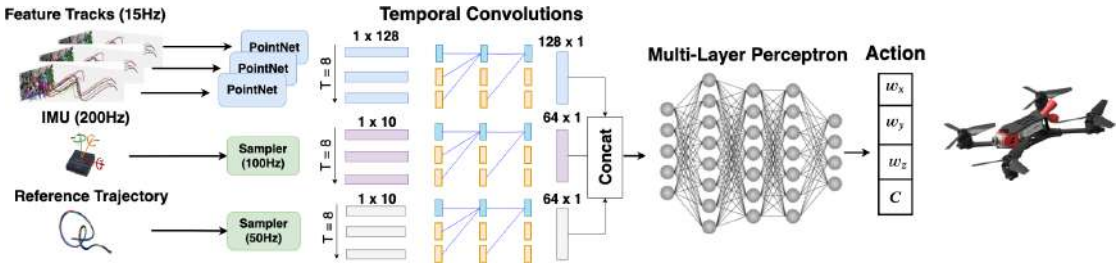


Figure F.3 – Network architecture. The network receives a history of feature tracks, IMU measurements, and reference trajectories as input. Each input modality is processed using temporal convolutions and updated at different input rates. The resulting intermediate representations are processed by a multi-layer perceptron at a fixed output rate to produce collective thrust and body rate commands.

computed by the MPC is the first element of the solution to the following optimization problem:

$$\begin{aligned} \pi^* = \min_{\boldsymbol{u}} \Bigg\{ & \boldsymbol{x}[N]^\top \mathcal{Q} \boldsymbol{x}[N] \\ & + \sum_{k=1}^{N-1} \Big(\boldsymbol{x}[k]^\top \mathcal{Q} \boldsymbol{x}[k] + \boldsymbol{u}[k]^\top \mathcal{R} \boldsymbol{u}[k] \Big) \Bigg\} \\ \text{s.t.} \quad & \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{u}) = 0 \\ & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}) \leq 0, \end{aligned} \tag{F.7}$$

where $\boldsymbol{x}[k] = \boldsymbol{\tau}_r[k] - \boldsymbol{s}[k]$ denotes the difference between the state of the platform at time k and the corresponding reference $\boldsymbol{\tau}_r[k]$, $\boldsymbol{r}(\boldsymbol{x}, \boldsymbol{u})$ are equality constraints imposed by the system dynamics (F.6), and $\boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u})$ are optional bounds on inputs and states. $\mathcal{Q}, \mathcal{R}$ are positive-semidefinite cost matrices.

F.4.3 Learning

The sensorimotor controller is trained by imitating demonstrations provided by the privileged expert. While the expert has access to privileged information in the form of ground-truth state estimates, the sensorimotor controller does not access any privileged information and can be directly deployed in the physical world [37].

A lemma by Pan et al. [218] formally defines an upper bound between the expert and the student performance as

$$\begin{aligned} J(\pi) - J(\pi^*) &\leq C_{\pi^*} \mathbb{E}_{\rho(\pi)} \Big[DW(\pi, \pi^*) \Big] \\ &\leq C_{\pi^*} \mathbb{E}_{\rho(\pi)} \mathbb{E}_{\boldsymbol{u}^* \sim \pi^*} \mathbb{E}_{\boldsymbol{u} \sim \pi} [\| \boldsymbol{u}^* - \boldsymbol{u} \|], \end{aligned} \tag{F.8}$$

where $DW(\cdot, \cdot)$ is the Wasserstein metric [90] and C_{π^*} is a constant depending on the smoothness of expert actions. Finding an agent π with the same performance as the privileged controller π^*

boils down to minimizing the discrepancy in actions between the two policies on the expected agent trajectories $\rho(\pi)$.

The aforementioned discrepancy can be minimized by an iterative supervised learning process known as DAGGER [239]. This process iteratively collects data by letting the student control the platform, annotating the collected observations with the experts' actions, and updating the student controller based on the supervised learning problem

$$\pi = \min_{\hat{\pi}} \mathbb{E}_{\boldsymbol{s}[k]\sim\rho(\pi)}[\|\boldsymbol{u}^*(\boldsymbol{s}[k]) - \hat{\pi}(\boldsymbol{o}[k])\|], \tag{F.9}$$

where $\boldsymbol{u}^*(\boldsymbol{s}[k])$ is the expert action and $\boldsymbol{o}[k]$ is the observation vector in the state $\boldsymbol{s}[k]$. Running this process for $O(N)$ iterations yields a policy π with performance $J(\pi) \leq J(\pi^*) + O(N)$ [239].

Naive application of this algorithm to the problem of agile flight in the physical world presents two major challenges: how can the expert access the ground-truth state $\boldsymbol{s}[k]$ and how can we protect the platform from damage when the partially trained student π is in control? To circumvent these challenges, we train *exclusively* in simulation. This significantly simplifies the training procedure, but presents a new hurdle: how do we minimize the difference between the sensory input received by the controller in simulation and reality?

Our approach to bridging the gap between simulation and reality is to leverage *abstraction* [202]. Rather than operating on raw sensory input, our sensorimotor controller operates on an intermediate representation produced by a perception module [312]. This intermediate representation is more consistent across simulation and reality than raw visual input.

We now formally show that training a network on abstractions of sensory input reduces the gap between simulation and reality. Let $M(\boldsymbol{z} \mid \boldsymbol{s}), L(\boldsymbol{z} \mid \boldsymbol{s}) \colon \mathbb{S} \to \mathbb{O}$ denote the observation models in the real world and in simulation, respectively. Such models describe how an onboard sensor measurement $\boldsymbol{z}$ senses a state $\boldsymbol{s}$. We further define $\pi_r = \mathbb{E}_{\boldsymbol{o}_r\sim M(\boldsymbol{s})}[\pi(\boldsymbol{o}_r[k])]$ and $\pi_s = \mathbb{E}_{\boldsymbol{o}_s\sim L(\boldsymbol{s})}[\pi(\boldsymbol{o}_s[k])]$ as the realizations of the policy π in the real world and in simulation. The following lemma shows that, disregarding actuation differences, the distance between the observation models upper-bounds the gap in performance in simulation and reality.

Lemma F.4.1. *For a Lipschitz-continuous policy π the simulation-to-reality gap $J(\pi_r) - J(\pi_s)$ is upper-bounded by*

$$J(\pi_r) - J(\pi_s) \leq C_{\pi_s} K \mathbb{E}_{\rho(\pi_r)}\Big[DW(M, L)\Big], \tag{F.10}$$

where K denotes the Lipschitz constant.

Proof. The lemma follows directly from (F.8) and the fact that

$$\begin{aligned} DW(\pi_r, \pi_s) &= \inf_{\gamma \in \Pi(\boldsymbol{o}_r, \boldsymbol{o}_s)} \mathbb{E}_{(\boldsymbol{o}_r, \boldsymbol{o}_s)}[d_p(\pi_r, \pi_s)] \\ &\leq K \inf_{\gamma \in \Pi(\boldsymbol{o}_r, \boldsymbol{o}_s)} \mathbb{E}_{(\boldsymbol{o}_r, \boldsymbol{o}_s)}[d_o(\boldsymbol{o}_r, \boldsymbol{o}_s)] \\ &= K \cdot DW(M, L), \end{aligned}$$

where d_o and d_p are distances in observation and action space, respectively. □

We now consider the effect of abstraction of the input observations. Let f be a mapping of the observations such that

$$DW(f(M), f(L)) \leq DW(M, L). \tag{F.11}$$

The mapping f is task-dependent and is generally designed – with domain knowledge – to contain sufficient information to solve the task while being invariant to nuisance factors. In our case, we use feature tracks as an abstraction of camera frames. The feature tracks are provided by a visual-inertial odometry (VIO) system. In contrast to camera frames, feature tracks primarily depend on scene geometry, rather than surface appearance. We also make inertial measurements independent of environmental conditions, such as temperature and pressure, by integration and de-biasing. As such, our input representations fulfill the requirements of Eq. (F.11).

As the following lemma shows, training on such representations reduces the gap between task performance in simulation and the real world.

Lemma F.4.2. *A policy that acts on an abstract representation of the observations $\pi_f \colon f(\mathbb{O}) \to \mathbb{U}$ has a lower simulation-to-reality gap than a policy $\pi_o \colon \mathbb{O} \to \mathbb{U}$ that acts on raw observations.*

Proof. The lemma follows directly from (F.10) and (F.11). □

F.4.4 Sensorimotor Controller

In contrast to the expert policy π^*, the student policy π is only provided with onboard sensor measurements from the forward-facing camera and the IMU. There are three main challenges for the controller to tackle: (i) it should keep track of its state based on the provided inputs, akin to a visual-inertial odometry system [72, 59], (ii) it should be invariant to environments and domains, so as to not require retraining for each scene, and (iii) it should process sensor readings that are provided at different frequencies.

We represent the policy as a neural network that fulfills all of the above requirements. The network consists of three input branches that process visual input, inertial measurements, and the

Maneuver	Input			Power Loop		Barrel Roll		Matty Flip		Combo	
	Ref	IMU	FT	Error (↓)	Success (↑)	Error (↓)	Success (↑)	Error (↓)	Success (↑)	Error (↓)	Success (↑)
VIO-MPC	✓	✓	✓	43 ± 14	**100%**	79 ± 43	**100%**	92 ± 41	**100%**	164 ± 51	70%
Ours (Only Ref)	✓			250 ± 50	20%	485 ± 112	85%	340 ± 120	15%	∞	0 %
Ours (No IMU)	✓		✓	210 ± 100	30%	543 ± 95	85%	380 ± 100	20%	∞	0 %
Ours (No FT)	✓	✓		28 ± 8	**100%**	64 ± 24	95%	67 ± 29	**100%**	134 ± 113	85%
Ours	✓	✓	✓	$\mathbf{24 \pm 5}$	**100%**	$\mathbf{58 \pm 9}$	**100%**	$\mathbf{53 \pm 15}$	**100%**	$\mathbf{128 \pm 57}$	**95%**

Table F.1 – Comparison of different variants of our approach with the baseline (VIO-MPC) in terms of the average tracking error in centimeters and the success rate. Results were averaged over 20 runs. Agents without access to IMU data perform poorly. An agent that has access only to IMU measurements has a significantly lower tracking error than the baseline. Adding feature tracks further improves tracking performance and success rate, especially for longer and more complicated maneuvers.

reference trajectory, followed by a multi-layer perceptron that produces actions. The architecture is illustrated in Fig. F.3. Similarly to visual-inertial odometry systems [44, 59, 72], we provide the network with a representation of the platform state by supplying it with a history of length $L = 8$ of visual and inertial information.

To ensure that the learned policies are scene- and domain-independent, we provide the network with appropriate abstractions of the inputs instead of directly feeding raw inputs. We design these abstractions to contain sufficient information to solve the task while being invariant to environmental factors that are hard to simulate accurately and are thus unlikely to transfer from simulation to reality.

The distribution of raw IMU measurements depends on the exact sensor as well as environmental factors such as pressure and temperature. Instead of using the raw measurements as input to the policy, we preprocess the IMU signal by applying bias subtraction and gravity alignment. Modern visual-inertial odometry systems perform a similar pre-integration of the inertial data in their optimization backend [227]. The resulting inertial signal contains the estimated platform velocity, orientation, and angular rate.

We use the history of filtered inertial measurements, sampled at 100 Hz, and process them using temporal convolutions [11]. Specifically, the inertial branch consists of a temporal convolutional layer with 128 filters, followed by three temporal convolutions with 64 filters each. A final fully-connected layer maps the signal to a 128-dimensional representation.

Another input branch processes a history of reference velocities, orientations, and angular rates. It has the same structure as the inertial branch. New reference states are added to the history at a rate of 50 Hz.

For the visual signal, we use *feature tracks*, i.e. the motion of salient keypoints in the image plane, as an abstraction of the input. Feature tracks depend on the scene structure, ego-motion, and image gradients, but not on absolute pixel intensities. At the same time, the information contained in the feature tracks is sufficient to infer the ego-motion of the platform up to an unknown scale. Information about the scale can be recovered from the inertial measurements. We leverage

the computationally efficient feature extraction and tracking frontend of VINS-Mono [227] to generate feature tracks. The frontend extracts Harris corners [103] and tracks them using the Lucas-Kanade method [178]. We perform geometric verification and exclude correspondences with a distance of more than 2 pixels from the epipolar line. We represent each feature track by a 5-dimensional vector that consists of the keypoint position, its displacement with respect to the previous keyframe (both on the rectified image plane), and the number of keyframes that the feature has been tracked (a measure of keypoint quality).

To facilitate efficient batch training, we randomly sample 40 keypoints per keyframe. The features are processed by a reduced version of the PointNet architecture proposed in [228] before we generate a fixed-size representation at each timestep. Specifically, we reduce the number of hidden layers from 6 to 4, with $32, 64, 128, 128$ filters, respectively, in order to reduce latency. The output of this subnetwork is reduced to a 128-dimensional vector by global average pooling over the feature tracks. The history of resulting hidden representations is then processed by a temporal convolutional network that has the same structure as the inertial and reference branches.

Finally, the outputs of the individual branches are concatenated and processed by a synchronous multi-layer perceptron with three hidden layers of size $128, 64, 32$. The final outputs are the body rates and collective thrust that are used to control the platform.

We account for the different input frequencies by allowing each of the input branches to operate asynchronously. Each branch operates independently from the others by generating an output only when a new input from the sensor arrives. The multi-layer perceptron uses the latest outputs from the asynchronous branches and operates at 100 Hz. It outputs control commands at approximately the same rate due to its minimal computational overhead.

F.4.5 Implementation Details

We use the Gazebo simulator to train our policies. Gazebo can model the physics of quadrotors with high fidelity using the RotorS extension [77]. We simulate the AscTec Hummingbird multirotor, which is equipped with a forward-facing fisheye camera. The platform is instantiated in a cubical simulated flying space with a side length of 70 meters. An example image is shown in Fig. F.4 (left).

For the real-world experiments we use a custom quadrotor that weighs 1.15 kg and has a thrust-to-weight ratio of 4:1. We use a Jetson TX2 for neural network inference. Images and inertial measurements are provided by an Intel RealSense T265 camera.

We use an off-policy learning approach. We execute the trained policy, collect rollouts, and add them to a dataset. After 30 new rollouts are added, we train for 40 epochs on the entire dataset. This collect-rollouts-and-train procedure is repeated 5 times: there are 150 rollouts in the dataset by the end. We use the Adam optimizer [148] with a learning rate of $3e-4$. We always use the latest available model for collecting rollouts. We execute a student action only if the difference to

Figure F.4 – Example images from simulation (left) and the real test environment (right).

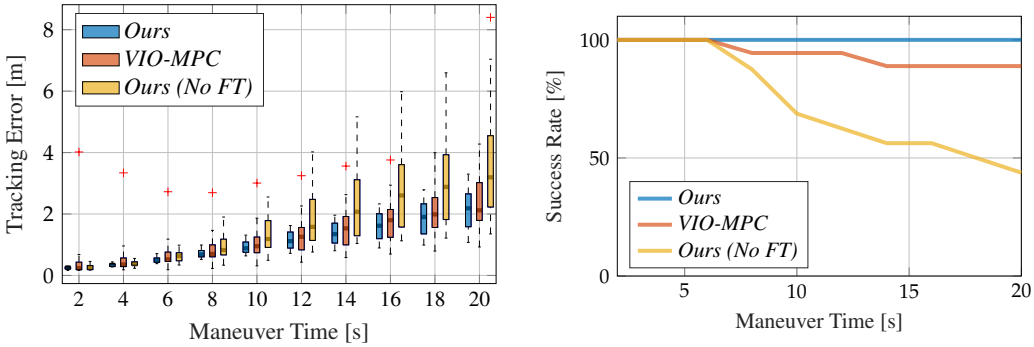


Figure F.5 – Tracking error (left) and success rate (right) over time when a maneuver is executed repeatedly in simulation. The controllers were trained to complete the maneuver for six seconds and generalize well to longer sequences. Our learned controller, which leverages both IMU and visual data, provides consistently good performance without a single failure.

the expert action is smaller than a threshold $t = 1.0$ to avoid crashes in the early stages of training. We double the threshold t every 30 rollouts. We perform a random action with a probability $p = 30\%$ at every stage of the training to increase the coverage of the state space. To facilitate transfer from simulation to reality, we randomize the IMU biases and the thrust-to-weight ratio of the platform by up to 10% of their nominal value in every iteration. We do not perform any randomization of the geometry and appearance of the scene during data collection.

F.5 Experiments

We design our evaluation procedure to address the following questions. Is the presented sensorimotor controller advantageous to a standard decomposition of state estimation and control? What is the role of input abstraction in facilitating transfer from simulation to reality? Finally, we validate our design choices with ablation studies.

F.5.1 Experimental Setup

We learn sensorimotor policies for three acrobatic maneuvers that are popular among professional drone pilots as well as a policy that consists of a sequence of multiple maneuvers.

1. Power Loop: Accelerate over a distance of $4\,\mathrm{m}$ to a speed of $4.5\,\mathrm{m\,s^{-1}}$ and enter a loop maneuver with a radius of $r = 1.5\,\mathrm{m}$.
2. Barrel Roll: Accelerate over a distance of $3\,\mathrm{m}$ to a speed of $4.5\,\mathrm{m\,s^{-1}}$ and enter a roll maneuver with a radius of $r = 1.5\,\mathrm{m}$.
3. Matty Flip: Accelerate over a distance of $4\,\mathrm{m}$ to a speed of $4.5\,\mathrm{m\,s^{-1}}$ while yawing $180°$ and enter a backwards loop maneuver with a radius of $r = 1.5\,\mathrm{m}$.
4. Combo: This sequence starts with a triple Barrel Roll, followed by a double Power Loop, and ends with a Matty Flip. The full maneuver is executed without stopping between maneuvers.

The maneuvers are listed by increasing difficulty. The trajectories of these maneuvers are illustrated in Fig. F.2. They contain high accelerations and fast angular velocities around the body axes of the platform. All maneuvers start and end in the hover condition.

For comparison, we construct a strong baseline by combining visual-inertial odometry [227] and model predictive control [67]. Our baseline receives the same inputs as the learned controllers: inertial measurements, camera images, and a reference trajectory.

We define two metrics to compare different approaches. We measure the average root mean square error in meters of the reference position with respect to the true position of the platform during the execution of the maneuver. Note that we can measure this error only for simulation experiments, as it requires exact state estimation. We thus define a second metric, the average success rate for completing a maneuver. In simulation, we define success as not crashing the platform into any obstacles during the maneuver. For the real-world experiments, we consider a maneuver successful if the safety pilot did not have to intervene during the execution and the maneuver was executed correctly.

F.5.2 Experiments in Simulation

We first evaluate the performance for individual maneuvers in simulation. The results are summarized in Table F.1. The learned sensorimotor controller that has access to both visual and inertial data (*Ours*) is consistently the best across all maneuvers. This policy exhibits a lower tracking error by up to 45% in comparison to the strong *VIO-MPC* baseline. The baseline can complete the simpler maneuvers with perfect success rate, but generally has higher tracking error due to drift in state estimation. The gap between the baseline and our controller widens for longer and more difficult sequences.

Input	Train		Test 1		Test 2	
	Error (↓)	Success (↑)	Error (↓)	Success (↑)	Error (↓)	Success (↑)
Image	90 ± 32	80%	∞	0%	∞	0%
Ours	$\mathbf{53 \pm 15}$	**100%**	$\mathbf{58 \pm 18}$	**100%**	$\mathbf{61 \pm 11}$	**100%**

Table F.2 – Sim-to-sim transfer for different visual input modalities. Policies that directly rely on images as input do not transfer to scenes with novel appearance (Test 1, Test 2). Feature tracks enable reliable transfer. Results are averaged over 10 runs.

Table F.1 also highlights the relative importance of the input modalities. Policies that only receive the reference trajectories but no sensory input (*Ours (Only Ref)*) – effectively operating open-loop – perform poorly across all maneuvers. Policies that have access to visual input but not to inertial data *(Ours (No IMU))* perform similarly poorly since they do not have sufficient information to determine the absolute scale of the ego-motion. On the other hand, policies that only rely on inertial data for sensing *(Ours (No FT))* are able to safely fly most maneuvers. Even though such controllers only have access to inertial data, they exhibit significantly lower tracking error than the *VIO-MPC* baseline. However, the longer the maneuver, the larger the drift accumulated by purely-inertial *(Ours (No FT))* controllers. When both inertial and visual data is incorporated *(Ours)*, drift is reduced and accuracy improves. For the longest sequence *(Combo)*, the abstracted visual input raises the success rate by 10 percentage points.

Fig. F.5 analyzes the evolution of tracking errors and success rates of different methods over time. For this experiment, we trained a policy to repeatedly fly barrel rolls for four seconds. We evaluate robustness and generalization of the learned policy by flying the maneuver for up to 20 seconds at test time. The results again show that (abstracted) visual input reduces drift and increases robustness. The controller that has access to both visual and inertial data *(Ours)* is able to perform barrel rolls for 20 seconds without a single failure.

