



**University of  
Zurich** <sup>UZH</sup>

Department of Informatics

---

# Learning Vision-Based Agile Flight: From Simulation to the Real World

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  
Elia Kaufmann  
from Sins, Switzerland

approved in July 2022

at the request of  
Prof. Dr. Davide Scaramuzza  
Prof. Dr. Vijay Kumar  
Prof. Dr. Wolfram Burgard  
Prof. Dr. Sertac Karaman  
Prof. Dr. Angela Schoellig

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, 20.07.2022

The Chairperson of the Doctoral Board: Prof. Dr. Thomas Fritz

*Systems that strive for perfection  
are the ones most sensitive to disruption.*  
— Sam Denby

To Nicole.

# Acknowledgements

I would like to thank *Prof. Davide Scaramuzza* for accepting me as a PhD student and providing me with many exciting opportunities and helpful advice over the years. It is truly remarkable how much time Davide invests in guiding us students, all while still allowing us freedom in our creativity and choice of projects.

I would also like to particularly thank my colleagues *Antonio Loquercio* and *Philipp Foehn*. I collaborated with them in many projects, throughout which they not only became colleagues, but also good friends. Both Philipp and Antonio were always open for discussing new ideas, sharing their view on challenging problems, or giving valuable feedback also on projects outside their field of expertise.

I would like to express my gratitude to all the current and past members, visitors, and students who I interacted with during my journey as a PhD student at the *Robotics and Perception Group*. I would like to particularly thank Leonard Bauersfeld, Yunlong Song, Mathias Gehrig, Daniel Gehrig, Nico Messikommer, Angel Romero, Drew Hanover, Manasi Muglikar, Giovanni Cioffi, Matthias Fässler, Davide Falanga, Henri Rebecq, Titus Cieslewski, Jeff Delmerico, Guillermo Gallego, Dario Brescianini, Sihao Sun, Robert Penicka, Christian Pfeiffer, Alessandro Simovic, Julien Kohler, Thomas Längle, Manuel Sutter, Alex Barden, and Tamar Tolcachier. I also had the pleasure to work with great students, namely Guillem Torrente, Mats Steinweg, Florian Fuchs, Selim Naji, HaoChih Lin, Yaswanth Mummaneni, Christoph Meyer, Livio Giacomini, Mario Bonsembiante, Jiaxu Xing, Alessandro Saviolo, Simon Muntwiler, and Moritz Zimmermann.

Furthermore, I was fortunate enough to work with great international collaborators, namely René Ranftl, Matthias Müller, Alexey Dosovitskiy, Tim Salzmann, Markus Ryll, and Vladlen Koltun.

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. Sertac Karaman, Prof. Wolfram Burgard, and Prof. Vijay Kumar for reviewing my thesis and their valuable feedback.

Personally, I thank my friends and family for their support and motivation throughout the years and accepting my absences when deadlines came closer. In particular, I'm deeply grateful to *Nicole*, for being always supportive and tolerant with me, for motivating me to reach for the stars, and for standing by me during difficult times.

*Zurich, April 2022*

*Elia Kaufmann*



# Abstract

Autonomous aerial vehicles have a huge industry potential, with an estimated market value of over \$80 billion US-Dollar by the year 2025 according to Forbes [8]. Already during the last decade, commercial drones – both autonomous and remote-controlled – have conquered new markets ranging from agriculture to transport, security, surveillance, inspection, entertainment, and search and rescue [333, 131]. Compared to grounded robots, aerial robots have the distinct advantage of being able to cover large distances in short time, which is an essential capability for tasks where a quick response time is required. In contrast to fixed-wing aerial robots, multirotors, and especially quadrotors, combine the ability to hover in-place with mechanical simplicity, allowing to build highly agile vehicles with a relatively simple mechanical setup. Unfortunately, this unique level of maneuverability comes at a price: even while hovering, multirotors need to exert constant mechanical power to stay airborne. This results in limited battery life and creates the need to push autonomous drones to high speeds in order to maximize the utility of every single battery charge [167, 22].

While human drone pilots achieve impressive maneuvering skills with their powerful drones, commercial autonomous drones today still lack the level of agility demonstrated by human pilots, even when being provided with the same physical specifications such as motors and platform weight. This gap between human-level performance and the capabilities of autonomous vision-based drones has been made obvious when comparing professional human drone racing pilots with autonomous racing drones, as they have been demonstrated at the Lockheed Martin AlphaPilot competition in 2019 [1, 61, 62, 92]. The main reason for this performance gap is two-fold: (i) navigating at such high speeds raises fundamental challenges in perception, planning, and control, as sensory readings become unreliable due to motion blur and limited field of view, and (ii) the modeling complexity of the system, while relatively simple around hover conditions, becomes increasingly complex at high speeds due to difficult-to-model aerodynamic effects such as drag, interaction between rotors, and turbulences.

Prior work on autonomous quadrotor navigation has approached the challenge of vision-based autonomous navigation by separating the system into a sequence of independent compute modules: perception, mapping, planning, and control. While such modularization of the system is beneficial in terms of interpretability and allows to easily exchange modules, it results in a substantial increase in latency from perception to action. Furthermore, the sequential nature of the modules ignores the tight coupling between perception and action, which results in errors being propagated from one module to the next. Apart from the modularization of the autonomy stack, prior work on autonomous quadrotor navigation heavily relies on control techniques such as traditional PID control [83, 78, 330], adaptive approaches such as INDI [336, 329, 269], or model predictive control (MPC) [79, 250, 162]. While these methods have demonstrated impressive feats in controlled environments, they require substantial amount of tuning [207], and are difficult, if not impossible, to scale to

complex dynamics models without large penalties in computation time.

This thesis investigates the deployment of machine learning approaches on a real-world robotic system, allowing for tight coupling of perception and action. Compared to traditional perception, planning, and control approaches, learning-based sensorimotor policies have the distinct advantage of mapping potentially high-dimensional sensory observation directly to control commands, while being able to cope with arbitrary complex observations and dynamics models. These capabilities result in unique strengths of learning-based systems: the direct mapping of observations to actions results in substantially lower latency, while the possibility to combine them with arbitrary observation- and dynamics-models allows to train such policies to cope with complex real world scenarios.

While learning-based methods have already shown impressive performance in the simulation domain, they are often believed to be too compute demanding for successful deployment on resource-constrained platforms such as small autonomous aerial vehicles. In this thesis, I demonstrate that such sensorimotor policies can very well be deployed on small aerial vehicles, and in fact surpass their traditional counterparts with respect to robustness and computational complexity. The research presented in this thesis allowed to not only substantially surpass the previous state of the art in autonomous drone flight, but also resulted in the first autonomous drone that outperformed a human expert pilot in a vision-based drone race.

In summary, the contributions presented in this work aim to answer the question of

*How can the capabilities of machine-learning algorithms be leveraged to push autonomous vision-based quadrotors to a new level of agility?*

The following list summarizes the contributions of this work:

- A system based on model-free reinforcement learning and real-world adaptation for vision-based autonomous drone racing. This system allowed for the first time to outperform a human expert pilot with a robotic system in a vision-based drone race.
- A benchmark comparison of learning-based control policies that investigates the performance and robustness of policies as a function of their output modality. Notably, we compare policies that directly output low-level, single-rotor-thrusts against policies that only output collective thrust and bodyrates.
- A method based on abstraction of sensory inputs to achieve robust transfer of sensorimotor policies between domains, such as between simulation and the real world. The method is then deployed in real-world tasks such as drone racing, acrobatic flight, and high-speed flight in the wild.
- Hybrid aerodynamics models combining first principles with a data-driven residual that models unexplained forces and torques when approaching the physical limits of the platform. These hybrid models are deployed on tasks including accurate simulation and online predictive control.
- A tightly integrated perception system for autonomous drone racing that allows to detect an arbitrary number of racing gates while being robust to partial occlusion.



- A framework for agile quadrotor flight completely open-source and open-hardware. The framework provides modular estimation, planning, and control capabilities, and can also be used in conjunction with learning-based controllers.



# List of Contributions

The \* symbol indicates shared first authorship.