To validate the importance of input abstraction, we compare our approach to a network that uses raw camera images instead of feature tracks as visual input. This network substitutes the PointNet in the input branch with a 5-layer convolutional network that directly operates on image pixels, but retains the same structure otherwise. We train this network on the Matty Flip and evaluate its robustness to changes in the background images. The results are summarized in Table F.2. In the training environment, the image-based network has a success rate of only 80%, with a 58% higher tracking error than the controller that receives an abstraction of the visual input in the form of feature tracks *(Ours)*. We attribute this to the higher sample complexity of learning from raw pixels [312]. Even more dramatically, the image-based controller fails completely when tested with previously unseen background images (*Test 1*, *Test 2*). (For backgrounds, we use randomly sampled images from the COCO dataset [166].) In contrast, our approach maintains a 100% success rate in these conditions.

Maneuver	Power Loop	Barrel Roll	Matty Flip
Ours (No FT)	100%	90%	100%
Ours	100%	100%	100%

Table F.3 – Success rate across 10 runs on the physical platform.

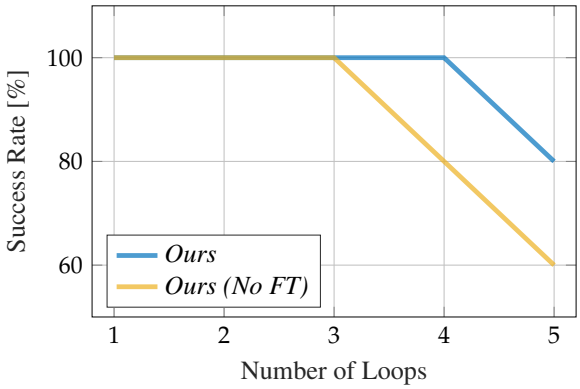


Figure F.6 – Number of successful back-to-back Power Loops on the physical quadrotor before the human expert pilot had to intervene. Results are averaged over 5 runs.

F.5.3 Deployment in the Physical World

We now perform direct simulation-to-reality transfer of the learned controllers. We use exactly the same sensorimotor controllers that were learned in simulation and quantitatively evaluated in Table F.1 to fly a physical quadrotor, with no fine-tuning. Despite the differences in the appearance of simulation and reality (see Fig. F.4), the abstraction scheme we use facilitates successful deployment of simulation-trained controllers in the physical world. The controllers are shown in action in the supplementary video.

We further evaluate the learned controllers with a series of quantitative experiments on the physical platform. The success rates of different maneuvers are shown in Table F.3. Our controllers can fly all maneuvers with no intervention. An additional experiment is presented in Fig. F.6, where a controller that was trained for a single loop was tested on repeated execution of the maneuver with no breaks. The results indicate that using all input modalities, including the abstracted visual input in the form of feature tracks, enhances robustness.

F.6 Conclusion

Our approach is the first to enable an autonomous flying machine to perform a wide range of acrobatics maneuvers that are highly challenging even for expert human pilots. The approach relies solely on onboard sensing and computation, and leverages sensorimotor policies that are trained entirely in simulation. We have shown that designing appropriate abstractions of the input facilitates direct transfer of the policies from simulation to physical reality. The presented

methodology is not limited to autonomous flight and can enable progress in other areas of robotics.

G Agile Autonomy: Learning High-Speed Flight in the Wild

The version presented here is reprinted, with permission, from:

Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Agile Autonomy: Learning High-Speed Flight in the Wild”. In: *Science Robotics* 7 (58 2021), pp. 1–12

Learning High-Speed Flight in the Wild Without a Single Real Image

Antonio Loquercio, Elia Kaufmann, Rene Ranftl, Matthias Mueller, Vladlen Koltun and

Davide Scaramuzza

Abstract — Quadrotors are agile. Unlike most other machines, they can traverse extremely complex environments at high speeds. To date, only expert human pilots have been able to fully exploit their capabilities. Autonomous operation with onboard sensing and computation has been limited to low speeds. State-of-the-art methods generally separate the navigation problem into subtasks: sensing, mapping, and planning. While this approach has proven successful at low speeds, the separation it builds upon can be problematic for high-speed navigation in cluttered environments. Indeed, the subtasks are executed sequentially, leading to increased latency and a compounding of errors through the pipeline. Here we propose an end-to-end approach that can autonomously fly quadrotors through complex natural and man-made environments at high speeds, with purely onboard sensing and computation. The key principle is to directly map noisy sensory observations to collision-free trajectories in a receding-horizon fashion. This direct mapping drastically reduces latency and increases robustness to noisy and incomplete perception. The sensorimotor mapping is performed by a convolutional network that is trained *exclusively* in simulation via privileged learning: imitating an expert with access to privileged information. By simulating realistic sensor noise, our approach achieves zero-shot transfer from simulation to challenging real-world environments that were never experienced during training: dense forests, snow-covered terrain, derailed trains, and collapsed buildings. Our work demonstrates that end-to-end policies trained in simulation enable high-speed autonomous flight through challenging environments, outperforming traditional obstacle avoidance pipelines.

Videos of the Experiments

A video of the experiments reported in this manuscript is available at https://youtu.be/uTWcC6IBsE4

G.1 Introduction

Quadrotors are among the most agile and dynamic machines ever created[1] [291, 4]. Thanks to their agility, they can traverse complex environments, ranging from cluttered forests to urban canyons, and reach locations that are otherwise inaccessible to humans and machines alike. This ability has led to their application in fields such as search and rescue, logistics, security, infrastructure, entertainment, and agriculture [112]. In the majority of these existing applications, the quadrotor needs to be controlled by expert human pilots, who take years to train, and are thus an expensive and scarce resource. Infusing quadrotors with autonomy, that is the capability to safely operate in the world without the need for human intervention, has the potential to massively enhance their usefulness and to revolutionize whole industries. However, the development of autonomous quadrotors that can navigate in complex environments with the agility and safety of expert human pilots or birds is a long-standing problem that is still open.

The limiting factor for autonomous agile flight in arbitrary unknown environments is the coupling of fast and robust perception with effective planning. The perception system has to be robust to disturbances such as sensor noise, motion blur, and changing illumination conditions. In addition, an effective planner is necessary to find a path that is both dynamically feasible and collision-free while relying only on noisy and partial observations of the environment. These requirements, together with the limited computational resources that are available onboard, make it difficult to achieve reliable perception and planning at low latency and high speeds [64].

Various approaches to enable autonomous flight have been proposed in the literature. Some works tackle only perception and build high-quality maps from imperfect measurements [109, 79, 250, 62, 19], while others focus on planning without considering perception errors [28, 233, 5, 168]. Numerous systems that combine online mapping with traditional planning algorithms have been proposed to achieve autonomous flight in previously unknown environments [214, 215, 197, 15, 311, 244, 284, 42, 309]. A taxonomy of prior works is presented in Figure S2 in the Appendix.

The division of the navigation task into the mapping and planning subtasks is attractive from an engineering perspective, since it enables parallel progress on each component and makes the overall system interpretable. However, it leads to pipelines that largely neglect interactions between the different stages and thus compound errors [309]. Their sequential nature also introduces additional latency, making high-speed and agile maneuvers difficult to impossible [64]. While these issues can be mitigated to some degree by careful hand-tuning and engineering, the divide-and-conquer principle that has been prevalent in research on autonomous flight in

[1] The quadrotor used for the experiments in this paper has a maximum acceleration of 4g. Formula 1 cars achieve accelerations of up to 1.45g, and the Eurofighter Typhoon reaches a longitudinal acceleration of up to 1.15g.

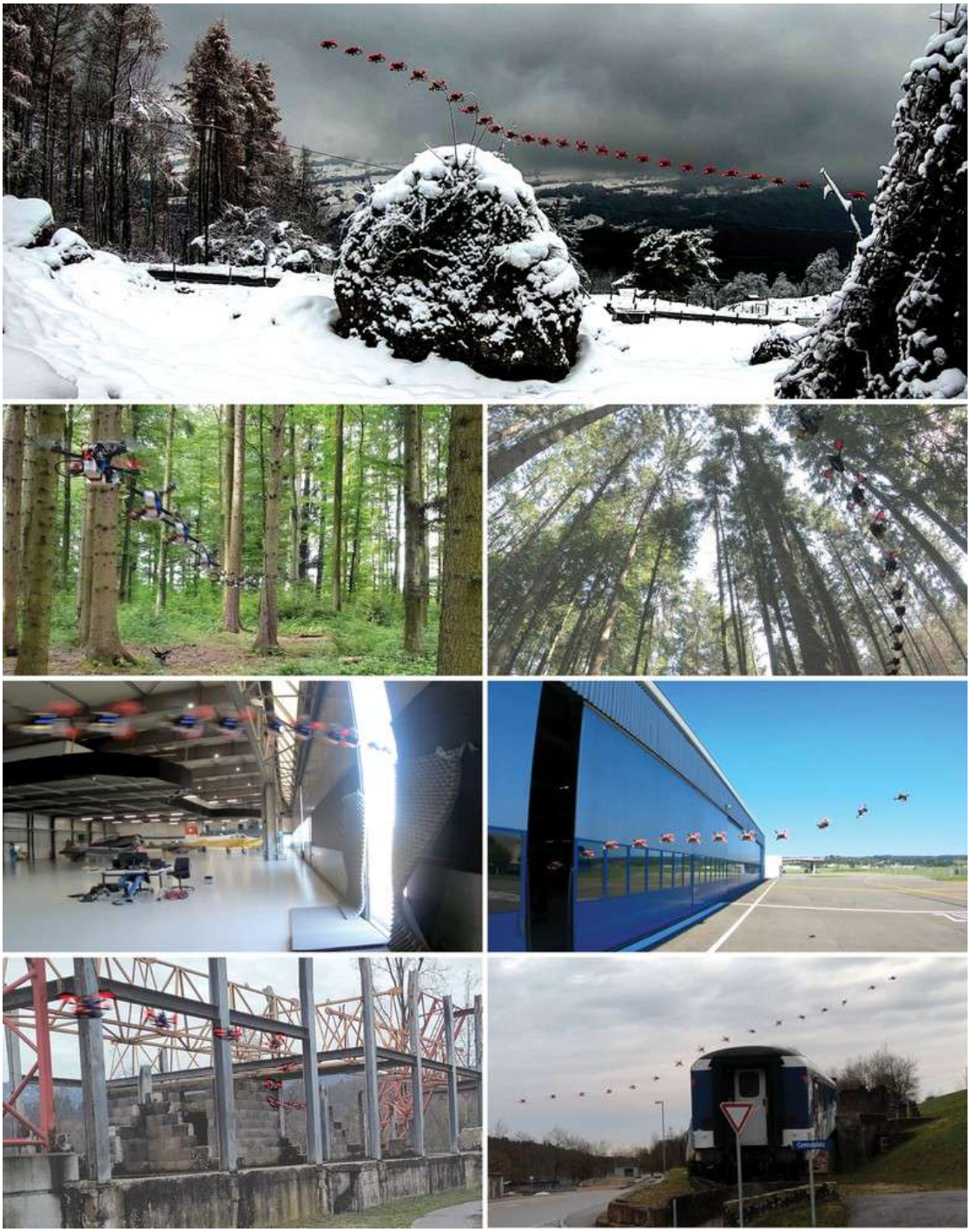


Figure G.1 – Deployment of the presented approach in challenging environments. To get a better sense of the speed and agility of the autonomous system, please watch Movie S1.

unknown environments for many years imposes fundamental limits on the speed and agility that a robotic system can achieve [170].

In contrast to these traditional pipelines, some recent works propose to learn end-to-end policies directly from data without explicit mapping and planning stages [240, 246, 84, 177]. These policies are trained by imitating a human [240, 177], from experience that was collected in simulation [246], or directly in the real world [84]. As the number of samples required to train general navigation policies is very high, existing approaches impose constraints on the quadrotor's motion model, for example by constraining the platform to planar motion [177, 84, 240] and/or discrete actions [246], at the cost of reduced maneuverability and agility. More recent work has demonstrated that very agile control policies can be trained in simulation [138]. Policies produced by the last approach can successfully perform acrobatic maneuvers, but can only operate in unobstructed free space and are essentially blind to obstacles in the environment.

Here we present an approach to fly a quadrotor at high speeds in a variety of environments with complex obstacle geometry while having access to only onboard sensing and computation. By predicting navigation commands directly from sensor measurements, we decrease the latency between perception and action while simultaneously being robust to perception artifacts, such as motion blur, missing data, and sensor noise. To deal with sample complexity and not endanger the physical platform, we train the policy *exclusively* in simulation. We leverage abstraction of the input data to transfer the policy from simulation to reality [202, 138]. To this end, we utilize a stereo matching algorithm to provide depth images as input to the policy. We show that this representation is both rich enough to safely navigate through complex environments and abstract enough to bridge simulation and reality. Our choice of input representation guarantees a strong similarity of the noise models between simulated and real observations and gives our policy robustness against common perceptual artifacts in existing depth sensors.

We train the navigation policy via privileged learning [37] on demonstrations that are provided by a novel sampling-based expert. Our expert has privileged access to a representation of the environment in the form of a 3D point cloud as well as perfect knowledge about the state of the quadrotor. Since simulation does not impose real-time constraints, the expert additionally has an unconstrained computational budget. While existing global planning algorithms [28, 5, 168] generally output a single trajectory, our expert uses Metropolis-Hastings sampling to compute a *distribution* of collision-free trajectories. This captures the multi-modal nature of the navigation task where many equally valid solutions can exist (for example, going either left or right around an obstacle). We therefore use our planner to compute trajectories with a short time horizon to ensure that they are predictable from onboard sensors and that the sampler remains computationally tractable. We bias the sampler towards obstacle-free regions by conditioning it on trajectories from a classic global planning algorithm [168].

We also reflect the multi-modal nature of the problem in the design and training of the neural network policy. Our policy takes a noisy depth image and inertial measurements as sensory inputs and produces a set of short-term trajectories together with an estimate of individual trajectory

costs. The trajectories are represented as high-order polynomials to ensure dynamical feasibility. We train the policy using a multi-hypothesis winner-takes-all loss that adaptively maps the predicted trajectories to the best trajectories that have been found by the sampling-based expert. At test time, we use the predicted trajectory costs to decide which trajectory to execute in a receding horizon. The policy network is designed to be extremely lightweight, which ensures that it can be executed onboard the quadrotor at the update rates required for high-speed flight.

The resulting policy can fly a physical quadrotor in natural and man-made environments at speeds that are unreachable by existing methods. We achieve this in a zero-shot generalization setting: we train on randomly generated obstacle courses composed of simple off-the-shelf objects, such as schematic trees and a small set of convex shapes such as cylinders and cubes. We then directly deploy the policy in the physical world without any adaptation or fine-tuning. Our platform experiences conditions at test time that were never seen during training. Examples include high dynamic range (when flying from indoor environments to outdoor environments), poorly textured surfaces (indoor environments and snow-covered terrain), thick vegetation in forests, and the irregular and complex layout of a disaster scenario (Figure G.2). These results suggest that our methodology enables a multitude of applications that rely on agile autonomous drones with purely onboard sensing and computation.

G.2 Results

Our experiments in simulation show that the proposed approach reduces the failure rate up to 10 times with respect to state-of-the-art methods. We confirm our results in a variety of real-world environments using a custom-built physical quadrotor; we deploy our policy trained in simulation without any further adaptations. In all experiments, the drone was provided with a reference trajectory, which is not collision-free (Figure G.3-C, depicted in red), to encode the intended flight path. This reference can be provided by a user or a higher-level planning algorithm. The drone is tasked to follow that flight path and make adjustments as necessary to avoid obstacles. Recordings of the experiments can be found in Movie S1.

G.2.1 High-Speed Flight in the Wild

We tested our approach in diverse real-world environments, as illustrated in Figures G.1 and G.2. High-speed flight in these environments is very challenging due to their complex structure (*e.g.* thick vegetation, thin branches, or collapsed walls) and multiple options available to avoid obstacles. In addition, a high-level understanding of the environment is necessary, for example to pass through far-away openings (Figure G.3-C) or nearby obstacles (Figure G.2-M). Flying at high speeds also necessitates low-latency perception and robustness to sensor noise, which is worsened by challenging illumination conditions and low-texture surfaces (*e.g.* due to snow). At the same time, only limited computational resources are available onboard. Despite all of these challenges, our approach was able to successfully navigate in all environments that it was

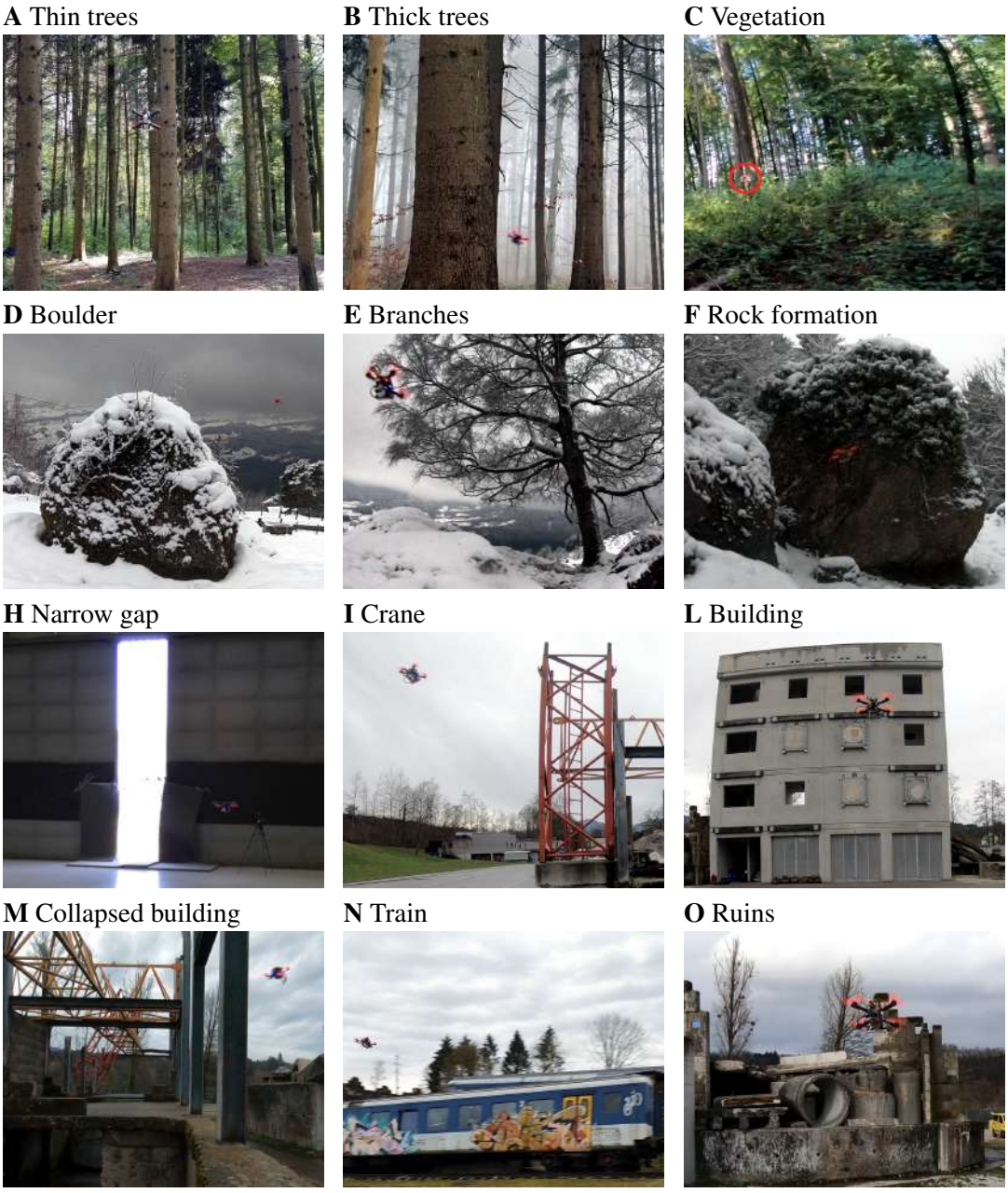


Figure G.2 – Testing Environments. Zero-shot generalization of our approach in complex natural (**A** to **F**) and man-made (**H** to **O**) environments. The encountered obstacles can often be avoided in multiple directions and have very different size and structure.

tested in. Note that our policy was trained in simulation and was never exposed to any of these environments or conditions at training time.

We measure performance according to success rate, *i.e.* the percentage of successful runs over the total number of runs, where we consider a run successful if the drone reaches the goal location

within a radius of 5 m without crashing. We performed a total of 56 experiments at different speeds. We report the cumulative and individual success rates at various speeds in Figures G.3-D and G.3-E. Our experimental platform performs state estimation and depth perception onboard using vision-based sensors. A detailed description of the platform is available in Section S1. We group the environments that were used for experiments into two classes, *natural* and *man-made*, and highlight the distinct challenges that these types of environments present for agile autonomous flight.