## Journal Publications

- **Elia Kaufmann**, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, Davide Scaramuzza, “Champion-Level Drone Racing Using Deep Reinforcement Learning”, *Nature* (2023), [Appendix H](#)
- Tim Salzmann, **Elia Kaufmann**, Marco Pavone, Davide Scaramuzza, Markus Ryll, “Neural-MPC: Deep Learning Model Predictive Control for Quadrotors and Agile Robotic Platforms”, *IEEE Robotics and Automation Letters (RA-L)* (2022), Links: [PDF](#)
- Robert Penicka, Yunlong Song, **Elia Kaufmann**, Davide Scaramuzza, “Learning Minimum-Time Flight in Cluttered Environments”, *IEEE Robotics and Automation Letters (RA-L)* (2022), Links: [PDF](#)
- Philipp Foehn\*, **Elia Kaufmann\***, Angel Romero, Robert Penicka, Sihao Sun, Leonard Bauersfeld, Thomas Laengle, Yunlong Song, Antonio Loquercio, Davide Scaramuzza, “Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight”, *Science Robotics* (2022), Links: [Webpage](#), [Code](#), [Appendix K](#)
- Sihao Sun, Angel Romero, Philipp Foehn, **Elia Kaufmann**, Davide Scaramuzza, “A Comparative Study of Nonlinear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight”, *IEEE Transactions on Robotics (TRO)* (2022), Links: [PDF](#), [Video](#)
- Antonio Loquercio\*, **Elia Kaufmann\***, René Ranftl, Matthias Müller, Vladlen Koltun, Davide Scaramuzza, “Learning High-Speed Flight in the Wild”, *Science Robotics* (2021), DOI: [10.1126/scirobotics.abg5810](https://doi.org/10.1126/scirobotics.abg5810), Links: [PDF](#), [Video](#), [Appendix F](#)
- Philipp Foehn\*, Dario Brescianini\*, **Elia Kaufmann\***, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, Davide Scaramuzza, “AlphaPilot: Autonomous Drone Racing”, *Springer: Autonomous Robots* (2021), DOI: [10.1007/s10514-021-10011-y](https://doi.org/10.1007/s10514-021-10011-y), Links: [PDF](#), [Video](#), [Talk](#), [Appendix C](#)
- Drew Hanover, Philipp Foehn, Sihao Sun, **Elia Kaufmann**, Davide Scaramuzza, “Performance, Precision, and Payloads: Adaptive Nonlinear MPC for Quadrotors”, *IEEE Robotics and Automation Letters (RA-L)* (2022), DOI: [10.1109/LRA.2021.3131690](https://doi.org/10.1109/LRA.2021.3131690), Links: [PDF](#), [Video](#)
- Guillem Torrente\*, **Elia Kaufmann\***, Philipp Foehn, Davide Scaramuzza, “Data-Driven MPC for Quadrotors”, *IEEE Robotics and Automation Letters (RA-L)* (2021), DOI: [10.1109/LRA.2021.3061307](https://doi.org/10.1109/LRA.2021.3061307), Links: [PDF](#), [Video](#), [Code Appendix I](#)
- Florian Fuchs, Yunlong Song, **Elia Kaufmann**, Davide Scaramuzza, Peter Duerr “Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning”, *IEEE Robotics and Automation Letters (RA-L)* (2020), DOI: [10.1109/LRA.2021.3064284](https://doi.org/10.1109/LRA.2021.3064284), Links: [PDF](#), [Video](#)

## List of Contributions

---

- Antonio Loquercio\*, [Elia Kaufmann\\*](#), René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, Davide Scaramuzza, “Deep Drone Racing: From Simulation to Reality with Domain Randomization”, *IEEE Transactions on Robotics (T-RO)* (2020), (**Best Paper Award Honorable Mention**), DOI: [10.1109/TRO.2019.2942989](https://doi.org/10.1109/TRO.2019.2942989), Links: [PDF](#), [Video](#), [Appendix D](#)