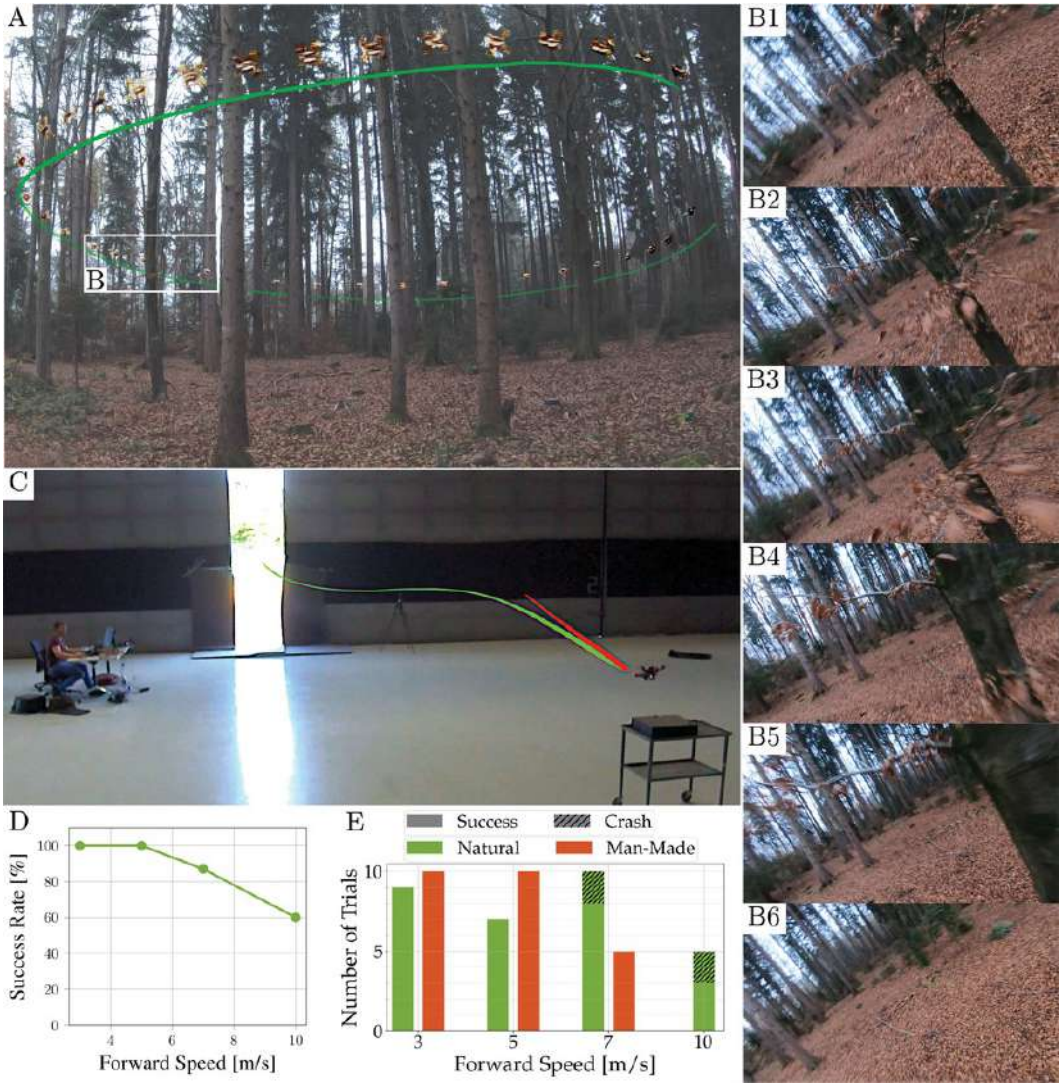


Figure G.3 – Evaluation in indoor and outdoor environments. (**A**) A circular path in the forest at an average speed of 7 m s^{-1}. (**B**) Sequence of first-person views from the maneuver in **A**, observed during the avoidance of tree branches. The maneuver requires fine and fast adaptations in height (**B2**,**B5**) and attitude (**B3**,**B4**) to avoid the vegetation. After the obstacle is passed, the drone accelerates in the direction of travel (**B6**). (**C**) Comparison of reference trajectory passing through a wall (in red), to actual flight path (in green) in a airplane hangar. (**D**) Success rates for all experiments aggregated according to flight speed. (**E**) Number of trials per environment class at different speeds.

Natural environments. We performed experiments in diverse natural environments: forests of different types and densities and steep snowy mountain terrains. Figures G.2-A to G.2-F illustrate the heterogeneity of those environments. We performed experiments with two different reference trajectories: a 40 m-long straight line and a circle with a 6 m radius (Figure G.3-A). Both trajectory types are not collision-free and would lead to a crash into obstacles if blindly executed. We flew the straight line at different average speeds in the range $3 - 10\,\mathrm{m\,s^{-1}}$. Flying at these speeds requires very precise and low-latency control of position and attitude to avoid bushes and pass through the openings between trees and branches (Figure G.3-B). Traditional mapping and planning approaches generally fail to achieve high speeds in such conditions, as both the thick vegetation and the texture-less snow terrain often cause very noisy depth observations.

We conducted a total of 31 experiments in natural environments. At average speeds of $3\,\mathrm{m\,s^{-1}}$ and $5\,\mathrm{m\,s^{-1}}$ our approach consistently completed the task without experiencing a single crash. For comparison, state-of-the-art methods with comparable actuation, sensing, and computation [310, 311] achieve in similar environments a maximum average speed of $2.29\,\mathrm{m\,s^{-1}}$. To study the limit of the system, we set the platform's average speed to $7\,\mathrm{m\,s^{-1}}$. In spite of the very high-speed, the maneuver was successfully completed in eight out of ten experiments. The two failures happened when objects entered the field of view very late due to the high angular velocity of the platform. Given the good performance at $7\,\mathrm{m\,s^{-1}}$, we push the average flight speed even higher to $10\,\mathrm{m\,s^{-1}}$. At this speed, external disturbances, *e.g.* aerodynamics, battery power drops, and motion-blur start to play a significant role and widen the simulation to reality gap. Nonetheless, we achieve a success rate of 60%, with failures mainly happening in the proximity of narrow openings less than a meter wide, where a single wrong action results in a crash.

Man-made environments. We also tested our approach in a set of man-made environments, illustrated in Figure G.2-G to G.2-O. In these environments, the drone faces a different set of challenges. It has to avoid obstacles with a variety of sizes and shapes (*e.g.* a train, a crane, a building, and ruins), slalom through concrete structures (c.f. Figure G.2-M), and exit a building through a single narrow opening (c.f. Figure G.2-H). The irregular and/or large structure of the encountered obstacles, the limited number of flyable openings, and the requirement to initiate the avoidance maneuver well in advance, offer a complementary set of challenges with respect to our natural testing environments.

As in the natural environments, we provide the drone with a straight reference trajectory with length of 40 m. The reference trajectory is in direct collision with obstacles and its blind execution would result in a crash. We performed a total of 19 experiments with flight speeds in the range of $3 - 7\,\mathrm{m\,s^{-1}}$. Given the lower success rate experienced in the forest environment at $10\,\mathrm{m\,s^{-1}}$, we did not test at higher speeds to avoid fatal crashes. As shown in Figure G.3-E, our approach is robust to zero-shot deployment in these environments, whose characteristics were never observed at training time, and consistently completes the task without crashing.

We further compare our approach to a commercial drone[2]. Specifically, the drones are required

[2]The commercial drone (a Skydio R1) was tasked to follow a person running through the narrow gap. The speed

to exit a hangar by passing through a narrow gap of about 0.8 m in width (Figure G.3-C). At the start of the experiment, the drones are placed at approximately 10 m in front of and about 5 m to the right of the gap. The task was represented by a straight reference trajectory passing through the wall (Figure G.3-C, in red). This experiment is challenging since it requires a high-level understanding of the environment to turn early enough towards the gap. The commercial drone, flying at a speed of approximately 2.7 m s^{-1} consistently failed to pass through the gap across three experiments. In two of the experiments it deemed the gap to be too small and stopped in front of it; in the third, it crashed into the wall. In contrast, the low latency and robustness to perception failures of our approach enabled the drone to successfully fly through the narrow gap every time. We performed a total of six experiments at flight speeds of both 3 m s^{-1} and 5 m s^{-1} and never experienced a crash.

G.2.2 Controlled Experiments

We perform a set of controlled experiments in simulation to compare the performance of our approach with several baselines. We select two representative state-of-the-art approaches as baselines for navigation in unknown environments: the mapping and planning method of Zhou et al. [311] (*FastPlanner*) and the reactive planner of Florence et al. [70] (*Reactive*). The first approach [311] incrementally builds a map from a series of observations and plans a trajectory in this map to reach the goal while avoiding obstacles. In contrast, the second approach [70] does not build a map but uses instantaneous depth information to select the best trajectory from a set of pre-defined motion primitives based on a cost that encodes collision and progress towards the goal.

The baselines receive the same input as our policy: the state of the platform, depth measurements from the stereo camera, and a goal in the form of a reference state that lies 1 s in the future. To provide a notion of the difficulty of the environments, we additionally show a naive baseline that blindly follows the reference trajectory to the goal without avoiding obstacles (*Blind*). As in the real-world experiments, we compare the different approaches according to their success rate, which measures how often the drone reaches the goal location within a radius of 5 meters without crashing.

We perform all experiments in the Flightmare simulator [266] using the RotorS [78] Gazebo plugin for accurate physics modeling and Unity as a rendering engine [130]. The experiments are conducted in two types of environments that resemble the setup of our real-world experiments: a forest and a narrow gap. We build these environments by adding trees and walls to the uneven ground of an out-of-the-box Unity environment[3].

Forest environments. We build a simulated forest [135] in a rectangular region $R(l, w)$ of width w and length l, and fill it with trees that have a diameter of approximately 0.6 m. Trees are ran-

of this drone cannot be enforced nor controlled, and was therefore estimated in post-processing.

[3] https://assetstore.unity.com/packages/3d/vegetation/forest-environment-dynamic-nature-150668

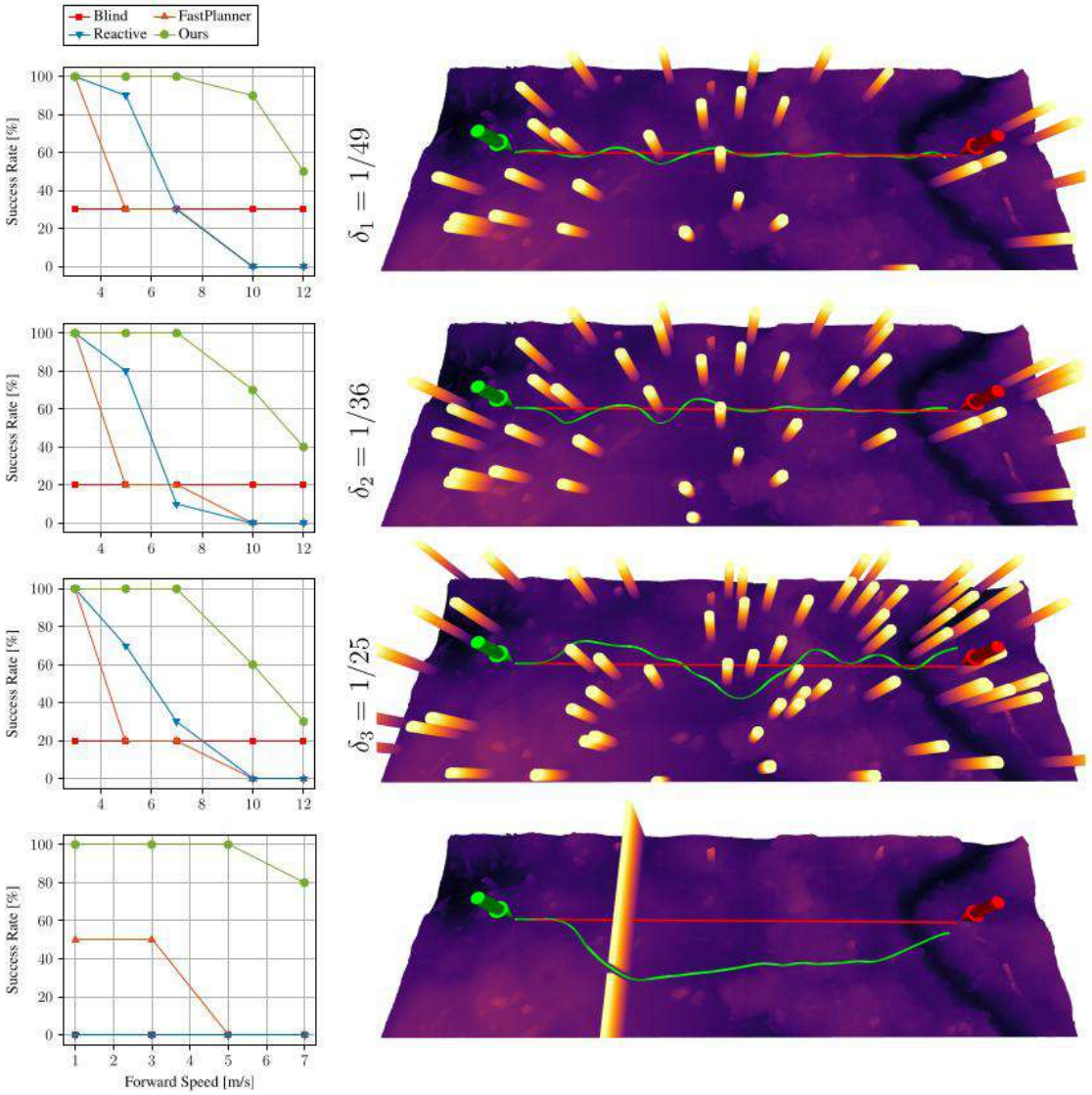


Figure G.4 – Simulation experiments. Top three rows illustrate experiments in the forest, while the bottom row depicts the narrow-gap experiment. Forest experiments are ordered by increasing difficulty, which is controlled by the tree density δ. The left column reports success rates at various speeds. The right column shows one of the random realizations of the environment together with paths taken by different policies from start (green arrow) to end (red arrow). The paths illustrate the blind policy (red) and the path taken by our approach (green). Our approach consistently outperforms the baselines in all environments and and at all speeds.

domly placed according to a homogeneous Poisson point process P with intensity δ treem^{-2} [135]. Note that Tomppo et al. [283] found that around 30% of the forests in Finland could be considered as a realization of a spatial Poisson point process. We control the task difficulty by changing the tree density δ. We set $w = 30\,\mathrm{m}$ and $l = 60\,\mathrm{m}$ and start the drone at position $s = (-\frac{l}{2}, -\frac{w}{2})$ (the origin of the coordinate system is at the center of $R(l,w)$). We provide the drone with a straight reference trajectory of 40 m length. We test on three different tree densities with increasing difficulty: $\delta_1 = \frac{1}{49}$ (*low*), $\delta_2 = \frac{1}{36}$ (*medium*), and $\delta_3 = \frac{1}{25}$ (*high*) treem^{-2} . We vary the average forward speed of the drone between $3\,\mathrm{m\,s^{-1}}$ and $12\,\mathrm{m\,s^{-1}}$. We repeat the experiments with 10 different random realizations of the forest for each difficulty, using the same random seed for all baselines.

The first three rows in Figure G.4 show the results of this experiment, together with one example environment for each difficulty. At a low speed of $3\,\mathrm{m\,s^{-1}}$, all methods successfully complete every run even for high difficulties. As speed increases, the success rates of the baselines quickly degrade. At $10\,\mathrm{m\,s^{-1}}$, no baseline completes even a single run successfully, irrespective of task difficulty. In contrast, our approach is significantly more robust at higher speeds. It achieves 100% success rate up to $5\,\mathrm{m\,s^{-1}}$. For speeds of $10\,\mathrm{m\,s^{-1}}$, our approach has a success rate of 90% in the low difficulty task and 60% in the high difficulty task. Moreover, we show that our approach can go as fast as $12\,\mathrm{m\,s^{-1}}$ with a success rate of up to 50%.

We identify two reasons for the drop in performance of the baselines at higher speeds. The first reason is latency. The baselines follow a modular approach which first builds a map and then finds a collision-free trajectory in it. This process is not fast enough to avoid collisions at high speeds. By contrast, our approach has low latency between sensing and action by design. The second reason for the performance drop is noise in the depth observations. High-speed motion results in little overlap between consecutive observations, which is challenging to existing map building approaches that require multiple observations to cope with noise. On the other hand, purely local point clouds are too noisy to find a single collision-free trajectory. Our data-driven approach allows leveraging regularities in the data, which makes it more robust to sensor noise. We show additional controlled studies on latency and sensor noise in Section G.2.3 and Section G.2.4.

Narrow gap. In the second set of experiments, we mimic the real-world narrow gap experiment. We render a 40 m long wall with a single opening, 10 m in front of the drone. The gap is placed at a randomized lateral offset in the range of $[-5, 5]\mathrm{m}$ with respect to the starting location. The width of the opening is also randomized uniformly between 0.8 m and 1.0 m. All experiments are repeated 10 times with different opening sizes and lateral gap offsets for each speed. An illustration of the setup and the results of the evaluation are shown in the last row of Figure G.4. The reactive and blind baselines consistently fail to solve this task, while FastPlanner has a success rate of up to 50% at $5\,\mathrm{m\,s^{-1}}$, but still consistently fails at $7\,\mathrm{m\,s^{-1}}$. We observe that the baselines adapt their trajectory only when being close to the wall, which is often too late to correct course towards the opening. Conversely, our approach always completes the task successfully at speeds of up to $5\,\mathrm{m\,s^{-1}}$. Even at a speed of $7\,\mathrm{m\,s^{-1}}$ it only fails in 2 out of 10 runs. The ability of our approach to combine long-term and short-term planning is of crucial importance to achieve

Method		μ [ms]	σ [ms]	Perc. [%]	Total Time [ms]
FastPlanner [311]	Sensing	14.6	2.3	22.3	65.2
	Mapping	49.2	8.7	75.5	
	Planning	1.4	1.6	2.2	
Reactive [70]	Sensing	13.8	1.3	72.3	19.1
	Planning	5.3	0.9	27.7	
Ours	Sensing	0.1	0.04	3.9	10.28 (2.58*)
	NN inference	10.1 (2.4*)	1.5 (0.8*)	93.0	
	Projection	0.08	0.01	3.1	
Ours (Onboard)	Sensing	0.2	0.1	0.4	41.6
	NN inference	38.9	4.5	93.6	
	Projection	2.5	1.5	6.0	

Table G.1 – Latency (μ) on a desktop computer equipped with a hexacore i7-8700 CPU and a GeForce RTX 2080 GPU. The standard deviation σ is computed over 1000 samples. For our approach, we report the computation time on the CPU and GPU (marked with *) on the desktop computer (*Ours*), as well as the computation time on the onboard computation unit Jetson TX2, (*Ours Onboard*). For the FastPlanner [311] and Reactive [70] baselines sensing represents the time to build a pointcloud from the depth image, while for our method sensing is the time to convert depth into an input tensor for the neural network. More details on the subtasks division are available in the Appendix in Section S2. The proposed methodology is significantly faster than prior works.

this performance, as it is necessary to steer the drone early enough towards the opening and at the same time perform small reactive corrections to avoid a collision. This ability, in addition to the low latency and robustness to sensor noise, gives our approach a significant performance advantage with respect to the baselines.

G.2.3 Computational Cost

In this section, we compare the computational cost of our algorithm with the baselines. Table G.1 shows the results of this evaluation. It highlights how each step of the methods contributes to the overall computation time. All timings were recorded on a desktop computer with a 6-core i7-8700 CPU, which was also used to run the simulation experiments. To ensure a fair comparison, we report the timings when using only the CPU for all approaches. We also report the timings of our approach when performing neural network inference on a GeForce RTX 2080 GPU, as accelerators can be used with our approach without any extra effort. To paint a complete and realistic picture, we additionally evaluate the timing of our algorithm on the onboard computer of the quadrotor which is a Jetson TX2.

With a total computation time of 65.2 ms per frame, FastPlanner incurs the highest latency. It is important to note that the temporal filtering operations that are necessary to cope with sensing errors effectively make perception even slower. Two to three observations of an obstacle can be required to add it to the map, which increases the effective latency of the system. By

foregoing the mapping stage altogether, the *Reactive* baseline significantly reduces computation time. This baseline is approximately three times faster than *FastPlanner*, with a total latency of 19.1 ms. However, the reduced latency comes at the cost of the trajectory complexity that can be represented, since the planner can only select primitives from a pre-defined library. In addition, the reactive baseline is sensitive to sensing errors, which can drastically affect performance at high speeds.

Our approach has significantly lower latency than both baselines; when network inference is performed on the GPU, our approach is 25.3 times faster than *FastPlanner* and 7.4 times faster than the *Reactive* baseline. When GPU inference is disabled, the network's latency increases by only 8 ms, and our approach is still much faster than both baselines. Moving from the desktop computer to the onboard embedded computing device, the network's forward pass requires 38.9 ms. Onboard, the total time to pass from the sensor reading to a plan is 41.6 ms, which corresponds to an update rate of about 24 Hz.

G.2.4 The Effect of Latency and Sensor Noise

We analyze the effect of sensor noise and planning latency in a controlled experiment. In this experiment, the quadrotor is traveling along a straight line at a constant forward speed and is required to laterally evade a single obstacle (a pole) while having only limited sensing range. This experimental setup was proposed in Falanga et al. [64] to understand the role of perception on the navigation ability of a robotic system subject to bounded inputs. Specifically, the authors derived an upper-bound for the forward speed at which a robot can fly and avoid a single obstacle as a function of latency and sensing range. They modeled the robot as a point-mass, which is a limited approximation for a quadrotor as it neglects the platform's rotational dynamics. We thus extend their formulation to account for the latency introduced by the rotational motion that is necessary to avoid the obstacle. A detailed description of the formulation can be found in the Appendix in Section S3.