## Peer-Reviewed Conference Papers

- [Elia Kaufmann](#), Leonard Bauersfeld, Davide Scaramuzza, “A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight”, *IEEE International Conference on Robotics and Automation (ICRA)* (2022), Links: [PDF](#), [Video](#), [Appendix G](#)
- Yunlong Song, HaoChih Lin, [Elia Kaufmann](#), Peter Duerr, Davide Scaramuzza, “Autonomous Overtaking in Gran Turismo Sport Using Curriculum Reinforcement Learning”, *IEEE International Conference on Robotics and Automation (ICRA)* (2021), DOI: [10.1109/ICRA48506.2021.9561049](https://doi.org/10.1109/ICRA48506.2021.9561049), Links: [PDF](#), [Video](#)
- Yunlong Song, Mats Steinweg, [Elia Kaufmann](#), Davide Scaramuzza, “Autonomous Drone Racing with Deep Reinforcement Learning”, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2021), DOI: [10.1109/IROS51168.2021.9636053](https://doi.org/10.1109/IROS51168.2021.9636053), Links: [PDF](#), [Video](#)
- Yunlong Song, Selim Naji, [Elia Kaufmann](#), Antonio Loquercio, Davide Scaramuzza, “Flightmare: A Flexible Quadrotor Simulator”, *Conference on Robot Learning (CoRL)* (2020), Links: [PDF](#), [Video](#), [Code](#)
- [Elia Kaufmann\\*](#), Antonio Loquercio\*, René Ranftl, Matthias Müller, Vladlen Koltun, Davide Scaramuzza, “Deep Drone Acrobatics”, *Robotics: Science and Systems (RSS)* (2020), (**Best Paper Award Finalist**), DOI: [10.15607/RSS.2020.XVI.040](https://doi.org/10.15607/RSS.2020.XVI.040), Links: [PDF](#), [Video](#), [Code Talk](#), [Appendix E](#)
- Leonard Bauersfeld\*, [Elia Kaufmann\\*](#), Philipp Foehn, Sihao Sun, Davide Scaramuzza, “NeuroBEM: Hybrid Aerodynamic Quadrotor Model”, *Robotics: Science and Systems (RSS)* (2021), DOI: [10.15607/RSS.2021.XVII.042](https://doi.org/10.15607/RSS.2021.XVII.042), Links: [PDF](#), [Video](#), [Code Appendix J](#)
- Philipp Foehn\*, Dario Brescianini\*, [Elia Kaufmann\\*](#), Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, Davide Scaramuzza, “AlphaPilot: Autonomous Drone Racing”, *Robotics: Science and Systems (RSS)* (2020) (**Best System Paper Award**), DOI: [10.15607/RSS.2020.XVI.081](https://doi.org/10.15607/RSS.2020.XVI.081), Links: [PDF](#), [Video](#), [Talk](#), [Appendix C](#)
- [Elia Kaufmann](#), Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, Davide Scaramuzza, “Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing”, *IEEE International Conference on Robotics and Automation (ICRA)* (2019), DOI: [10.1109/ICRA.2019.8793631](https://doi.org/10.1109/ICRA.2019.8793631), Links: [PDF](#), [Video](#), [Appendix B](#)
- [Elia Kaufmann\\*](#), Antonio Loquercio\*, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, Davide Scaramuzza, “Deep Drone Racing: Learning Agile Flight in Dynamic Environments”, *Conference on Robot Learning (CoRL)* (2018) (**Best System Paper Award**), Links: [PDF](#), [Video](#), [Appendix A](#)
- Titus Cieslewski, [Elia Kaufmann](#), Davide Scaramuzza, “Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight”, *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), (**Best Search and Rescue Robotics Paper Award Finalist**), DOI: [10.1109/IROS.2017.8206030](https://doi.org/10.1109/IROS.2017.8206030), Links: [PDF](#), [Video](#), [Appendix L](#)

## Awards

- *Robotics: Science and Systems (RSS) 2020*, **Best System Paper Award** for Paper C and invitation to publish in *Springer: Autonomous Robots*.
- *IEEE Transactions on Robotics, 2020*, **King-Sun Fu Memorial Best Paper Award (Honorable Mention)** for Paper D
- *Robotics: Science and Systems (RSS) 2020*, **Best Paper Award (Honorable Mention)** for Paper E
- *AlphaPilot Challenge, 2019*, organized by *Lockheed Martin* and the *Drone Racing League*, **2nd Place out of 430 participants worldwide** with the approach presented in Paper C.
- *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2018* in the *Autonomous Drone Racing (ADR)* challenge, **Winner 1st Place** with the approach presented in Paper B.
- *Conference on Robot Learning (CoRL) 2018*, **Best System Paper Award**, for Paper A and invitation to publish in *IEEE Transactions on Robotics*.
- *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2017*, **Best Search and Rescue Robotics Paper Award (Honorable Mention)**, for Paper L.

## Open-source Software

- **Agilicious Flightstack**: Paper K  
available at <https://agilicious.dev>
- **Learning High-Speed Flight in the Wild**: Paper F  
available at [https://github.com/uzh-rpg/agile\\_autonomy](https://github.com/uzh-rpg/agile_autonomy)
- **Deep Drone Acrobatics**: Paper E  
available at [https://github.com/uzh-rpg/deep\\_drone\\_acrobatics](https://github.com/uzh-rpg/deep_drone_acrobatics)
- **Deep Drone Racing**: Paper D  
available at [https://github.com/uzh-rpg/sim2real\\_drone\\_racing](https://github.com/uzh-rpg/sim2real_drone_racing)
- **Data-Driven MPC for Quadrotors**: Paper I  
available at [https://github.com/uzh-rpg/data\\_driven\\_mpc](https://github.com/uzh-rpg/data_driven_mpc)



# Contents

<b>Acknowledgements</b>	<b>v</b>
<b>Abstract</b>	<b>vii</b>
<b>List of Contributions</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	4
1.1.1 Advantages . . . . .	6
1.1.2 Challenges . . . . .	8
1.2 Related Work . . . . .	10
1.2.1 Autonomous Navigation: External Sensing . . . . .	11
1.2.2 Autonomous Navigation: Onboard Sensing and Computation . . . . .	11
1.2.3 Data-Driven Dynamics Models . . . . .	14
1.2.4 Simulation . . . . .	16
1.2.5 Hardware Platforms . . . . .	19
1.2.6 Simulation-to-Reality Transfer . . . . .	19
1.2.7 Autonomous Drone Racing . . . . .	20
<b>2 Contributions</b>	<b>23</b>
2.1 Tight Coupling of Learning-Based Perception and Optimal Planning and Control . . . . .	25
2.1.1 Paper A: Deep Drone Racing: Learning Agile Flight in Dynamic Environments . . . . .	26
2.1.2 Paper B: Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing . . . . .	27
2.1.3 Paper C: AlphaPilot: Autonomous Drone Racing . . . . .	28
2.2 Simulation-to-Reality Transfer for Agile Drone Flight . . . . .	29
2.2.1 Paper D: Deep Drone Racing: From Simulation to Reality with Domain Randomization . . . . .	30
2.2.2 Paper E: Deep Drone Acrobatics . . . . .	31
2.2.3 Paper F: Learning High-Speed Flight in the Wild . . . . .	32
2.2.4 Paper G: A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight . . . . .	34
2.2.5 Paper H: Champion-Level Performance in Drone Racing using Deep Reinforcement Learning . . . . .	35
2.3 Data-Driven Dynamics Models . . . . .	36
2.3.1 Paper I: Data-Driven MPC for Quadrotors . . . . .	37
2.3.2 Paper J: NeuroBEM: Hybrid Aerodynamic Quadrotor Model . . . . .	38
	xv

## Contents

---

2.4	Additional Contributions . . . . .	39
2.4.1	Paper K: Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight . . . . .	40
2.4.2	Paper L: Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight . . . . .	41
<b>3</b>	<b>Future Directions</b>	<b>43</b>
<b>A</b>	<b>Deep Drone Racing: Learning Agile Flight in Dynamic Environments</b>	<b>45</b>
A.1	Introduction . . . . .	47
A.2	Related Work . . . . .	48
A.3	Method . . . . .	49
A.3.1	Training procedure . . . . .	50
A.4	Experiments in Simulation . . . . .	51
A.4.1	Comparison to end-to-end learning approach . . . . .	51
A.4.2	Performance on a complex track . . . . .	52
A.4.3	Generalization to dynamic environments . . . . .	53
A.5	Experiments in the Physical World . . . . .	54
A.5.1	Experiments on a race track . . . . .	55
A.6	Discussion . . . . .	56
<b>B</b>	<b>Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing</b>	<b>59</b>
B.1	Introduction . . . . .	61
B.2	Related Work . . . . .	62
B.3	Methodology . . . . .	63
B.3.1	Notation and Frame Convention . . . . .	64
B.3.2	Perception System . . . . .	64
B.3.3	Mapping System . . . . .	66
B.3.4	Planning and Control System . . . . .	67
B.4	Experimental Setup . . . . .	68
B.4.1	Simulation . . . . .	68
B.4.2	Physical System . . . . .	69
B.5	Results . . . . .	69
B.5.1	Simulation . . . . .	69
B.5.2	Physical System . . . . .	70
B.6	Conclusion . . . . .	71
<b>C</b>	<b>AlphaPilot: Autonomous Drone Racing</b>	<b>73</b>
C.1	Introduction . . . . .	75
C.1.1	Motivation . . . . .	75
C.1.2	Related Work . . . . .	75
C.1.3	Contribution . . . . .	77
C.2	AlphaPilot Race Format and Drone . . . . .	77
C.2.1	Race Format . . . . .	77
C.2.2	Drone Specifications . . . . .	78
C.2.3	Drone Model . . . . .	78