We set up the experiment by spawning a quadrotor with an initial forward velocity v at a distance of 6 m from a pole with diameter 0.6 m. According to our formulation, we compute a theoretical maximum speed – *i.e.* the speed at which the task is not feasible anymore – of $v_{max} = 13\,\mathrm{m\,s^{-1}}$. We then perform the controlled experiment with varying forward speeds v in the range $3 - 13\mathrm{m\,s^{-1}}$. We perform 10 experiments for each speed with all approaches and report the success rate. We run the experiment in two settings: (i) with ground-truth depth information, to isolate the effect of latency on performance, and (ii) with depth estimated by stereo matching [114] to analyze the combined effect of latency and sensing errors.

Ground-truth depth. Figure G.5-B1 illustrates the results of this experiment when perfect depth perception (Figure G.5-A1) is available. All approaches can complete the task perfectly up to $5\,\mathrm{m\,s^{-1}}$. However, even in these ideal conditions, the performance of the baselines drops for speeds beyond $5\,\mathrm{m\,s^{-1}}$. This drop in performance for *Reactive* can be attributed to the

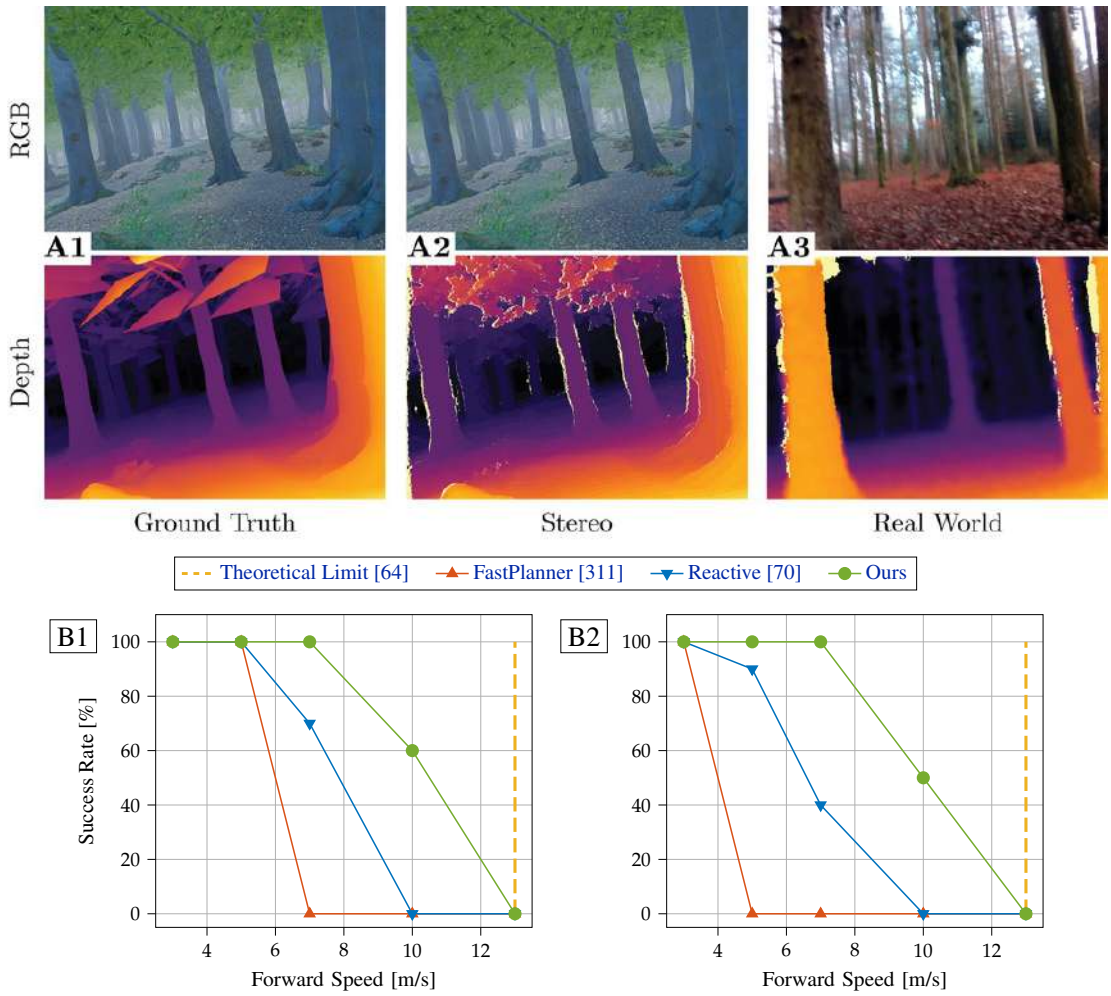


Figure G.5 – The effect of sensor noise on performance. (**A**) When comparing RGB and depth images generated in simulation with images captured in the real world, we observe that the corresponding depth images are more similar than the RGB images. In addition, simulated depth estimated by stereo matching (**A2**) contains the typical failure cases of a depth sensor (**A3**), *e.g.* missing values and noise. (**B**) Results of the controlled experiment to study the relationship between perception latency and navigation ability. The experiment is performed on ground-truth depth (**B1**) and stereo depth (**B2**). Our approach can fly closer to the theoretical limit than the baselines and is only minimally affected by the noise of stereo depth estimates.

fact that the finite library of motion primitives does not contain maneuvers that are aggressive enough to complete this task. Similarly, the performance degrades for *FastPlanner* as a result of sub-optimal planning of actions. Even though this baseline manages to map the obstacle in time, the planner frequently commands actions to stop the platform which leads to crashes when flying at high speeds. It is important to point out that the *FastPlanner* baseline was only demonstrated up to speeds of $3\,\mathrm{m\,s^{-1}}$ in the original work [311], and thus was not designed to operate at high speeds. Our approach can successfully avoid the obstacle without a single failure up to $7\,\mathrm{m\,s^{-1}}$. For higher speeds, performance gracefully degrades to 60% at $10\,\mathrm{m\,s^{-1}}$. This decrease in performance can be attributed to the sensitivity to imperfect network predictions when flying at high speed, where a single wrong action can lead to a crash.

Estimated depth. While the previous experiments mainly focused on latency and the avoidance strategy, we now study the influence of imperfect sensory measurements on performance. We repeat the same experiment, but provide all methods with depth maps that have been computed from the stereo pairs (Figure G.5-A2). Figure G.5-B2 shows the results of this experiment. The baselines experience a significant drop in performance compared to when provided with perfect sensory readings. *FastPlanner* completely fails for speeds of $5\,\mathrm{m\,s^{-1}}$ and beyond. This sharp drop in performance is due to the need for additional filtering of the noisy depth measurements that drastically increases the latency of the mapping stage. As a result, this baseline detects obstacles too late to be able to successfully evade them. Similarly, the performance of the *Reactive* baseline drops by 30% at $7\,\mathrm{m\,s^{-1}}$. In contrast to the baselines, our approach is only marginally affected by the noisy depth readings, with only a 10% drop in performance at $10\,\mathrm{m\,s^{-1}}$, but no change in performance at lower speeds. This is because our policy, trained on depth from stereo, learns to account for common issues in the data such as discretization artifacts and missing values.

G.3 Discussion

Existing autonomous flight systems are highly engineered and modular. The navigation task is usually split into sensing, mapping, planning, and control. The separation into multiple modules simplifies the implementation of engineered systems, enables parallel development of each component, and makes the overall system more interpretable. However, modularity comes at a high cost: the communication between modules introduces latency, errors compound across modules, and interactions between modules are not modeled. Additionally, it is an open question if certain subtasks, such as maintaining an explicit map of the environment, are even necessary for agile flight.

Our work replaces the traditional components of sensing, mapping, and planning with a single function that is represented by a neural network. This drastically reduces the latency of the system and increases its robustness against sensor noise. We demonstrate that our approach can reach speeds of up to $10\,\mathrm{m\,s^{-1}}$ in complex environments and reduces the failure rate at high speeds by up to 10 times when compared to the state of the art.

We achieve this by training a neural network to imitate an expert with privileged information in simulation. To cope with the complexity of the task and to enable seamless transfer from simulation to reality, we make several technical contributions. These include a sampling-based expert, a novel neural network architecture, and a training procedure, all of which take the task's multi-modality into account. We also use an abstract, but sufficiently rich input representation that considers real-world sensor noise. The combination of these innovations enables the training of robust navigation policies in simulation that can be directly transferred to diverse real-world environments without any fine-tuning on real data.

We see several opportunities for future work. Currently, the learned policy exhibits low success rates at average speeds of $10\,\mathrm{m\,s^{-1}}$ or higher. At these speeds even our expert policy, despite having perfect knowledge of the environment, often fails to find collision-free trajectories. This is mainly due to the fact that, at speeds of $10\,\mathrm{m\,s^{-1}}$ or higher, feasible solutions require temporal consistency over a long time horizon and strong variations of the instantaneous flying speed as a function of the obstacle density. This requirement makes feasible trajectories extremely sparse in parameter space, resulting in intractable sampling. Engineering a more complex expert to tackle this problem can be very challenging and might require specifically tailored heuristics to find approximate solutions. Therefore, we believe this problem represents a big opportunity for model-free methods, which have the potential to ease the engineering requirements. The second reason for the performance drop at very high speeds is the mismatch between the simulated and physical drone in terms of dynamics and perception. The mismatch in dynamics is due to aerodynamics effects, motor delays, and dropping battery voltage. Therefore, we hypothesise that performance would benefit from increasing the fidelity of the simulated drone and making the policy robust to the unavoidable model mismatches. A third reason is perception latency. This could be further reduced with event cameras [82], especially in presence of dynamic obstacles [65].

Overall, our approach is a stepping stone towards the development of autonomous systems that can navigate at high speeds through previously unseen environments with only on-board sensing and computation. Combining our short-horizon controller for local obstacle avoidance with a long-term planner is a major opportunity for many robotics applications, including autonomous exploration, delivery, and cinematography.

G.4 Materials and Methods

To perform agile flight through cluttered and previously-unseen environments, we train a sensorimotor policy to predict receding-horizon trajectories from on-board sensor measurements and a reference trajectory. We assume that the reference trajectory is provided by a higher-level planning algorithm or the user. The reference is not necessarily collision-free and only encodes the long-term goal of the platform. The agent is responsible to fly in the direction dictated by the reference while adapting its flight path to the environment.

An agent's observation $\boldsymbol{o}$ consists of a depth image $\boldsymbol{d} \in \mathbb{R}^{640\times480}$, an estimate of the platform

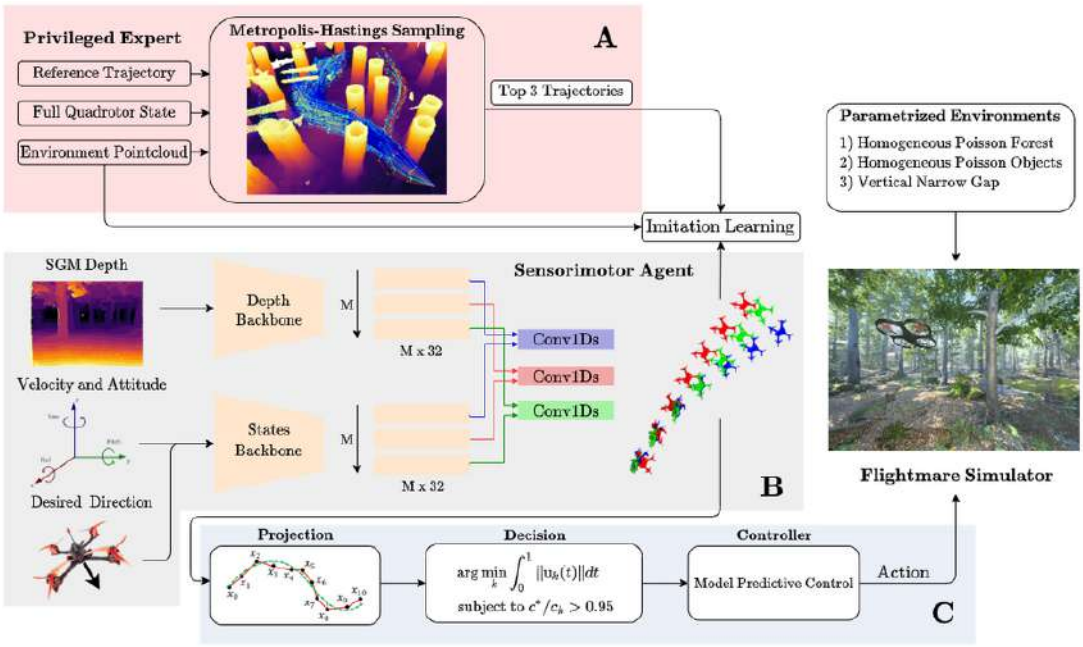


Figure G.6 – Method overview. (**A**) Our offline planning algorithm computes a distribution of collision-free trajectories to follow a reference trajectory. The trajectories are computed with Metropolis-Hastings sampling and are conditioned on complete 3D knowledge of the environment, which is represented by a point cloud. (**B**) A sensorimotor agent is trained with imitation learning to predict the best three trajectories from the estimated depth, the drone's velocity and attitude, and the desired direction that encodes the goal. (**C**) The predictions are projected on the space of polynomial trajectories and ranked according to their predicted collision cost c_k. The trajectory with the lowest predicted cost c_k is then tracked with a model predictive controller. If multiple trajectories have similar predicted cost (within a 5% range of the minimum $c^* = \min c_k$), the one with the smallest actuation cost is used.

velocity $\boldsymbol{v} \in \mathbb{R}^3$ and attitude (expressed as a rotation matrix) $\boldsymbol{q} \in \mathbb{R}^9$, and a desired flight direction $\boldsymbol{\omega} \in \mathbb{R}^3$. The policy output is a set of motion hypotheses which are represented as receding-horizon trajectories with the corresponding estimated risk of collision. A model predictive controller then tracks the trajectory with the lowest collision probability and input cost. The controller is trained using privileged learning [37]. Specifically, the policy is trained on demonstrations provided by a sampling-based planner that has access to information that is not available to the sensorimotor student: the precise knowledge of the 3D structure of the environment and the exact platform state. Figure G.6 shows an overview of our method.

We collect demonstrations and perform training entirely in simulation. This trivially facilitates access to perfect 3D and state data for the privileged expert, enables the synthesis of unlimited training samples for any desired trajectory, and does not put the physical platform in danger. We use the Flightmare simulator [266] with the RotorS [78] Gazebo plugin and Unity as a rendering engine [130]. Both the training and the testing environments are built by adding obstacles to the uneven ground of an out-of-the-box Unity environment[4].

[4] https://assetstore.unity.com/packages/3d/vegetation/forest-environment-dynamic-nature-150668

To enable zero-shot transfer to real environments, special care has to be taken to minimize the domain shift of the (visual) input modalities from simulation to reality. To this end we use depth as an abstract input representation that shows only a negligible domain shift from simulation to the real-world (cf. Figure G.5-A). Specifically, the sensorimotor agent is trained with depth images that have been computed using Semi-Global Matching (SGM) [114] from a simulated stereo camera pair. As off-the-shelf depth sensors, such as the Intel RealSense 435 camera used on our physical platform, compute depth from stereo images using similar principles [142], this strategy ensures that the characteristics of the input aligns between simulation and the real world. As a result, the trained sensorimotor agent can be directly deployed in the real world. The next section presents both the privileged expert as well as the sensorimotor agent in detail.

G.4.1 The Privileged Expert

Our privileged expert is a novel sampling-based motion planning algorithm. The expert has perfect knowledge of the platform state and the environment (complete 3D map), both of which are only available in simulation. The expert generates a set of collision-free trajectories $\boldsymbol{\tau}$ representing the desired state of the quadrotor $\boldsymbol{x}_{des} \in \mathbb{R}^{13}$ over the next second, starting from the current state of the drone, i.e. $\boldsymbol{\tau}(0) = \boldsymbol{x}$. To do so, it samples from a probability distribution P that encodes distance from obstacles and proximity to the reference trajectory. Specifically, the distribution of collision-free trajectories $P(\boldsymbol{\tau} \mid \boldsymbol{\tau}_{\text{ref}}, \mathcal{C})$ is conditioned on the reference trajectory $\boldsymbol{\tau}_{\text{ref}}$ and the structure of the environment in the form of a point cloud $\mathcal{C} \in \mathbb{R}^{n\times 3}$. According to P, the probability of a trajectory $\boldsymbol{\tau}$ is large if far from obstacles and close to the reference $\boldsymbol{\tau}_{\text{ref}}$. We define P as the following:

$$P(\boldsymbol{\tau} \mid \boldsymbol{\tau}_{\text{ref}}, \mathcal{C}) = \frac{1}{Z} \exp(-c(\boldsymbol{\tau}, \boldsymbol{\tau}_{\text{ref}}, \mathcal{C})) \tag{G.1}$$

where $Z = \int_{\boldsymbol{\tau}} P(\boldsymbol{\tau} \mid \boldsymbol{\tau}_{\text{ref}}, \mathcal{C})$ is the normalization factor and $c(\boldsymbol{\tau}, \boldsymbol{\tau}_{\text{ref}}, \mathcal{C}) \in \mathbb{R}_+$ is a cost function indicating proximity to the reference and distance from obstacles. We define the trajectory cost function as

$$c(\boldsymbol{\tau}, \boldsymbol{\tau}_{\text{ref}}, \mathcal{C}) = \int_0^1 \lambda_c C_{collision}(\boldsymbol{\tau}(t)) + [\boldsymbol{\tau}(t) - \boldsymbol{\tau}_{ref}(t)]^\top \boldsymbol{Q} [\boldsymbol{\tau}(t) - \boldsymbol{\tau}_{ref}(t)]\, dt \tag{G.2}$$

where $\lambda_c = 1000$, $\boldsymbol{Q}$ is a positive semidefinite state cost matrix, and $C_{collision}$ is a measure of the distance of the quadrotor to the points in $\mathcal{C}$. We model the quadrotor as a sphere of radius $r_q = 0.2\,\text{m}$ and define the collision cost as a truncated quadratic function of d_c, *i.e.* the distance between the quadrotor and the closest point in the environment:

$$C_{collision}(\boldsymbol{\tau}(t)) = \begin{cases} 0 & \text{if } d_c > 2r_q \\ -d_c^2 / r_q^2 + 4 & \text{otherwise.} \end{cases} \tag{G.3}$$

The distribution P is complex due to the presence of arbitrary obstacles and frequently multi-modal in cluttered environments since obstacles can be avoided in multiple ways. Therefore, the

analytical computation of P is generally intractable.

To approximate the density P, the expert uses random sampling. We generate samples with the Metropolis-Hastings (M-H) algorithm [107] as it provides asymptotic convergence guarantees to the true distribution. To estimate P, the M-H algorithm requires a target score function $s(\boldsymbol{\tau}) \propto P(\boldsymbol{\tau} \mid \boldsymbol{\tau}_{\text{ref}}, \mathcal{C})$. We define $s(\boldsymbol{\tau}) = \exp(-c(\boldsymbol{\tau}, \boldsymbol{\tau}_{\text{ref}}, \mathcal{C}))$, where $c(\cdot)$ is the cost of the trajectory $\boldsymbol{\tau}$. It is easy to show that this definition satisfies the conditions for the M-H algorithm to asymptotically estimate the target distribution P. Hence, the trajectories sampled with M-H will asymptotically cover all of the different modes of P. We point the interested reader to the Appendix, Section S4, for an overview of the M-H algorithm and its convergence criteria.

To decrease the dimension of the sampling space, we use a compact yet expressive representation of the trajectories $\boldsymbol{\tau}$. We represent $\boldsymbol{\tau}$ as a cubic B-spline $\boldsymbol{\tau}_{bspline} \in \mathbb{R}^{3\times 3}$ curve with 3 control points and a uniform knot vector, enabling interpolation with high computational efficiency [83]. Cubic B-Splines are twice continuously differentiable and have a bounded derivative in a closed interval. Due to the differential flatness property of quadrotors [190], continuous and bounded acceleration directly translates to continuous attitude over the trajectory duration. This encourages dynamically feasible trajectories that can be tracked by a model-predictive controller accurately [190, 67]. Therefore, instead of naively sampling the states of $\boldsymbol{\tau}$, we vary the shape of the trajectory by sampling the control points of the B-spline in a spherical coordinate system. In addition, for computational reasons, we discretize the trajectory at equally spaced time intervals of 0.1 s and evaluate the discrete version of Equation (G.2). Specifically, we sample a total of 50K trajectories using a Gaussian with a variance of $2, 5, 10$ that increases every 16K samples as the proposal distribution. In spite of this efficient representation, the privileged expert cannot run in real-time, given the large computational overhead introduced by sampling.