---

C.3	System Overview . . . . .	79
C.3.1	Perception . . . . .	79
C.3.2	State Estimation . . . . .	80
C.3.3	Planning and Control . . . . .	80
C.3.4	Software Architecture . . . . .	80
C.4	Gate Detection . . . . .	81
C.4.1	Stage 1: Predicting Corner Maps and Part Affinity Fields . . . . .	82
C.4.2	Stage 2: Corner Association . . . . .	83
C.4.3	Training Data . . . . .	84
C.4.4	Network Architecture and Deployment . . . . .	84
C.5	State Estimation . . . . .	84
C.5.1	Measurement Modalities . . . . .	86
C.6	Path Planning . . . . .	87
C.6.1	Time-Optimal Motion Primitive . . . . .	88
C.6.2	Sampling-Based Receding Horizon Path Planning . . . . .	88
C.6.3	Path Parameterization . . . . .	89
C.7	Control . . . . .	90
C.7.1	Position Control . . . . .	90
C.7.2	Attitude Control . . . . .	91
C.8	Results . . . . .	91
C.8.1	Gate Detection . . . . .	92
C.8.2	State Estimation . . . . .	93
C.8.3	Planning and Control . . . . .	93
C.9	Discussion and Conclusion . . . . .	94
<b>D Deep Drone Racing: From Simulation to Reality with Domain Randomization</b>		<b>95</b>
D.1	Introduction . . . . .	97
D.2	Related Work . . . . .	98
D.2.1	Data-driven Algorithms for Autonomous Navigation . . . . .	99
D.2.2	Drone Racing . . . . .	99
D.2.3	Transfer from Simulation to Reality . . . . .	100
D.3	Method . . . . .	100
D.3.1	Training Procedure . . . . .	101
D.3.2	Trajectory Generation . . . . .	104
D.4	Experiments . . . . .	106
D.4.1	Experimental Setup . . . . .	106
D.4.2	Experiments in Simulation . . . . .	107
D.4.3	Analysis of Accuracy and Efficiency . . . . .	111
D.4.4	Experiments in the Real World . . . . .	112
D.4.5	Simulation to Real World Transfer . . . . .	116
D.5	Discussion and Conclusion . . . . .	118

<b>E</b>	<b>Deep Drone Acrobatics</b>	<b>121</b>
E.1	Introduction . . . . .	123
E.2	Related Work . . . . .	124
E.3	Overview . . . . .	125
E.4	Method . . . . .	126
E.4.1	Reference Trajectories . . . . .	127
E.4.2	Privileged Expert . . . . .	128
E.4.3	Learning . . . . .	129
E.4.4	Sensorimotor Controller . . . . .	131
E.4.5	Implementation Details . . . . .	133
E.5	Experiments . . . . .	134
E.5.1	Experimental Setup . . . . .	134
E.5.2	Experiments in Simulation . . . . .	135
E.5.3	Deployment in the Physical World . . . . .	136
E.6	Conclusion . . . . .	137
<b>F</b>	<b>Learning High-Speed Flight in the Wild</b>	<b>139</b>
F.1	Introduction . . . . .	141
F.2	Results . . . . .	144
F.2.1	High-Speed Flight in the Wild . . . . .	144
F.2.2	Controlled Experiments . . . . .	148
F.2.3	Computational Cost . . . . .	151
F.2.4	The Effect of Latency and Sensor Noise . . . . .	152
F.3	Discussion . . . . .	154
F.4	Materials and Methods . . . . .	155
F.4.1	The Privileged Expert . . . . .	157
F.4.2	The Student Policy . . . . .	158
F.4.3	Training Environments . . . . .	161
F.4.4	Method Validation . . . . .	161
F.5	Experimental Platform . . . . .	162
F.6	Computational Complexity . . . . .	164
F.7	Rotational Dynamics . . . . .	164
F.8	Metropolis-Hastings Sampling . . . . .	165
<b>G</b>	<b>A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight</b>	<b>169</b>
G.1	Introduction . . . . .	171
G.2	Related Work . . . . .	172
G.3	Quadrotor Dynamics . . . . .	174
G.3.1	Notation . . . . .	174
G.3.2	Quadrotor Dynamics . . . . .	174
G.4	Methodology . . . . .	175
G.4.1	Observations, Actions, and Rewards . . . . .	176
G.4.2	Policy Learning . . . . .	177
G.4.3	Training Details . . . . .	178
G.5	Experiments . . . . .	178
G.5.1	Simulation Experiments . . . . .	178

G.5.2	Real World Experiments . . . . .	182
G.6	Conclusion . . . . .	182
G.7	Supplementary Material . . . . .	182
G.7.1	MPC Baselines . . . . .	182
G.7.2	Ablation Studies . . . . .	183
G.7.3	Tracking Performance . . . . .	186
G.7.4	Reference Trajectories . . . . .	186
<b>H</b>	<b>Champion-Level Drone Racing using Deep Reinforcement Learning</b>	<b>191</b>
H.1	Introduction . . . . .	193
H.2	Results . . . . .	197
H.3	Methods . . . . .	200
H.3.1	Quadrotor Simulation . . . . .	200
<b>I</b>	<b>Data-Driven MPC for Quadrotors</b>	<b>211</b>
I.1	Introduction . . . . .	213
I.2	Related Work . . . . .	215
I.3	Methodology . . . . .	216
I.3.1	Notation . . . . .	216
I.3.2	Nominal Quadrotor Dynamics Model . . . . .	217
I.3.3	Gaussian Process-Augmented Dynamics . . . . .	217
I.3.4	MPC Formulation . . . . .	218
I.3.5	Practical Implementation . . . . .	219
I.3.6	Data Collection and Model Learning . . . . .	220
I.4	Experiments and Results . . . . .	220
I.4.1	Experimental Setup . . . . .	220
I.4.2	Experiments in Simulation . . . . .	222
I.4.3	Experiments in the Real World . . . . .	224
I.5	Conclusion . . . . .	226
<b>J</b>	<b>NeuroBEM: Hybrid Aerodynamic Quadrotor Model</b>	<b>229</b>
J.1	Introduction . . . . .	231
J.2	Related Work . . . . .	233
J.3	Quadrotor Model . . . . .	234
J.3.1	Notation . . . . .	234
J.3.2	Quadrotor Dynamics . . . . .	235
J.3.3	Rotor Model: Quadratic . . . . .	235
J.3.4	Rotor Model: BEM . . . . .	236
J.3.5	Learned Residual Dynamics . . . . .	241
J.4	Experimental Setup . . . . .	241
J.4.1	Data Collection . . . . .	241
J.4.2	Quadrotor Platform . . . . .	242
J.4.3	Control System . . . . .	242
J.4.4	Simulator Extension . . . . .	243
J.5	Experiments and Results . . . . .	243
J.5.1	Experimental Setup . . . . .	243
J.5.2	Comparison of Predictive Performance . . . . .	244