To bias the sampled trajectories towards obstacle-free regions, we replace the raw reference trajectory $\boldsymbol{\tau}_{ref}$ in Equation (G.2) with a global collision-free trajectory $\boldsymbol{\tau}_{gbl}$ from start to goal, that we compute using the approach of Liu et al. [168]. As illustrated in Figure S3, conditioning sampling on $\boldsymbol{\tau}_{gbl}$ practically increases the horizon of the expert and generates more conservative trajectories. An animation explaining our expert is available in Movie S1. After removing all generated trajectories in collision with obstacles, we select the three best trajectories with lower costs. Those trajectories are used to train the student policy.

G.4.2 The Student Policy

In contrast to the privileged expert, the student policy produces collision-free trajectories in real time with access only to on-board sensor measurements. These measurements include a depth image estimated with semi-global matching (SGM) [114], the platform velocity and attitude, and the desired direction of flight. The latter is represented as a normalized vector heading towards the reference point one second in the future with respect to the closest reference state. We hypothesize that this information is sufficient for generating the highest probability samples

of the distribution $P(\boldsymbol{\tau} \mid \boldsymbol{\tau}_{\text{ref}}, \mathcal{C})$ without actually having access to the point cloud $\mathcal{C}$. There are two main challenges to accomplishing this: (i) the environment is only partially observable from noisy sensor observations, and (ii) the distribution P is in general multi-modal. Multiple motion hypotheses with high probabilities can be available, but their average can have a very low probability.

We represent the policy as a neural network that is designed to be able to cope with these issues. The network consists of an architecture with two branches that produce a latent encoding of visual, inertial, and reference information, and outputs $M = 3$ trajectories and their respective collision cost. We use a pre-trained MobileNet-V3 architecture [117] to efficiently extract features from the depth image. The features are then processed by a 1D convolution to generate M feature vectors of size 32. The current platform's velocity and attitude are then concatenated with the desired reference direction and processed by a 4-layer perceptron with $[64, 32, 32, 32]$ hidden nodes and LeakyReLU activations. We again use 1D convolutions to create a 32-dimensional feature vector for each mode. The visual and state features are then concatenated and processed independently for each mode by another 4-layer perceptron with $[64, 128, 128]$ hidden nodes and LeakyReLU activations. The latter predicts, for each mode, a trajectory $\boldsymbol{\tau}$ and its collision cost. In summary, our architecture receives as input a depth image $\boldsymbol{d} \in \mathbb{R}^{640\times480}$, the platform's velocity $\boldsymbol{v} \in \mathbb{R}^3$, the drone's attitude expressed as a rotation matrix $\boldsymbol{q} \in \mathbb{R}^9$, and reference direction $\boldsymbol{\omega} \in \mathbb{R}^3$. From this input it predicts a set $\mathcal{T}_n$ of trajectories and their relative collision cost, *i.e.* $\mathcal{T}_n = \{(\boldsymbol{\tau}_n^k, c_k) \mid k \in [0, 1, \dots, M-1]\}$, where $c_k \in \mathbb{R}_+$. Differently from the privileged expert, the trajectory predicted by the network does not describe the full state evolution but only its position component, i.e. $\boldsymbol{\tau}_n^k \in \mathbb{R}^{10\times3}$. Specifically, the network trajectory $\boldsymbol{\tau}_n^k$ is described by:

$$\boldsymbol{\tau}_n^k = [\ \boldsymbol{p}(t_i)\ ]_{i=1}^{10}, \quad t_i = \frac{i}{10}, \tag{G.4}$$

where $\boldsymbol{p}(t_i) \in \mathbb{R}^3$ is the drone's position at time $t = t_i$ relative to its current state $\boldsymbol{x}$. This representation is more general than the B-spline with 3 control points used by the sampling-based planner, and it was preferred to the latter representation to avoid the computational costs of interpolation at test time.

We train the neural network with supervised learning on the 3 trajectories with lowest cost found by the expert. To account for the multi-hypotheses prediction, we minimize the following *Relaxed Winner-Takes-All* (R-WTA) loss for each sample:

$$\text{R-WTA}(\mathcal{T}_e, \mathcal{T}_n) = \sum_{i=0}^{|\mathcal{T}_e|} \sum_{k=0}^{|\mathcal{T}_n|} \alpha(\boldsymbol{\tau}_{e,p}^i, \boldsymbol{\tau}_n^k) \|\boldsymbol{\tau}_{e,p}^i - \boldsymbol{\tau}_n^k\|^2, \tag{G.5}$$

where $\mathcal{T}_e$ and $\mathcal{T}_n$ are the set of expert and network trajectories, $\boldsymbol{\tau}_{e,p}$ denotes the position component of $\boldsymbol{\tau}_e$, and $\alpha(\cdot)$ is defined as

$$\alpha(\boldsymbol{\tau}_{e,p}^i, \boldsymbol{\tau}_n^k) = \begin{cases} 1-\epsilon & \text{if } \|\boldsymbol{\tau}_{e,p}^i - \boldsymbol{\tau}_n^k\|^2 \leq \|\boldsymbol{\tau}_{e,p}^j - \boldsymbol{\tau}_n^k\|^2 \quad \forall j \neq i \\ \frac{\epsilon}{M-1} & \text{otherwise.} \end{cases} \tag{G.6}$$

Intuitively, an expert trajectory $\boldsymbol{\tau}_e^i \in \mathcal{T}_e$ is associated with the closest network trajectory $\boldsymbol{\tau}_n^k \in \mathcal{T}_n$ with a weight of $1-\epsilon = 0.95$ and with $\epsilon/(M-1) = 0.025$ to all remaining hypotheses. This formulation, proposed by Rupprecht et al. [242], prevents mode collapse and was shown to outperform other popular approaches for multi-modal learning such as Mixture Density Networks [18]. In addition, the predicted collision cost c_k is trained with supervised learning on the ground-truth cost $C_{collision}(\boldsymbol{\tau}_n^k)$ computed according to Equation (G.3). In summary, the final training loss for each sample is equal to:

$$\mathcal{L}((\mathcal{T}_e, \mathcal{T}_n) = \lambda_1 \text{R-WTA}(\mathcal{T}_e, \mathcal{T}_n) + \lambda_2 \sum_{k=0}^{|\mathcal{T}_n|} \|c_k - C_{collision}(\boldsymbol{\tau}_n^k)\|^2, \tag{G.7}$$

where $\lambda_1 = 10$ and $\lambda_2 = 0.1$ were empirically found to equalize the magnitudes of the two terms. This loss is averaged over a minibatch of 8 samples and minimized with the Adam optimizer [148] and a learning rate of 1e$-$3.

At test time we retrieve the full state information from the predicted trajectories by projecting them on the space of order-5 polynomials for each axis independently. This representation enforces continuity in position, velocity, and acceleration with respect to the current state, and facilitates dynamic feasibility due to differential flatness [190]. Considering for example the x axis, we define the polynomial projection $\mu_x(t) = \boldsymbol{a}_x^\top \cdot \mathbf{T}(t)$, where $\boldsymbol{a}_x^\top = [a_0, a_1, \ldots, a_5]$ and $\mathbf{T}(t)^\top = \left[1, t, \ldots, t^5\right]$. The projection term $\boldsymbol{a}_x$ is found by solving the following optimization problem:

$$\begin{aligned} \underset{\boldsymbol{a}_x}{\text{minimize}} \quad & \sum_{i=1}^{10} \left(\boldsymbol{\tau}_{n,x}^{k,i} - \boldsymbol{a}_x^\top \cdot \mathbf{T}(\frac{i}{10}) \right)^2 \\ \text{subject to} \quad & s_x(0) - \boldsymbol{a}_x^\top \cdot \mathbf{T}(0) = 0 \\ & \dot{s}_x(0) - \boldsymbol{a}_x^\top \cdot \dot{\mathbf{T}}(0) = 0 \\ & \ddot{s}_x(0) - \boldsymbol{a}_x^\top \cdot \ddot{\mathbf{T}}(0) = 0 \end{aligned} \tag{G.8}$$

where $\boldsymbol{\tau}_{n,x}^{k,i}$ is the x component of the i^{th} element of $\boldsymbol{\tau}_n^k$ and $s_x(0), \dot{s}_x(0), \ddot{s}_x(0)$ are the x position of the quadrotor and its derivatives obtained from the current state estimate. The latter corresponds to the ground-truth state when deployed in simulation and to the state estimate computed by the Intel RealSense T265 when deployed on the physical platform. To reduce abrupt changes in velocity during flight, which would lead to strong pitching motion, we additionally constrain the polynomial's average speed to a desired value v_{des}. To do so, we scale the time t of polynomial $\mu_x(t)$ by a factor $\beta = v_{des}/v_\mu^x$, *i.e.* $t' = \beta t$, where $v_\mu^x = \|\mu(1) - \mu(0)\|$.

Once all predicted trajectories are projected we select one for execution. To do so, we select trajectories with $c^*/c_k \geq 0.95$ ($c^* = \min c_k$) and compute their input costs according to Mellinger et al. [190]. The one with the lowest input costs is tracked by a model-predictive controller [67]. Intuitively, this choice enforces temporal continuity in the trajectories. For example, when dodging an obstacle to the right, we do not want to steer toward the left at the

next iteration, unless strictly necessary due to the appearance of a new obstacle.

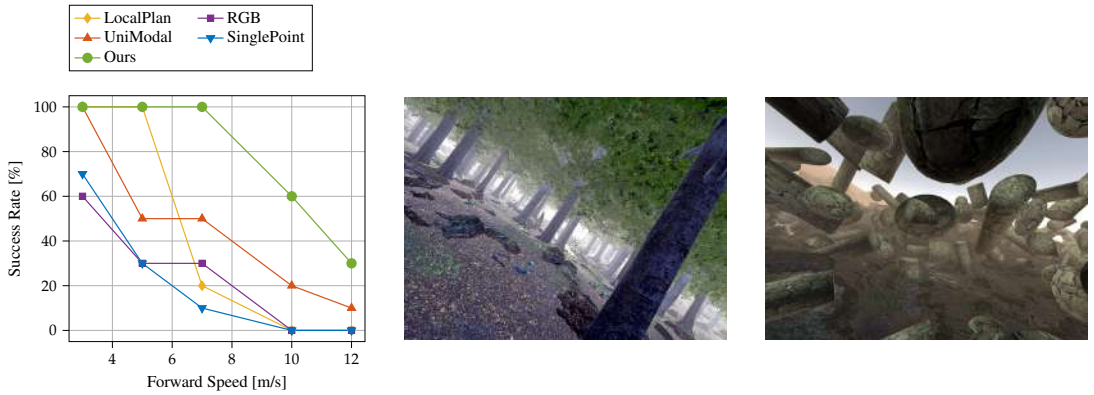
G.4.3 Training Environments

We build custom environments in the Flightmare simulator [266] to collect training data. All environments are built by spawning items on the uneven empty ground of an off-the-shelf Unity environment. We spawn items belonging to two categories: simulated trees, available off-the-shelf, and a set of convex shapes such as ellipsoids, cuboids, and cylinders (Figure G.7). The dimensions of these shapes are randomized according to a continuous uniform random distribution with $x \in \mathcal{U}(0.5, 4)$, $y \in \mathcal{U}(0.5, 4)$, and $z \in \mathcal{U}(0.5, 8)$. Training environments are created by spawning either of the two categories of items according to a homogeneous Poisson point process with intensity δ.

We generate a total of 850 environments by uniform randomization of the following two quantities: item category, *i.e.* trees or shapes, and the intensity $\delta \in \mathcal{U}(4, 7)$, with $\delta \in \mathbb{N}_+$. For each environment, we compute a global collision-free trajectory $\boldsymbol{\tau}_{gbl}$ from the starting location to a point 40 m in front of it. The trajectory $\boldsymbol{\tau}_{gbl}$ is not observed by the student policy, but only by the expert. The student is only provided with a straight, potentially not collision-free, trajectory from start to end to convey the goal.

To assure sufficient coverage of the state space, we use the dataset aggregation strategy (DAgger) [239]. This process consists of rolling out the student policy and labeling the visited states with the expert policy. To avoid divergence from $\boldsymbol{\tau}_{gbl}$ and prevent crashes in the early stages of training, we track a trajectory predicted by the student only if the drone's distance from the closest point on $\boldsymbol{\tau}_{gbl}$ is smaller than a threshold ξ, initialized to zero. Otherwise, we directly track $\boldsymbol{\tau}_{gbl}$ with a model-predictive controller. Every 30 environments the student is re-trained on all available data and the threshold ξ set to $\xi' = \min(\xi + 0.25, 6)$. Aggregating data over all environments results in a dataset of approximately 90K samples. For the narrow-gap experiments, we finetune the student policy on 100 environments created by adding a 50 m long wall with a single vertical gap of random width $w_g \in \mathcal{U}(0.7, 1.2)$ meters in the center. In those environments, the drone starts at a 10 m distance from the wall with a randomized lateral offset $l \in \mathcal{U}(-5, 5)$ from the gap. We collect approximately 10K training samples from those environments.

We evaluate the trained policies in simulation on environments not seen during training but coming from the same distributions. The same policies are then used to control the physical platforms in real-world environments. When deployed in simulation, we estimate depth with SGM [114] from a simulated stereo pair and use the ground-truth state of the platform. Conversely, on the physical platform depth is estimated by an off-the-shelf Intel RealSense 435 and state estimation is performed by an Intel RealSense T265. More details on our experimental platform are available in Section S1.



(a) Ablation study (b) Poisson forest (c) Poisson convex shapes

Figure G.7 – (a) Method validation. Without initialization of the sampler on a global trajectory (*LocalPlan*), multi-modal training (*UniModal*), or training on SGM depth image (*RGB*), performance significantly drops. Predicting a single point in the future instead of a full trajectory (*SinglePoint*) has similar effects on performance. *Ours* consistently outperforms all the ablated versions of the system. (b-c) Training environments. We build training environments by spawning trees or convex shapes according to an homogeneous Poisson point process with varying intensities.

G.4.4 Method Validation

Our approach is based on several design choices that we validate in an ablation study. We ablate the following components: (i) the use of global planning to initialize the sampling of the privileged expert, (ii) the use of depth as an intermediate representation for action, (iii) multi-modal network prediction. The results in Figure G.7 show that all components are important and that some choices have a larger impact than others. Our study indicates that depth perception plays a fundamental rule for high-speed obstacle avoidance: when training on color images (*RGB*) performance drops significantly. This is inline with previous findings [312, 138] showing that intermediate image representations have a strong positive effect on performance when sufficiently informative for the task. Not accounting for multi-modal trajectory prediction (*UniModal*) is also detrimental for performance. This is because the l_2-loss pushes predicted trajectories toward the average of the expert trajectories, which is often in collision with obstacles. Not initializing the sampling of expert trajectories on a global plan (*LocalPlan*) is not important for low-speed flight, but plays an important role for success at higher speeds. Biasing the motion away from regions that are densely populated by obstacles is particularly important for high-speed flight where little slack is available for reacting to unexpected obstacles. In addition, we compare our output representation, *i.e.* a trajectory one second in the future, to the less complex output representation in prior work [176], *i.e.* a single waypoint in the future (*SinglePoint*). The results indicate that the limited representation power of the latter representation causes performance to drop significantly, especially at high speeds.

Supplementary Materials

S1 Experimental Platform

Figure S1 – Illustration of our experimental platform. The main computational unit is an NVIDIA Jetson TX2, whose GPU is used for neural network inference and CPU for the control stack. Sensing is performed by an Intel Realsense T265 for state estimation and an Intel Realsense D435 for depth estimation.

To validate our approach with real-world experiments, we designed a lightweight, but powerful, quadrotor platform. The main frame is an Armattan Chameleon 6 inches, equipped with Hobbywing XRotor 2306 motors and 5 inches, three-bladed propellers. The platform has a total weight of 890 grams and can produce a maximum thrust of approximately 40 N, which results in a thrust-to-weight ratio of 4.4. The weight and power of this platform is comparable to the ones used by professional pilots in drone racing competitions.

The platform's main computational unit is an NVIDIA Jetson TX2 accompanied by a ConnectTech Quasar carrier board. The GPU of the Jetson TX2 is used to run neural network inference and its CPU for the rest of our control framework. The output of this framework is a low-level control command including a collective thrust and angular rates to be achieved for flying. The desired commands are sent to a commercial flight controller running BetaFlight, which produces single-rotor commands that are fed to the 4-in-1 motor controller.

Our quadrotor is equipped with two off-the-shelf sensing units: an Intel RealSense T265 and an Intel RealSense D435i. Both have a stereo camera setup and an integrated IMU. The RealSense T265 runs a visual-inertial odometry pipeline to output the state estimation of the platform at 200 Hz, which we directly use without any additional processing. Our second sensing unit, the RealSense D435i, outputs a hardware-accelerated depth estimation pipeline on its stereo setup. The latter pipeline provides to our framework a dense depth in VGA resolution (640×480 px) at 30 Hz, which we use without further processing. The horizontal field of view of this observation is approximately 90°, which appeared to be sufficient for the task of obstacle avoidance. For experiments at speeds of $7\,\mathrm{m\,s^{-1}}$ and above, we tilt the depth sensor by 30° to assure that the camera would look forward during flight. This is indeed a typical camera setup for high-speed flight in drone racing competitions. To support the transfer from simulation to reality, we build a simulated stereo setup with the same characteristics of the RealSense D435, on which we run a

GPU implementation of SGM[5] [110] to estimate dense depth in simulation.

Our software stack is developed in both C++ and Python, and will be made publicly available upon acceptance. Specifically, we implement the trajectory prediction framework with Tensorflow in a Python node and the rest of our trajectory projection and tracking software in separate C++ nodes. The communication between different nodes is implemented with ROS. Specifically, the Tensorflow node predicts trajectories at 24.7 Hz on the physical platform, and the MPC generates commands in the form of collective thrust and body rates at 100 Hz to track those trajectories. The low-level controller, responsible for tracking desired body rates and collective thrust predicted by the MPC, runs at 2 kHz. The platform only receives a start and stop command from the base computer, and is therefore completely autonomous during flight.

S2 Computational Complexity

The baseline with the largest latency is FastPlanner. This baseline has three components: sensing, mapping and planning. Sensing includes transforming a depth image to a pointcloud after filtering. Mapping includes ray casting and Euclidean Signed Distance Field (ESDF) computation. Finally, planning includes finding the best trajectory to reach the goal while avoiding obstacles. The total time to perform all these operations is 65.2 ms. However, it is important to note that, while the depth filtering and the probabilistic ray casting process are necessary to remove sensing errors, those operations make the inclusions of obstacles in the local map slower. Practically, 2-3 observations are required to add an obstacle to the local map, therefore largely increasing the overall latency of the system.

Removing the mapping stage altogether, the *Reactive* baseline experiences significant gains in computation time. For this baseline, the sensing latency is the time between receiving a depth observation and generating a point cloud after filtering and outlier rejection. The planning latency consists of building a KD-Tree from the filtered point cloud, and selecting the best trajectory out of the available motion primitives according to a cost based on collision probability and proximity to the goal. This baseline is approximately three times faster than *FastPlanner*, with a total latency of 19.1 ms. However, the reduced latency comes at the cost of a lower trajectory-representation power, since the planner can only select a primitive from a pre-defined motion library. In addition, given the lack of temporal filtering, the reactive baseline is very sensitive to sensing errors, which can drastically affect performance at high speeds.

Our approach has significantly lower latency than both baselines: when network inference is performed on the GPU, our approach is 25.3 times faster than *FastPlanner* and 7.4 times faster than the *Reactive* baseline. When GPU inference is disabled, the network's latency increases by only 8 ms, and our approach is still significantly faster than both baselines. For our approach, we

[5] https://github.com/dhernandez0/sgm

break down computation into three operations: (i) sensing, which includes the time for recording an image and convert it to an input tensor, (ii) neural network inference, and (iii) projection, which is the time to project the prediction of the neural network into the space of dynamically feasible trajectories for the quadrotor. Moving from the desktop computer to the onboard embedded computing device, the network's forward pass requires 38.9 ms. Onboard, the total time to pass from sensor to action is then 41.6 ms, which is sufficient to update actions at up to 24.3 Hz.

S3 Rotational Dynamics

Extending [64] to the quadrotor platform, we approximate the maximum speed $v_{\max}$ that still allows successful avoidance of a vertical cylindrical obstacle of radius r_{obs}. The avoidance maneuver is described as a sequence of two motion primitives consisting of pure rolling and pure acceleration. Both primitives are executed with maximum motor inputs. Concretely, we treat the time required to reorient the quadrotor as additional latency in the system, which can be computed by

$$t_{\text{rot}} = \sqrt{\frac{2\phi J}{T_{\max}}}, \tag{9}$$

with J being the moment of inertia, $T_{\max}$ the maximum torque the quadrotor can produce, and ϕ the desired roll angle. As the roll angle becomes a decision variable itself in this setting, we identify the best roll angle by maximizing the speed that still allows successful avoidance. Assuming that the quadrotor oriented at a roll angle ϕ accelerates with full thrust, its lateral position p_{lat} can be described by

$$p_{\text{lat}}(t) = \frac{1}{2} \sin\phi \cdot c_{\max} \cdot t^2 \,, \tag{10}$$

where $c_{\max}$ denotes the maximum mass-normalized thrust of the platform. Solving this equation for t and setting $p_{\text{lat}} = r_{\text{obs}}$ allows to formulate a maximum linear speed that still allows successful avoidance when considering the sensing range s, the sensing latency t_{s}, and the latency introduced to reorient the platform t_{rot}:

$$v_{\max} = \frac{s}{t_{\text{s}} + t_{\text{rot}} + \sqrt{\frac{2r_{\text{obs}}}{\sin\phi \cdot c_{\max}}}} \,. \tag{11}$$

Inserting (9) into (11) we can identifying ϕ that maximizes $v_{\max}$. In our study case, we set $r_{\text{obs}} = 0.5\,\text{m}$, which is the sum of the radius of the pole to avoid and the size of the drone. In addition, we set $c_{\max} = 33\,\text{m}\,\text{s}^{-2}$, $J = 0.007\,\text{kg}\,\text{m}^{-2}$, and $t_{\text{s}} = 66\,\text{ms}$, since images are rendered at 15 Hz. Using these values, we derive a rotational latency $t_{\text{rot}} = 119\,\text{ms}$ and a rotation angle $\phi = 69.1°$. Therefore, the total latency due to perception and the system's rotational inertia

is 185 ms. This latency and the sensing range of the depth camera s can be used to compute $v_{\max}$ according to Equation 11. Note that this approximation does not account for the fact that the quadrotor platform already performs some lateral acceleration during the rotation phase.

S4 Metropolis-Hastings Sampling

In statistics, the Metropolis-Hastings (M-H) algorithm [107] is used to sample a distribution $P(w)$ which can't be directly accessed. To generate the samples, the M-H algorithm requires a score function $d(w)$ proportional to $P(w)$. Requiring $d(w) \propto P(w)$ waives the need to find the normalization factor $Z = \int_w d(w)$ such that $P(w) = \frac{1}{Z} d(w)$, which can't be easily calculated for high-dimensional spaces. Metropolis-Hastings is a Markov Chain Monte Carlo sampling method [6]. Therefore, it generates samples by constructing a Markov chain that has the desired distribution as its equilibrium distribution. In this Markov chain, the next sample w_{t+1} comes from a distribution $t(w_{t+1}|w_t)$, referred to as transition model, which only depends on the current sample w_t. The transition model $t(w_{t+1}|w_t)$ is generally a pre-defined parametric distribution, e.g. a Gaussian. The next sample w_{t+1} is then accepted and used for the next iteration, or it is rejected, discarded, and the current sample w_t is re-used. Specifically, the sample is accepted with probability equal to

$$\alpha = \min\left(1, \frac{d(w_{t+1})}{d(w_t)}\right) = \min\left(1, \frac{P(w_{t+1})}{P(w_t)}\right). \tag{12}$$

Therefore, M-H always accepts a sample with a higher score than its predecessor. However, the move to a sample with a smaller score will sometimes be rejected, and the higher the drop in score $\frac{1}{\alpha}$, the smaller the probability of acceptance. Therefore, many samples come from the high-density regions of $P(w)$, while relatively few from the low-density regions. Roberts et. al [236] have shown that under the mild condition that

$$\alpha > 0 \quad \forall \quad w_t, w_{t+1} \in \mathcal{W}, \tag{13}$$

$\hat{P}(w)$ will asymptotically converge to the target distribution $P(w)$. According to Eq. (13), the probability of accepting a sample with lower score than its predecessor is always different from zero, which implies that the method will not ultimately get stuck into a local extremum. Intuitively, this is why the empirical sample distribution $\hat{P}(w)$ approximates the target distribution $P(w)$. In contrast to other Monte Carlo statistical methods, e.g. importance sampling, the MH algorithm tend to suffer less from the curse of dimensionality and are therefore preferred for sampling in high-dimensional spaces [236].

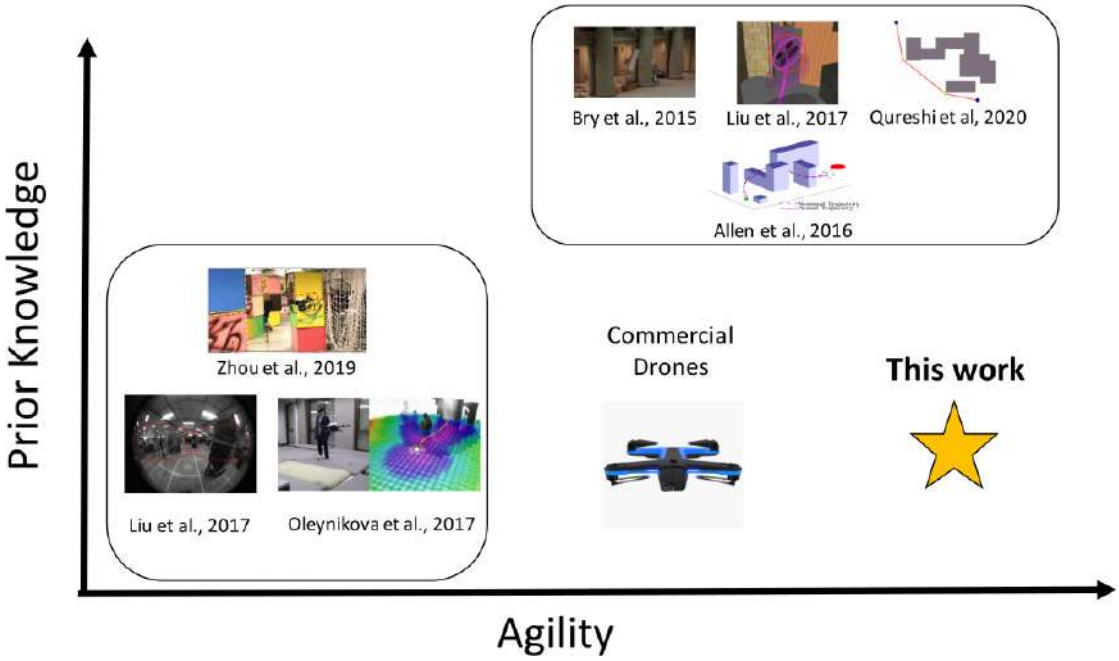
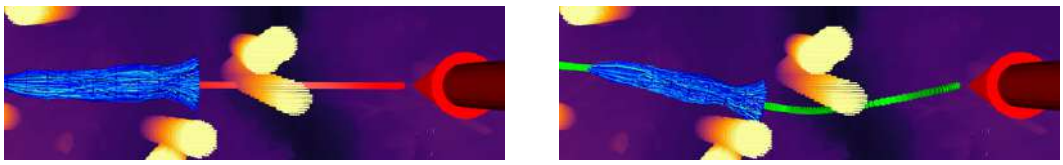


Figure S2 – Taxonomy of existing approaches for drone navigation in challenging and cluttered environments. Approaches are ordered with respect to the required prior knowledge about the environment and the maximum agility they achieve.



(a) Without global planning **(b)** With global planning

Figure S3 – Illustration of the influence of global planning on the sampled trajectories. Sampling around the raw reference trajectory (in red) strictly limits the expert's sight to the immediate horizon (a). Conversely, sampling around a global collision-free trajectory (in green) results in a bias towards obstacle-free regions even beyond the immediate horizon (b). Best viewed in color.

Bibliography

[1] Pieter Abbeel, Adam Coates, and Andrew Y Ng. "Autonomous helicopter aerobatics through apprenticeship learning". In: *The International Journal of Robotics Research* 29.13 (2010), pp. 1608–1639.

[2] Alessandro Achille and Stefano Soatto. "Where is the Information in a Deep Neural Network?" In: *arXiv preprint arXiv:1905.12213* (2019).

[3] Pulkit Agrawal et al. "Learning to Poke by Poking: Experiential Learning of Intuitive Physics". In: *International Conference on Neural Information Processing Systems (NIPS)*. 2016, pp. 5092–5100.

[4] *AI-Powered Drone Learns Extreme Acrobatics*. https://spectrum.ieee.org/automaton/robotics/drones/ai-powered-drone-extreme-acrobatics. 2020.

[5] Ross Allen and Marco Pavone. "A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance". In: *AIAA Guidance, Navigation, and Control Conference*. 2016, p. 1374.

[6] Christophe Andrieu et al. "An Introduction to MCMC for Machine Learning". In: *Mach. Learn.* 50.1-2 (2003), pp. 5–43.

[7] Marcin Andrychowicz et al. "Learning to learn by gradient descent by gradient descent". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016, pp. 3981–3989.

[8] Amado Antonini et al. "The Blackbird UAV dataset". In: *Int. J. Robot. Research* 39.10-11 (2020), pp. 1346–136.

[9] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[10] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 39.12 (2017), pp. 2481–2495. DOI: 10.1109/tpami.2016.2644615.

[11] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv:1803.01271* (2018).

[12] Christian Bailer, Bertram Taetz, and Didier Stricker. "Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2015, pp. 4015–4023.

[13] Bowen Baker et al. "Emergent Tool Use From Multi-Agent Autocurricula". In: *International Conference on Learning Representations*. 2020.

[14] Linchao Bao, Baoyuan Wu, and Wei Liu. "CNN in MRF: Video Object Segmentation via Inference in A CNN-Based Higher-Order Spatio-Temporal MRF". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[15] Andrew J. Barry, Peter R. Florence, and Russ Tedrake. "High-speed autonomous obstacle avoidance with pushbroom stereo". In: *J. Field Robot.* 35.1 (2018), pp. 52–68. DOI: 10.1002/rob.21741.

[16] Yoshua Bengio et al. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 41–48.

[17] D. Beymer et al. "A real-time computer vision system for measuring traffic parameters". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1997. DOI: 10.1109/cvpr.1997.609371.

[18] Christopher M Bishop. *Mixture density networks*. Aston University, 1994.

[19] Michael Blösch et al. "Vision based MAV navigation in unknown and unstructured environments". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2010, pp. 21–28.

[20] Charles Blundell et al. "Weight uncertainty in neural networks". In: *International Conference on Machine Learning*. 2015, pp. 1613–1622.

[21] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[22] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[23] Konstantinos Bousmalis et al. "Using simulation and domain adaptation to improve efficiency of deep robotic grasping". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018.

[24] Koller Boyen. "Tractable inference for complex stochastic processes". In: *InUAI* (1998), pp. 33–42.

[25] G.J. Brostow and R. Cipolla. "Unsupervised Bayesian Detection of Independent Motion in Crowds". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2006. DOI: 10.1109/cvpr.2006.320.

[26] Thomas Brox, Andrés Bruhn, and Joachim Weickert. "Variational Motion Segmentation with Level Sets". In: *IEEE European Conference on Computer Vision (ECCV)*. 2006, pp. 471–483. DOI: 10.1007/11744023_37.

[27] Adam Bry, Abraham Bachrach, and Nicholas Roy. "State estimation for aggressive flight in GPS-denied environments using onboard sensing". In: *International Conference on Robotics and Automation*. IEEE. 2012, pp. 1–8.

[28] Adam Bry and Nicholas Roy. "Rapidly-exploring Random Belief Trees for motion planning under uncertainty". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2011.

[29] Adam Bry et al. "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments". In: *The International Journal of Robotics Research* 34.7 (2015), pp. 969–1002.

[30] Zoya Bylinskii et al. "Towards the quantitative evaluation of visual attention models". In: *Vision research* 116 (2015), pp. 258–268.

[31] Cesar Cadena et al. “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age”. In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332.

[32] Stephen J. Carey et al. “A 100,000 fps Vision Sensor with Embedded 535 GOPS/W 256x256 SIMD Processor Array”. In: *VLSI Circuits Symp.* 2013.

[33] Vincent Casser et al. “Depth Prediction without the Sensors: Leveraging Structure for Unsupervised Learning from Monocular Videos”. In: *Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*. 2019, pp. 8001–8008.

[34] Tony F Chan and Luminita A Vese. “Active Contours Without Edges”. In: *IEEE TRANSACTIONS ON IMAGE PROCESSING* 10.2 (2001).

[35] Angel X Chang et al. “Shapenet: An information-rich 3d model repository”. In: *arXiv preprint arXiv:1512.03012* (2015).

[36] Chenyi Chen et al. “Deepdriving: Learning affordance for direct perception in autonomous driving”. In: *International Conference on Computer Vision (ICCV)*. 2015.

[37] Dian Chen et al. “Learning by cheating”. In: *Conference on Robot Learning*. PMLR. 2019, pp. 66–75.

[38] Ying Chen and Néstor O Pérez-Arancibia. “Controller Synthesis and Performance Optimization for Aerobatic Quadrotor Flight”. In: *IEEE Transactions on Control Systems Technology* (2019), pp. 1–16.

[39] Bowen Cheng et al. “Panoptic-DeepLab: A Simple, Strong, and Fast Baseline for Bottom-Up Panoptic Segmentation”. In: *CVPR*. 2020.

[40] Jingchun Cheng et al. “SegFlow: Joint Learning for Video Object Segmentation and Optical Flow”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017.

[41] Kurtland Chua et al. “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 4754–4765.

[42] Titus Cieslewski, Elia Kaufmann, and Davide Scaramuzza. “Rapid exploration with multi-rotors: A frontier selection method for high speed flight”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017, pp. 2135–2142.

[43] J. Civera, A.J. Davison, and J. Montiel. “Inverse Depth Parametrization for Monocular SLAM”. In: *IEEE Trans. Robot.* 24.5 (2008), pp. 932–945.

[44] Ronald Clark et al. “VINet: Visual-inertial odometry as a sequence-to-sequence learning problem”. In: *AAAI Conference on Artificial Intelligence*. 2017.

[45] Ignasi Clavera, David Held, and Pieter Abbeel. “Policy Transfer via Modularity and Reward Guiding”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017.

[46] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[47] Daniel Cremers and Stefano Soatto. “Motion Competition: A Variational Approach to Piecewise Parametric Motion Segmentation”. In: *IEEE International Journal of Computer Vision* 62.3 (2004), pp. 249–265. DOI: 10.1007/s11263-005-4882-4.

[48] Daniel Cremers and Stefano Soatto. "Motion competition: A variational approach to piecewise parametric motion segmentation". In: *International Journal of Computer Vision* 62.3 (2005), pp. 249–265.

[49] Jin Dundar Culurciello. "Robust convolutional neural networks under adversarial noise". In: *InWorkshop Proceedings of the ICLR* (2016).

[50] M. Davies et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning". In: *IEEE Micro* 38.1 (2018), pp. 82–99.

[51] J. Delmerico and D. Scaramuzza. "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018.

[52] J. Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.

[53] John Denker and Yann LeCun. "Transforming neural-net output levels to probability distributions". In: *Advances in Neural Information Processing Systems 3* (1991).

[54] Coline Devin et al. "Learning modular neural network policies for multi-task and multi-robot transfer". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.

[55] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Conference on Robot Learning (CORL)*. 2017, pp. 1–16.

[56] Paul Drews et al. "Aggressive Deep Driving: Combining Convolutional Neural Networks and Model Predictive Control". In: *Conference on Robot Learning (CoRL)*. 2017.

[57] David Eigen, Christian Puhrsch, and Rob Fergus. "Depth map prediction from a single image using a multi-scale deep network". In: *Conf. Neural Inf. Process. Syst. (NIPS)*. 2014, pp. 2366–2374.

[58] Ahmed Elnakib et al. "Medical image segmentation: a brief survey". In: *Multi Modality State-of-the-Art Medical Image Segmentation and Registration Methodologies*. Springer, 2011, pp. 1–39.

[59] Jakob Engel, Vladlen Koltun, and Daniel Cremers. "Direct sparse odometry". In: *IEEE transactions on pattern analysis and machine intelligence (T-PAMI)* 40.3 (2018), pp. 611–625.

[60] S. M. Ali Eslami et al. "Neural scene representation and rendering". In: *Science* 360.6394 (2018), pp. 1204–1210.

[61] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza. "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 620–626.

[62] Matthias Faessler et al. "Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle". In: *J. Field Robotics* 33.4 (2016), pp. 431–450.

[63] Alon Faktor and Michal Irani. "Video Object Segmentation by Non-Local Consensus voting". In: *British Machine Vision Conference (BMVC)*. British Machine Vision Association, 2014. DOI: 10.5244/c.28.21.

[64] Davide Falanga, Suseong Kim, and Davide Scaramuzza. "How Fast is Too Fast? The Role of Perception Latency in High-Speed Sense and Avoid". In: *IEEE Robot. Autom. Lett.* 4.2 (Apr. 2019), pp. 1884–1891. ISSN: 2377-3766. DOI: 10.1109/LRA.2019.2898117.

[65] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. "Dynamic obstacle avoidance for quadrotors with event cameras". In: *Science Robotics* 5.40 (2020).

[66] Davide Falanga et al. "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.

[67] Davide Falanga et al. "PAMPC: Perception-aware model predictive control for quadrotors". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–8.

[68] Di Feng, Lars Rosenbaum, and Klaus Dietmayer. "Towards Safe Autonomous Driving: Capture Uncertainty in the Deep Neural Network For Lidar 3D Vehicle Detection". In: *Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems* (2018).

[69] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. 2017, pp. 1126–1135.

[70] Pete Florence, John Carter, and Russ Tedrake. "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps". In: *Algorithmic Foundations of Robotics XII*. Springer, 2020, pp. 304–319.

[71] Philipp Foehn et al. "AlphaPilot: Autonomous Drone Racing". In: *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA, July 2020. DOI: 10.15607/RSS.2020.XVI.081.

[72] C. Forster et al. "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry". In: *IEEE Transactions on Robotics* 3.1 (2017).

[73] Christian Forster et al. "SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems". In: *IEEE Transactions on Robotics* 33.2 (2017), pp. 249–265. DOI: 10.1109/TRO.2016.2623335.

[74] Marco Fraccaro et al. "A disentangled recognition and nonlinear dynamics model for unsupervised learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3601–3610.

[75] Frey and Hinton. "Variational learning in nonlinear Gaussian belief networks". In: *Neural Comput.* 11.1 (1999), pp. 193–213.

[76] Paul Furgale and Timothy D. Barfoot. "Visual teach and repeat for long-range rover autonomy". In: *J. Field Robot.* 27.5 (2010), pp. 534–560.

[77] Fadri Furrer et al. "RotorS–A modular Gazebo MAV simulator framework". In: *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[78] Fadri Furrer et al. "RotorS—A modular gazebo MAV simulator framework". In: *Robot Operating System (ROS)*. Springer, 2016, pp. 595–625.

[79] Sajad Saeedi G. et al. "3D Mapping for Autonomous Quadrotor Aircraft". In: *Unmanned Syst.* 5.3 (2017), pp. 181–196. DOI: 10.1142/S2301385017400064. URL: https://doi.org/10.1142/S2301385017400064.

[80] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation:Representing Model Uncertainty in Deep Learning". In: *Proceedings of the 33rd International Conference on Machine Learning* (2015).

[81] Yarin Gal, Jiri Hron, and Alex Kendall. "Concrete dropout". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3581–3590.

[82] Guillermo Gallego et al. "Event-based Vision: A Survey". In: *IEEE Trans. Pattern Anal. Machine Intell.* (2020).

[83] Jean Gallier. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN: 1558605991.

[84] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. "Learning to fly by crashing". In: *International Conference on Intelligent Robots and Systems, IROS*. 2017, pp. 3948–3955.

[85] Wei Gao et al. "Intention-Net: Integrating Planning and Deep Learning for Goal-Directed Autonomous Navigation". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017.

[86] Ravi Garg et al. "Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue". In: *Eur. Conf. Comput. Vis. (ECCV)*. 2016, pp. 740–756.

[87] Jochen Gast and Stefan Roth. "Lightweight Probabilistic Deep Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018).

[88] Daniel Gehrig et al. "End-to-End Learning of Representations for Asynchronous Event-Based Data". In: *International Conference on Computer Vision, ICCV*. 2019, pp. 5632–5642. DOI: 10.1109/ICCV.2019.00573.

[89] Asma Ghandeharioun et al. "Approximating Interactive Human Evaluation with Self-Play for Open-Domain Dialog Systems". In: 2019, pp. 13658–13669.

[90] Alison L Gibbs and Francis Edward Su. "On choosing and bounding probability metrics". In: *International Statistical Review* 70.3 (2002), pp. 419–435.

[91] James J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.

[92] A. Giusti et al. "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots". In: *IEEE Robotics and Automation Letters* (2016).

[93] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. "Unsupervised Monocular Depth Estimation with Left-Right Consistency". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2017, pp. 6602–6611.

[94] E Bruce Goldstein and James Brockmole. *Sensation and perception*. Cengage Learning, 2016.

[95] Ariel Gordon et al. "Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras". In: *Int. Conf. Comput. Vis. (ICCV)*. 2019, pp. 8977–8986.

[96] Joachim de Greeff and Tony Belpaeme. "Why Robots Should Be Social: Enhancing Machine Learning through Social Human-Robot Interaction". In: *PLOS ONE* 10.9 (2015).

[97] Joan E. Grusec. "Social learning theory and developmental psychology: The legacies of Robert R. Sears and Albert Bandura." In: *A century of developmental psychology.* American Psychological Association, 1994, pp. 473–497.

[98] Winter Guerra et al. "FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2019.

[99] Abhishek Gupta et al. "Learning invariant feature spaces to transfer skills with reinforcement learning". In: *Internation Conference on Learning Representation (ICLR)* (2017).

[100] Raia Hadsell et al. "Learning long-range vision for autonomous off-road driving". In: *J. Field Robot.* 26.2 (2009), pp. 120–144.

[101] York Hagmayer et al. "Causal Reasoning Through Intervention". In: *Causal Learning*. Oxford University Press, 2007, pp. 86–100.

[102] Nicklas Hansen et al. "Self-supervised policy adaptation during deployment". In: *arXiv preprint arXiv:2007.04309* (2020).

[103] Christopher G. Harris and Mike Stephens. "A Combined Corner and Edge Detector". In: *Proceedings of the Alvey Vision Conference*. 1988.

[104] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Second Edition. Cambridge University Press, 2003.

[105] Richard I. Hartley. "In Defense of the Eight-Point Algorithm". In: *IEEE Trans. Pattern Anal. Machine Intell.* 19.6 (1997), pp. 580–593.

[106] Richard I. Hartley and Peter F. Sturm. "Triangulation". In: *Computer Vision and Image Understanding* 68.2 (1997), pp. 146–157.

[107] W. K. Hastings. "Monte Carlo sampling methods using Markov chains and their applications". In: *Biometrika* 57.1 (1970), pp. 97–109.

[108] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[109] Lionel Heng et al. "Autonomous Visual Mapping and Exploration With a Micro Aerial Vehicle". In: *J. Field Robotics* 31.4 (2014), pp. 654–675.

[110] Daniel Hernandez-Juarez et al. "Embedded Real-time Stereo Estimation via Semi-Global Matching on the GPU". In: *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*. 2016.

[111] José Miguel Hernández-Lobato and Ryan Adams. "Probabilistic backpropagation for scalable learning of bayesian neural networks". In: *International Conference on Machine Learning*. 2015, pp. 1861–1869.

[112] Juliane Hilf and Klaus Umbach. "The Commercial Use of Drones". In: *Computer Law Review International* 16.3 (2015).

[113] H. Hirschmuller and D. Scharstein. "Evaluation of Stereo Matching Costs on Images with Radiometric Differences". In: *IEEE Trans. Pattern Anal. Machine Intell.* 31.9 (2009), pp. 1582–1599.

[114] Heiko Hirschmuller. "Stereo processing by semiglobal matching and mutual information". In: *IEEE Transactions on pattern analysis and machine intelligence* 30.2 (2007), pp. 328–341.

[115] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780.

[116] Andreas Hock and Angela P Schoellig. "Distributed iterative learning control for multi-agent systems". In: *Autonomous Robots* 43.8 (2019), pp. 1989–2010.

[117] Andrew Howard et al. "Searching for MobileNetV3". In: *International Conference on Computer Vision, ICCV*. 2019, pp. 1314–1324.

[118] Ian P Howard and Brian J Rogers. *Seeing in depth, Vol. 2: Depth perception.* University of Toronto Press, 2002.

[119] G. Huang et al. "Densely Connected Convolutional Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243.

[120] Jemin Hwangbo, Joonho Lee, and Marco Hutter. "Per-contact iteration method for solving contact dynamics". In: *IEEE Robot. Autom. Lett.* 3.2 (2018), pp. 895–902.

[121] Jemin Hwangbo et al. "Control of a quadrotor with reinforcement learning". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103.

[122] Myung Hwangbo, Jun-Sik Kim, and Takeo Kanade. "IMU self-calibration using factorization". In: *IEEE Transactions on Robotics* 29.2 (2013), pp. 493–507.

[123] Osband Ian. "Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout". In: *Advances in Neural Information Processing Systems Workshops*. 2016.

[124] Eddy Ilg et al. "Flownet 2.0: Evolution of optical flow estimation with deep networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2462–2470.

[125] Laurent Itti and Christof Koch. "A saliency-based search mechanism for overt and covert shifts of visual attention". In: *Vision research* 40.10-12 (2000), pp. 1489–1506.

[126] Suyog Dutt Jain, Bo Xiong, and Kristen Grauman. "FusionSeg: Learning to Combine Motion and Appearance for Fully Automatic Segmentation of Generic Objects in Videos". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017. DOI: 10.1109/cvpr.2017.228.

[127] Stephen James, Andrew J Davison, and Edward Johns. "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task". In: *Conference on Robot Learning (CoRL)* (2017).

[128] Natasha Jaques et al. "Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning". In: *International Conference on Machine Learning(ICML)*. Vol. 97. 2019, pp. 3040–3049.

[129] Matthew Johnson-Roberson et al. "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?" In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.

[130] Arthur Juliani et al. "Unity: A general platform for intelligent agents". In: *arXiv e-prints* (2018).

[131] Sunggoo Jung et al. "Perception, Guidance, and Navigation for Indoor Autonomous Drone Racing Using Deep Learning". In: *IEEE Robotics Autom. Lett.* 3.3 (2018), pp. 2539–2544. DOI: 10.1109/LRA.2018.2808368.

[132] Gregory Kahn et al. "PLATO: Policy learning using adaptive trajectory optimization". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. IEEE, 2017, pp. 3342–3349.

[133] Gregory Kahn et al. "Uncertainty-Aware Reinforcement Learning for Collision Avoidance". In: *Arxiv Preprint*. 2017.

[134] Nitin R Kapania. "Trajectory Planning and Control for an Autonomous Race Vehicle". PhD thesis. Stanford University, 2016.

[135] Sertac Karaman and Emilio Frazzoli. "High-speed flight in an ergodic forest". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2899–2906.

[136] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. "Neural 3D Mesh Renderer". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2018, pp. 3907–3916.

[137] Elia Kaufmann et al. "Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2019.

[138] Elia Kaufmann* et al. "Deep Drone Acrobatics". In: *RSS: Robotics, Science, and Systems* (2020).

[139] Elia Kaufmann* et al. "Deep drone racing: Learning agile flight in dynamic environments". In: *Conference on Robot Learning (CoRL)*. 2018.

[140] John C Kegelman, Lene K Harbott, and J Christian Gerdes. "Insights into vehicle trajectories at the handling limits: analysing open data from race car drivers". In: *Vehicle system dynamics* 55.2 (2017), pp. 191–207.

[141] Alex Kendall and Yarin Gal. "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In: *31st Conference on Neural Information Processing Systems (NIPS 2017)* (2017).

[142] Leonid Keselman et al. "Intel RealSense Stereoscopic Depth Cameras". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2017.

[143] Lee Keuntaek et al. "Ensemble Bayesian Decision Making with Redundant Deep Perceptual Control Policies". In: *ArXiv* (2018).

[144] Margret Keuper, Bjoern Andres, and Thomas Brox. "Motion Trajectory Segmentation via Minimum Cost Multicuts". In: *IEEE International Conference on Computer Vision (ICCV)*. 2015. DOI: 10.1109/iccv.2015.374.

[145] Oussama Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". In: *Autonomous Robot Vehicles*. Springer, 1986, pp. 396–404.

[146] H. Kim and Andrew Ng. "Stable adaptive control with online learning". In: *Advances in Neural Information Processing Systems*. Ed. by L. Saul, Y. Weiss, and L. Bottou. Vol. 17. MIT Press, 2005. URL: https://proceedings.neurips.cc/paper/2004/file/8e68c3c7bf14ad0bcaba52babfa470bd-Paper.pdf.

[147] Jinkyu Kim and John Canny. "Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention". In: *The IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[148] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. 2015.

[149] Jan J Koenderink. "Optic flow". In: *Vision research* 26.1 (1986), pp. 161–179.

[150] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[151] Yeong Jun Koh and Chang-Su Kim. "Primary Object Segmentation in Videos Based on Region Augmentation and Reduction". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017. DOI: 10.1109/cvpr.2017.784.

[152] Stefan Kohlbrecher et al. "Hector open source modules for autonomous mapping and navigation with rescue robots". In: *Robot Soccer World Cup*. Springer. 2013, pp. 624–631.

[153] Philipp Krähenbühl and Vladlen Koltun. "Efficient inference in fully connected crfs with gaussian edge potentials". In: *Advances in neural information processing systems*. 2011, pp. 109–117.

[154] Krisada Kritayakirana and J Christian Gerdes. "Autonomous vehicle control at the limits of handling". In: *International Journal of Vehicle Autonomous Systems* 10.4 (2012), pp. 271–296.

[155] Iro Laina et al. "Deeper depth prediction with fully convolutional residual networks". In: *(3DV)*. IEEE. 2016, pp. 239–248.

[156] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6402–6413.

[157] Dong Lao and Ganesh Sundaramoorthi. "Extending Layered Models to 3D Motion". In: *IEEE European Conference on Computer Vision (ECCV)*. 2018.

[158] Keuntaek Lee, Jason Gibson, and Evangelos A. Theodorou. "Aggressive Perception-Aware Navigation Using Deep Optical Flow Dynamics and PixelMPC". In: *IEEE Robotics Autom. Lett.* 5.2 (2020), pp. 1207–1214.

[159] Karen Leung, Nikos Aréchiga, and Marco Pavone. "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods". In: *arXiv preprint arXiv:2008.00097* (2020).

[160] Sergey Levine et al. "End-to-End Training of Deep Visuomotor Policies". In: *J. Mach. Learn. Res.* 17 (2016), 39:1–39:40.

[161] S Li et al. "Autonomous drone race: A computationally efficient vision-based navigation and control strategy". In: *arXiv preprint arXiv:1809.05958* (2018).

[162] Shuo Li et al. "Aggressive Online Control of a Quadrotor via Deep Network Representations of Optimality Principles". In: *arXiv:1912.07067* (2019).

[163] Shuo Li et al. "Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone". In: *J. Field Robotics* 37.4 (2020), pp. 667–692. DOI: 10.1002/rob.21956.

[164] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[165] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. "Learning Efficient Point Cloud Generation for Dense 3D Object Reconstruction". In: *AAAI Conf. Artificial Intell.* 2018, pp. 7114–7121.

[166] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *European Conference on Computer Vision (ECCV)*. 2014.

[167] Rosanne Liu et al. "An intriguing failing of convolutional neural networks and the CoordConv solution". In: *Conf. Neural Inf. Process. Syst. (NIPS)*. 2018, pp. 9605–9616.

[168] Sikang Liu et al. "Search-based motion planning for aggressive flight in se (3)". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2439–2446.

[169] Lennart Ljung and Svante Gunnarsson. "Adaptation and tracking in system identification—A survey". In: *Automatica* 26.1 (1990), pp. 7–21.

[170] Giuseppe Loianno and Davide Scaramuzza. "Special issue on future challenges and opportunities in vision-based drone navigation". In: *Journal of Field Robotics* 37.4 (2020), pp. 495–496.

[171] Giuseppe Loianno et al. "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU". In: *IEEE Robotics and Automation Letters* 2.2 (2016), pp. 404–411.

[172] H. C. Longuet-Higgins. "A computer algorithm for reconstructing a scene from two projections". In: *Nature* 293 (1981), pp. 133–135.

[173] Antonio Loquercio, A. Dosovitskiy, and D. Scaramuzza. "Learning Depth With Very Sparse Supervision". In: *IEEE Robotics and Automation Letters* 5 (2020), pp. 5542–5549.

[174] Antonio Loquercio, Mattia Segù, and Davide Scaramuzza. "A General Framework for Uncertainty Estimation in Deep Learning". In: *IEEE Robotics Autom. Lett.* 5.2 (2020), pp. 3153–3160. DOI: 10.1109/LRA.2020.2974682.

[175] Antonio Loquercio et al. "Agile Autonomy: Learning High-Speed Flight in the Wild". In: *Science Robotics* 7 (58 2021), pp. 1–12.

[176] Antonio Loquercio et al. "Deep Drone Racing: From Simulation to Reality With Domain Randomization". In: *IEEE Trans. Robotics* 36.1 (2019), pp. 1–14.

[177] Antonio Loquercio et al. "DroNet: Learning to Fly by Driving". In: *IEEE Robotics Autom. Lett.* 3.2 (2018), pp. 1088–1095.

[178] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision". In: *International Joint Conference on Artificial Intelligence*. 1981.

[179] Sergei Lupashin et al. "A simple learning strategy for high-speed quadrocopter multi-flips". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2010, pp. 1642–1648.

[180] Simon Lynen et al. "Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization". In: *Robotics: Science and Systems XI*. July 2015. DOI: 10.15607/rss.2015.xi.037.

[181] David JC MacKay. "A practical Bayesian framework for backpropagation networks". In: *Neural Computation* 4.3 (1992), pp. 448–472.

[182] Reza Mahjourian, Martin Wicke, and Anelia Angelova. "Unsupervised Learning of Depth and Ego-Motion From Monocular Video Using 3D Geometric Constraints". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2018, pp. 5667–5675.

[183] R. Mahony, V. Kumar, and P. Corke. "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor". In: *IEEE Robot. Autom. Mag.* 19.3 (2012), pp. 20–32. ISSN: 1070-9932. DOI: 10.1109/MRA.2012.2206474.

[184] Michele Mancini et al. "Towards Domain Independence for Learning-Based Monocular Depth Estimation". In: *IEEE Robot. Autom. Lett.* (2017).

[185] K.-K. Maninis et al. "Video Object Segmentation Without Temporal Information". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2018).

[186] Ana I. Maqueda et al. "Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 5419–5427. DOI: 10.1109/CVPR.2018.00568.

[187] Nikolaus Mayer et al. "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2016, pp. 4040–4048.

[188] K. N. McGuire et al. "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment". In: *Science Robotics* 4.35 (2019), eaaw9710. DOI: 10.1126/scirobotics.aaw9710.

[189] Simon Meister, Junhwa Hur, and Stefan Roth. "UnFlow: Unsupervised learning of optical flow with a bidirectional census loss". In: *AAAI Conf. Artificial Intell.* 2018, pp. 146–157.

[190] Daniel Mellinger and Vijay Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2011, pp. 2520–2525.

[191] Daniel Mellinger, Nathan Michael, and Vijay Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 664–674.

[192] Nico Messikommer et al. "Event-Based Asynchronous Sparse Convolutional Networks". In: *European Conference Computer Vision (ECCV)*. Vol. 12353. 2020, pp. 415–431. DOI: 10.1007/978-3-030-58598-3_25.

[193] Nathan Michael et al. "Collaborative mapping of an earthquake-damaged building via ground and aerial robots". In: *Journal of Field Robotics* 29.5 (2012), pp. 832–841.

[194] Francesco Milano et al. "Primal-Dual Mesh Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[195] Thomas Peter Minka. "A family of algorithms for approximate Bayesian inference". PhD thesis. Massachusetts Institute of Technology, 2001.

[196] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nat.* 518.7540 (2015), pp. 529–533.

[197] Kartik Mohta et al. "Fast, autonomous flight in GPS-denied and cluttered environments". In: *J. Field Robot.* 35.1 (2018), pp. 101–120. DOI: 10.1002/rob.21774.

[198] H. Moon et al. "The IROS 2016 Competitions". In: *IEEE Robotics and Automation Magazine* 24.1 (2017), pp. 20–29.

[199] Hyungpil Moon et al. "Challenges and implemented technologies used in autonomous drone racing". In: *Intelligent Service Robotics* 1.1 (2019), pp. 611–625.

[200] Benjamin Morrell et al. "Differential flatness transformations for aggressive quadrotor flight". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018.

[201] Mark W Mueller, Markus Hehn, and Raffaello D'Andrea. "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2013.

[202] Matthias Müller et al. "Driving Policy Transfer via Modularity and Abstraction". In: *Conference on Robot Learning*. 2018, pp. 1–15.

[203] Matthias Müller et al. "Sim4cv: A photo-realistic simulator for computer vision applications". In: *Int. J. Comput. Vis.* 126.9 (2018), pp. 902–919.

[204] Matthias Müller et al. "Teaching UAVs to Race: End-to-End Regression of Agile Controls in Simulation". In: *European Conference on Computer Vision Workshops*. 2018.

[205] David Mumford and Jayant Shah. "Optimal approximations by piecewise smooth functions and associated variational problems". In: *Communications on pure and applied mathematics* 42.5 (1989), pp. 577–685.

[206] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Trans. Robot.* 31.5 (2015), pp. 1147–1163.

[207] K Nakayama and JM Loomis. "Optical velocity patterns, velocity-sensitive neurons, and space perception: a hypothesis". In: *Perception* 3.1 (1974), pp. 63–80.

[208] K.S. Narendra and K. Parthasarathy. "Identification and control of dynamical systems using neural networks". In: *IEEE Transactions on Neural Networks* 1.1 (1990), pp. 4–27.

[209] Radford M Neal. "Bayesian learning for neural networks". In: *PhD thesis, University of Toronto* (1995).

[210] Lukas Neumann, Andrew Zisserman, and Andrea Vedaldi. "Relaxed Softmax: Efficient Confidence Auto-Calibration for Safe Pedestrian Detection". In: *Advances in Neural Information Processing Systems Workshops*. 2018.

[211] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. "DTAM: Dense Tracking and Mapping in Real-Time". In: *Int. Conf. Comput. Vis. (ICCV)*. 2011, pp. 2320–2327.

[212] Peter Ochs, Jitendra Malik, and Thomas Brox. "Segmentation of Moving Objects by Long Term Video Analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 36.6 (2014), pp. 1187–1200. DOI: 10.1109/tpami.2013.242.

[213] Helen Oleynikova et al. "Continuous-time trajectory optimization for online UAV replanning". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2016.

[214] Helen Oleynikova et al. "Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017, pp. 1366–1373.

[215] T. Ozaslan et al. "Autonomous Navigation and Mapping for Inspection of Penstocks and Tunnels With MAVs". In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1740–1747.

[216] Daniele Palossi et al. "A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones". In: *Demo at IROS* (2018).

[217] Daniele Palossi et al. "A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones". In: *IEEE Internet of Things Journal* (2019).

[218] Yunpeng Pan et al. "Agile Autonomous Driving Using End-to-End Deep Imitation Learning". In: *Robotics: Science and Systems (RSS)*. 2018.

[219] Yu Pang and Haibin Ling. "Finding the best from the second bests-inhibiting subjective bias in evaluation of visual tracking algorithms". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2013, pp. 2784–2791.

[220] Anestis Papazoglou and Vittorio Ferrari. "Fast Object Segmentation in Unconstrained Video". In: *IEEE International Conference on Computer Vision (ICCV)*. 2013. DOI: 10.1109/iccv.2013.223.

[221] Judea Pearl et al. "Causal inference in statistics: An overview". In: *Statistics surveys* 3 (2009), pp. 96–146.

[222] X. B. Peng et al. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *International Conference on Robotics and Automation (ICRA)*. 2018, pp. 3803–3810.

[223] F. Perazzi et al. "A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016. DOI: 10.1109/cvpr.2016.85.

[224] Mark Pfeiffer et al. "From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.

[225] Marc Pollefeys, Reinhard Koch, and Luc Van Gool. "Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters". In: *International Journal of Computer Vision* 32.1 (1999), pp. 7–25.

[226] Janis Postels et al. "Sampling-Free Epistemic Uncertainty Estimation Using Approximated Variance Propagation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019.

[227] Tong Qin, Peiliang Li, and Shaojie Shen. "Vins-mono: A robust and versatile monocular visual-inertial state estimator". In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020.

[228] René Ranftl and Vladlen Koltun. "Deep fundamental matrix estimation". In: *European Conference on Computer Vision (ECCV)*. 2018.

[229] René Ranftl et al. "Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer". In: *arXiv:1907.01341* (2019).

[230] Sachin Ravi and Hugo Larochelle. "Optimization as a Model for Few-Shot Learning". In: *International Conference on Learning Representations (ICLR)*. 2017.

[231] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[232] Danilo Jimenez Rezende et al. "Unsupervised Learning of 3D Structure from Images". In: *Conf. Neural Inf. Process. Syst. (NIPS)*. 2016, pp. 4997–5005.

[233] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments". In: *Proc. Int. Symp. Robot. Research (ISRR)*. 2013.

[234] Charles Richter and Nicholas Roy. "Safe Visual Navigation via Deep Learning and Novelty Detection". In: *Robotics: Science and Systems*. 2017.

[235] Stephan R. Richter et al. "Playing for Data: Ground Truth from Computer Games". In: *European Conference on Computer Vision (ECCV)*. 2016.

[236] Gareth O Roberts and Adrian FM Smith. "Simple conditions for the convergence of the Gibbs sampler and Metropolis-Hastings algorithms". In: *Stochastic processes and their applications* 49.2 (1994), pp. 207–216.

[237] José I Ronda, Antonio Valdés, and Guillermo Gallego. "Line geometry and camera autocalibration". In: *Journal of Mathematical Imaging and Vision* 32.2 (2008), pp. 193–214.

[238] Antoni Rosinol Vidal et al. "Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High Speed Scenarios". In: *IEEE Robot. Autom. Lett.* 3.2 (Apr. 2018), pp. 994–1001. DOI: 10.1109/LRA.2018.2793357.

[239] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.

[240] Stéphane Ross et al. "Learning monocular reactive UAV control in cluttered natural environments". In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2013, pp. 1765–1772.

[241] Kenneth J Rothman and Sander Greenland. "Causation and causal inference in epidemiology". In: *American journal of public health* 95.S1 (2005), S144–S150.

[242] Christian Rupprecht et al. "Learning in an Uncertain World: Representing Ambiguity Through Multiple Hypotheses". In: *International Conference on Computer Vision, ICCV*. 2017, pp. 3611–3620.

[243] Andrei A Rusu et al. "Sim-to-real robot learning from pixels with progressive nets". In: *Conference on Robot Learning (CoRL)*. 2017.

[244] Markus Ryll et al. "Efficient trajectory planning for high speed flight in unknown environments". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 732–738.

[245] Grady William s et al. "Aggressive Driving with Model Predictive Path Integral Control". In: *2016 IEEE Int. Conf. Robot. Autom. (ICRA)*. Stockholm, Sweden, May 2016, pp. 1433–1440.

[246] Fereshteh Sadeghi and Sergey Levine. "CAD2RL: Real Single-Image Flight Without a Single Real Image". In: *Robotics: Science and Systems RSS*. Ed. by Nancy M. Amato et al. 2017.

[247] Fereshteh Sadeghi et al. "Sim2Real Viewpoint Invariant Visual Servoing by Recurrent Control". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. DOI: 10.1109/cvpr.2018.00493.

[248] Axel Sauer and Andreas Geiger. "Counterfactual Generative Networks". In: *International Conference on Learning Representations*. 2021.

[249] Alexander Sax et al. "Learning to Navigate Using Mid-Level Visual Priors". In: *Conference on Robot Learning*. PMLR. 2020, pp. 791–812.

[250] Davide Scaramuzza et al. "Vision-Controlled Micro Flying Robots: From System Design to Autonomous Navigation and Mapping in GPS-Denied Environments". In: *IEEE Robotics Autom. Mag.* 21.3 (2014), pp. 26–40.

[251] Sebastian Scherer et al. "River mapping from a flying robot: state estimation, river detection, and obstacle mapping". In: *Autonomous Robots* 33.1-2 (2012), pp. 189–214.

[252] Angela P. Schoellig, Fabian L. Mueller, and Raffaello D'Andrea. "Optimization-based iterative learning for precise quadrocopter trajectory tracking". In: *Auton. Robots* 33.1-2 (2012), pp. 103–127.

[253] J. L. Schönberger and J.-M. Frahm. "Structure-from-Motion Revisited". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2016, pp. 4104–4113.

[254] John Schulman et al. "Trust region policy optimization". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 1889–1897.

[255] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[256] Rafael Serrano-Gotarredona et al. "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory-Processing-Learning-Actuating System for High-Speed Visual Object Recognition and Tracking". In: *IEEE Trans. Neural Netw.* 20.9 (2009), pp. 1417–1438. DOI: 10.1109/TNN.2009.2023653.

[257] Shital Shah et al. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and Service Robot.* Springer. 2018, pp. 621–635.

[258] Apoorva Sharma, Navid Azizan, and Marco Pavone. *Sketching Curvature for Efficient Out-of-Distribution Detection for Deep Neural Networks*. 2021. arXiv: 2102.12567.

[259] Shaojie Shen et al. "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.

[260] Shaojie Shen et al. "Vision-Based State Estimation and Trajectory Control Towards High-Speed Flight with a Quadrotor". In: *Robotics: Science and Systems (RSS)*. 2013.

[261] Jianbo Shi and J. Malik. "Motion segmentation and tracking using normalized cuts". In: *IEEE International Conference on Computer Vision (ICCV)*. 1998. DOI: 10.1109/iccv.1998.710861.

[262] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nat.* 550.7676 (2017), pp. 354–359.

[263] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[264] Nikolai Smolyanskiy et al. "Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)* (2017).

[265] Hongmei Song et al. "Pyramid Dilated Deeper ConvLSTM for Video Salient Object Detection". In: *IEEE European Conference on Computer Vision (ECCV)*. 2018. DOI: 10.1007/978-3-030-01252-6_44.

[266] Yunlong Song et al. "Flightmare: A Flexible Quadrotor Simulator". In: *Conference on Robot Learning*. 2020.

[267] Riccardo Spica et al. "A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing". In: *IEEE Trans. Robotics* 36.5 (2020), pp. 1389–1403.

[268] Nitish Srivastava et al. "Dropout: A simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014).

[269] Freek Stulp et al. "Learning to grasp under uncertainty". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 5703–5708.

[270] Jürgen Sturm et al. "A benchmark for the evaluation of RGB-D SLAM systems". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2012, pp. 573–580.

[271] Qinliang Su et al. "Nonlinear statistical learning with truncated gaussian graphical models". In: *International Conference on Machine Learning*. 2016, pp. 1948–1957.

[272] Deqing Sun et al. "A Fully-Connected Layered Model of Foreground and Background Flow". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2013. DOI: 10.1109/cvpr.2013.317.

[273] Deqing Sun et al. "PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2018, pp. 8934–8943.

[274] S. Sun et al. "Autonomous Quadrotor Flight despite Rotor Failure with Onboard Vision Sensors: Frames vs. Events". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 580–587. DOI: 10.1109/LRA.2020.3048875.

[275] Niko Sünderhauf et al. "The limits and potentials of deep learning for robotics". In: *Int. J. Robot. Research* 37.4-5 (2018), pp. 405–420.

[276] O. Tahri and F. Chaumette. "Point-based and region-based image moments for visual servoing of planar objects". In: *IEEE Transactions on Robotics* 21.6 (2005), pp. 1116–1127.

[277] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. "Multi-view 3D Models from Single Images with a Convolutional Network". In: *Eur. Conf. Comput. Vis. (ECCV)*. 2016, pp. 322–337.

[278] Brian Taylor, Vasiliy Karasev, and Stefano Soatto. "Causal video object segmentation from persistence of occlusions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7299055.

[279] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017.

[280] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2012, pp. 5026–5033.

[281] Pavel Tokmakov, Karteek Alahari, and Cordelia Schmid. "Learning Motion Patterns in Videos". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017. DOI: 10.1109/cvpr.2017.64.

[282] Pavel Tokmakov, Karteek Alahari, and Cordelia Schmid. "Learning Video Object Segmentation with Visual Memory". In: *IEEE International Conference on Computer Vision (ICCV)*. 2017. DOI: 10.1109/iccv.2017.480.

[283] Erkki Tomppo. "Models and methods for analysing spatial patterns of trees". PhD dissertation. Finnish Forest Research Institute, 1986.

[284] Jesus Tordesillas, Brett Thomas Lopez, and Jonathan P. How. "FASTER: Fast and Safe Trajectory Planner for Flights in Unknown Environments". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 1934–1940. DOI: 10.1109/IROS40897.2019.8968021.

[285] B. Triggs et al. "Bundle Adjustment – A Modern Synthesis". In: *Vision Algorithms: Theory and Practice*. Ed. by W. Triggs, A. Zisserman, and R. Szeliski. Vol. 1883. LNCS. Springer Verlag, 2000, pp. 298–372.

[286] David Tsai, Matthew Flagg, and James Rehg. "Motion Coherent Tracking with Multi-label MRF optimization". In: *British Machine Vision Conference (BMVC) 2010*. British Machine Vision Association, 2010. DOI: 10.5244/c.24.56.

[287] Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. "Multi-View Consistency as Supervisory Signal for Learning Shape and Pose Prediction". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2018, pp. 2897–2905.

[288] Udacity. *An Open Source Self-Driving Car*. https://www.udacity.com/self-driving-car. 2016.

[289] Benjamin Ummenhofer et al. "Demon: Depth and motion network for learning monocular stereo". In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2017, pp. 5038–5047.

[290] Hal R Varian. "Causal inference in economics and marketing". In: *Proceedings of the National Academy of Sciences* 113.27 (2016), pp. 7310–7315.

[291] Jon Verbeke and Joris De Schutter. "Experimental maneuverability and agility quantification for rotary unmanned aerial vehicle". In: *International Journal of Micro Air Vehicles* 10.1 (2018), pp. 3–11.

[292] Hao Wang, SHI Xingjian, and Dit-Yan Yeung. "Natural-parameter networks: A class of probabilistic neural networks". In: *Advances in Neural Information Processing Systems*. 2016, pp. 118–126.

[293] J.Y.A. Wang and E.H. Adelson. "Representing moving images with layers". In: *IEEE Transactions on Image Processing* 3.5 (1994), pp. 625–638. DOI: 10.1109/83.334981.

[294] Wenguan Wang, Jianbing Shen, and Fatih Porikli. "Saliency-aware geodesic video object segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015. DOI: 10.1109/cvpr.2015.7298961.

[295] Yunhai Wang et al. "Active co-analysis of a set of shapes". In: *Transactions on Computer Graphics* 31.6 (2012), p. 165.

[296] Tonghan Wang* et al. "Learning Nearly Decomposable Value Functions Via Communication Minimization". In: *International Conference on Learning Representations*. 2020.

[297] Wang Wenguan et al. "Learning unsupervised video object segmentation through visual attention". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3064–3074.

[298] Hao Wu et al. “Deep generative markov state models”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 3975–3984.

[299] Markus Wulfmeier, Ingmar Posner, and Pieter Abbeel. “Mutual alignment transfer learning”. In: *Conference on Robot Learning (CoRL)*. 2017.

[300] Jianxiong Xiao, Andrew Owens, and Antonio Torralba. “Sun3D: A database of big spaces reconstructed using sfm and object labels”. In: *Int. Conf. Comput. Vis. (ICCV)*. 2013, pp. 1625–1632.

[301] Huazhe Xu et al. “End-To-End Learning of Driving Models From Large-Scale Video Datasets”. In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. July 2017.

[302] Guang-Zhong Yang et al. “The grand challenges of Science Robotics”. In: *Science Robotics* 3.14 (2018), eaar7650.

[303] Shichao Yang et al. “Obstacle Avoidance through Deep Networks based Intermediate Perception”. In: *arXiv preprint arXiv:1704.08759* (2017).

[304] Yanchao Yang and Stefano Soatto. “Conditional prior networks for optical flow”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 271–287.

[305] Yanchao Yang and Ganesh Sundaramoorthi. “Shape tracking with occlusions via coarse-to-fine region-based sobolev descent”. In: *IEEE transactions on pattern analysis and machine intelligence* 37.5 (2015), pp. 1053–1066.

[306] Yanchao Yang, Ganesh Sundaramoorthi, and Stefano Soatto. “Self-occlusions and disocclusions in causal video object segmentation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4408–4416.

[307] Yanchao Yang* et al. “Unsupervised Moving Object Detection via Contextual Information Separation”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2019, pp. 879–888. DOI: 10.1109/CVPR.2019.00097.

[308] Tianhao Zhang et al. “Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2016.

[309] Zichao Zhang and Davide Scaramuzza. “Perception-aware Receding Horizon Navigation for MAVs”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018, pp. 2534–2541.

[310] Boyu Zhou et al. “RAPTOR: Robust and Perception-aware Trajectory Replanning for Quadrotor Fast Flight”. In: *CoRR* abs/2007.03465 (2020). arXiv: 2007.03465. URL: https://arxiv.org/abs/2007.03465.

[311] Boyu Zhou et al. “Robust and efficient quadrotor trajectory generation for fast autonomous flight”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3529–3536.

[312] Brady Zhou, Philipp Krähenbühl, and Vladlen Koltun. “Does computer vision matter for action?” In: *Sci. Robotics* 4.30 (2019).

[313] H. Zhou, B. Ummenhofer, and T. Brox. “DeepTAM: Deep Tracking and Mapping”. In: *Eur. Conf. Comput. Vis. (ECCV)*. 2018, pp. 822–838.

[314] Tinghui Zhou et al. “Unsupervised Learning of Depth and Ego-Motion from Video”. In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2017, pp. 6612–6619.

[315] Y. Zhu et al. “Target-driven visual navigation in indoor scenes using deep reinforcement learning”. In: *International Conference on Robotics and Automation (ICRA)*. 2017, pp. 3357–3364.

[316] Yuke Zhu et al. “Reinforcement and Imitation Learning for Diverse Visuomotor Skills”. In: *Proceedings of Robotics: Science and Systems*. 2018.

Antonio Loquercio

Date of birth: 26.03.1992

Education

Feb 17–July 21 **Doctoral program** at the *University of Zurich*, Department of Informatics. Chair of Robotics and Perception Group, Prof. Dr. Davide Scaramuzza

Sep 14–Jan 17 **M.Sc.** in Robotics, System and Control, *ETH Zurich*
Master Thesis: *Efficient Descriptor Learning for Large Scale Localization*
Advisor: Prof. Roland Siegwart

Sep 11–July 14 **B.Sc.** in Advanced Control and Informatics, *Università Tor Vergata, Rome*
Bachelor Thesis: The Unscented and Extended Kalman Filter in Mobile Robotics
Advisor: Prof. Francesco Martinelli

Research Interests

My general research interests are centered around learning-based control of resource-constrained robotics systems, e.g. drones, in complex and possibly dynamic environments, e.g. a city or a forest. Applications of my research include industrial and home automation, research and rescue, and autonomous driving. Topics of recent interest include

1. learning-based control for high-speed flight
2. unsupervised robot-learning via interaction with the environment
3. uncertainty prediction in deep neural networks
4. simulation to real world transfer for robot control

Awards

2020 **Best Paper Award** Honorable Mention, IEEE Transactions on Robotics (T-RO)
(paper: Deep Drone Racing: From Simulation to Reality with Domain Randomization)

2020 **Best Paper Award** Finalist, Conference on Robotics and Intelligent Machines (IRIM-3D)
(paper: Agile Autonomy: High-Speed Flight with On-Board Sensing and Computing)

2020 **Best Paper Award** Finalist, Robotics, Science and Systems (RSS)
(paper: Deep Drone Acrobatics)

2018 **Best System Paper Award**, Conference on Robotics Learning (CORL)
(paper: Deep Drone Racing: Learning Agile Flight in Dynamic Environments)

2017 **ETH Medal** for outstanding master thesis
(awarded to each department's best master theses at ETH Zurich)

Scholarships

2015 Excellence Scholarship and Opportunity Program, ETH Zurich
(ETH most prestigious scholarship awards for master students)
2012–2013 Merit Scholarship Faculty of Engineering, Università Roma Tor Vergata
(awarded annually to the best student of the faculty of Engineering)
2012–2013 Top Ten Students in Engineering Sciences, Università Roma Tor Vergata
(awarded annually to the top 10 students of the Engineering Sciences study program)
2011–2013 Collegio Universitario Lamaro Pozzani Scholarship
(national scholarship covering all living and study costs for university students)
2011 Rotary Club Merit Scholarship
(awarded to the top five high-school graduates in Viterbo, Italy)

Research Experience

Feb 17–current **Graduate Student Researcher** at *University of Zurich* and *ETH Zurich*
at *Robotics and Perception Group*, advised by Davide Scaramuzza

Feb 18–Nov 18 **Graduate Student Researcher** at *University of California Los Angeles*
at *UCLA Vision Lab*, advised by Stefano Soatto

Oct 16–Jan 17 **Machine Learning Research Intern** at *Semeion Research Center, Italy*,
advised by Massimo Buscema

Sep 15–Feb 16 **Student Research Assistant** at *ETH Zurich*,
at *Autonomous Systems Lab*, advised by Roland Siegwart and Marcin Dymczyk

Educational Activities

2021 Guest Lecturer at *Aerial robotics*, EPFL Lausanne
2021 **Lecturer** at *Materials+, The AI PowerBoat Project*, ETH Zurich
2020 Guest Lecturer at *Vision Algorithms for Mobile Robotics*, ETH Zurich
2020 Guest Lecturer at *DSI Studium Digitale*, University of Zurich
2019 Guest Lecturer at *Vision Algorithms for Mobile Robotics*, ETH Zurich
2017-2018 Teaching Assistant at *Vision Algorithms for Mobile Robotics*, ETH Zurich
2016 Teaching Assistant at *Advanced Machine Learning*, ETH Zurich

Advising

Visiting Students

2019 Bianca Sangiovanni
2019 Yuto Suebe

Master Theses

2021 Simone Arreghini
2021 Mario Bonsembiante
2021 Lorenzo Ferrini
2020 Alessandro Saviolo
2019 Daniel Mouritzen
2018 Bojana Nenezic
2017 Yawei Ye

Semester Theses

2019 Mattia Segu
2019 Christoph Meyer
2018 Simon Muntwiler
2018 Moritz Zimmermann

Proposal Writing

I have supported my advisor Davide Scaramuzza in the writing of the following proposals (mainly in the machine learning chapter):

2020 Intel Network for Intelligent Systems Grant (200K USD)
2019 ERC (European Research Council) Consolidator Grant (2.0 M EUR)
2019 Intel Network for Intelligent Systems Grant (200K USD)
2018 Intel Network for Intelligent Systems Grant (170K USD)

Media Coverage

29.06.2020 **Der Spiegel**, Akrobatische Drohnen [PDF]
25.06.2020 **Drones Crunch**, Must Watch! Programming Precision Aerobatics [Link]
24.06.2020 **NCYT**, Acrobacias para drones [Link]
24.06.2020 **ZDNet**, An autonomous daredevil pushes the limits of flight [Link]
24.06.2020 **DailyMail**, Drones all in a spin! AI algorithm enables quadcopters to perform acrobatic manoeuvres like power loops and barrel rolls autonomously [Link]
23.06.2020 **Blick**, Navigationsalgorithmus der Uni Zurich lehrt Drohnen Kunststuecklein [Link]
24.06.2020 **Robohub**, Drones learn acrobatics by themselves [Link]
24.06.2020 **New Atlas**, AI algorithm enables autonomous drones to do barrel rolls and flips [Link]
24.06.2020 **InceptiveMind**, A navigation algorithm enables drones to learn challenging acrobatic maneuvers [Link]
17.06.2020 **DroneDj**, Drones trained to do acrobatics thanks to artificial intelligence [Link]
27.06.2018 **WIRED**, Drones Just Learned to Fly Solo, Racers May Soon Meet Their Match [Link]

14.02.2018 **La Repubblica**, Tra alberi e palazzi ora il drone fa lo slalom [Link]
26.01.2018 Drone Life, DroNet Algorithm Learns From Traffic to Navigate City Streets [Link]
26.01.2018 **The Robot Report**, DroNet Teaches Drones to Autonomously Navigate Cities [Link]
26.01.2018 **ZDNet**, Autonomous high flying drones learn to navigate by watching traffic below [Link]
26.01.2018 **Blick**, Zürcher Algorithmus lenkt Drohnen sicher durch die Stadt [Link]
26.01.2018 **MIT Tech Review**, This drone learned to fly through streets by studying driverless-car data [Link]
25.01.2018 **IEEE Spectrum**, AI-Powered Drone Mimics Cars and Bikes to Navigate Through City Streets [Link]
24.01.2018 **Tages Anzeiger**, Diese Drohne lernt durch Imitation [Link]
24.01.2018 **NZZ**, So kommen Drohnen sicher durch die Stadt [Link]
24.01.2018 **Digital Trends**, The DroNet algorithm teaches drones to navigate city streets like cars [Link]
23.01.2018 **Phys.org**, Drones learn to navigate autonomously by imitating cars and bicycles [Link]
23.01.2018 **Science Daily**, Drones learn to navigate autonomously by imitating cars and bicycles [Link]
23.01.2018 **Alpha Galileo**, Drones learn to navigate autonomously by imitating cars and bicycles [Link]
23.01.2018 **ORF Science**, So kommen Drohnen sicher durch die Stadt [Link]
23.01.2018 **Spektrum.de**, Drohne lernt von Fahrradfahrern [Link]
23.01.2018 **Blick am Abend**, Sicher durch die Stadt [Link]
23.01.2018 **20 Minuten**, Uni macht Drohnenflüge sicherer [Link]

Professional Service

Organizer/Co-Organizer

2021 ICRA Workshop *Perception and Action in Dynamic Environments*, Online, [Link]
2020 AAAI Spring Symposium *Machine Learning for Mobile Robot Navigation in the Wild*, Palo Alto, California

Technical Reviewer

Journals IEEE Transactions on Robotics (T-RO) ● IEEE Robotics and Automation Letters (RA-L) ● Science Robotics ● IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI) ● The International Journal of Robotics Research (IJRR) ● Computer Graphics Forum
Conferences *Robotics:* Robotics: Science and Systems (RSS) ● Conference on Robotics Learning (CORL) ● International Conference on Robotics and Automation (ICRA) ● International Conference on Intelligent Robots and Systems (IROS)
Computer Vision: Computer Vision and Pattern Recognition (CVPR) ● International Conference on Computer Vision (ICCV)
Machine Learning: Conference on Neural Information Processing Systems (NeurIPS) ● International Conference on Machine Learning (ICML)
Books Foundations and Trends in Robotics (Now Publishers)

Invited Speaker

Nov 20 **keynote:** Autonomy Talks, ETH Zurich, [Youtube]
Nov 20 **keynote:** UZH Machine Learning Workshop , [Link]
May 20 **keynote:** Workshop on Perception, Action, and Learning, ICRA (with Davide Scaramuzza) [Youtube]
Apr 20 Workshop Bridging AI and Cognitive Science (BAICS), ICLR
Apr 20 **keynote:** UZH Deep Learning Symposium, Zurich

Nov 19	**keynote:** Zurich Machine Learning Meetup
June 18	Presentation at University of California Los Angeles (UCLA)
May 18	Workshop on Perception, Inference, and Learning, ICRA
Feb 18	Presentation at National University of Singapore (NUS)
Nov 17	**keynote:** Swiss Machine Learning Day, EPFL, Lausanne