## Contents

---

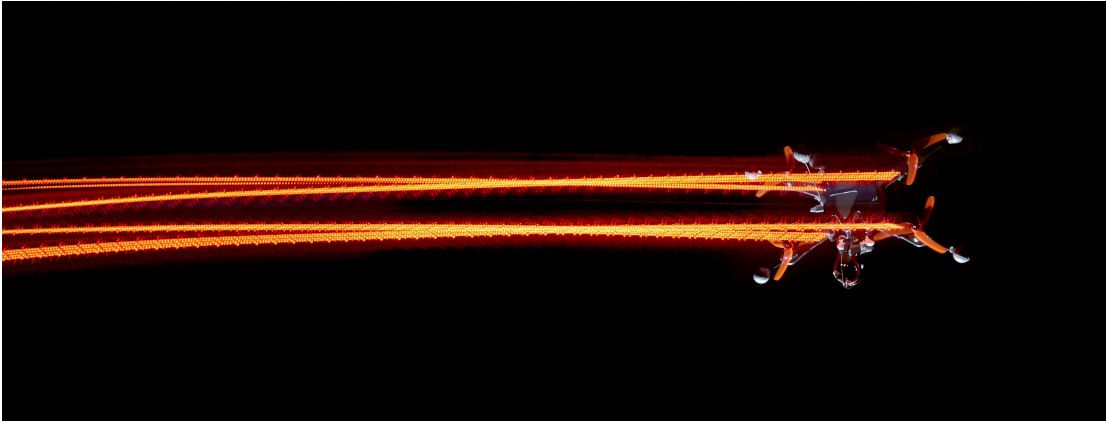
J.5.3	Closed-Loop Comparison . . . . .	246
J.6	Discussion . . . . .	247
J.7	Conclusion . . . . .	249
<b>K</b>	<b>Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight</b>	<b>251</b>
K.1	Introduction . . . . .	253
K.2	Results . . . . .	259
K.2.1	Agile Flight in a Tracking Arena . . . . .	259
K.2.2	Hardware in the Loop Simulation . . . . .	261
K.2.3	Vision-based Agile Flight with Onboard Sensing and Computation	263
K.3	Discussion . . . . .	267
K.4	Materials and Methods . . . . .	269
K.4.1	Compute Hardware . . . . .	269
K.4.2	Flight Hardware . . . . .	271
K.4.3	The Agilicious Flight Stack Software . . . . .	272
<b>L</b>	<b>Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight</b>	<b>277</b>
L.1	Introduction . . . . .	279
L.2	Related work . . . . .	280
L.3	Flight velocity for optimal energy use . . . . .	281
L.4	Methodology . . . . .	283
L.5	Experiments . . . . .	285
L.5.1	Simulation . . . . .	285
L.5.2	Real World Experiments . . . . .	288
L.5.3	Measurements . . . . .	288
L.6	Results . . . . .	290
L.6.1	Simulation . . . . .	290
L.6.2	Real World Experiments . . . . .	292
L.7	Conclusion . . . . .	294
	<b>Bibliography</b>	<b>295</b>
	<b>Curriculum Vitae</b>	<b>325</b>

# 1 Introduction

This thesis presents novel methods for modeling, planning, and control for autonomous, vision-based quadrotor vehicles. Focusing on real-world applications, all approaches presented in this thesis are designed to be deployed on compute-constrained small aerial vehicles. Such focus necessarily restricts the set of applicable algorithms, demands co-design of software and hardware, and requires exploiting specialized compute components, as strict real-time constraints have to be satisfied. Deploying algorithms on high-speed autonomous machines necessitates a high level of robustness to account for the unavoidable mismatch between simulation and the real robotic system. The approaches presented in this thesis aim to close the gap between the impressive piloting skills demonstrated by human drone pilots and their autonomous counterparts. While this feat has recently been achieved with the help of highly accurate external tracking systems [91], achieving the same performance with only onboard sensing and computation remained unsolved.

As it allows for simple and objective comparison to human expert pilots, one recurring demonstrator used in this work is the task of *drone racing*. Drone racing is an emerging sport where pilots race against each other with remote-controlled quadrotors while being provided a first-person-view (FPV) video stream from a camera mounted on the drone. The level of performance demonstrated even by non-professional human pilots made the gap between small autonomous aerial systems and human pilots obvious, with the human pilot at the 2019 AlphaPilot challenge outracing the fastest autonomous team by a factor of two [61]. As of the year 2017, at the start of this thesis, this performance gap between autonomous agents and human pilots exceeded one order of magnitude. By the end of this thesis, human pilots have first been outraced by a fully autonomous vision-based quadrotor.

Pushing an autonomous quadrotor to high speeds while only relying on onboard sensing and computation is extremely difficult and raises fundamental challenges in robotics regarding perception, planning, and control [366]. These challenges arise from the fact that sensory readings become unreliable due to motion blur, the limited field of view of the onboard camera necessitates tight coupling of perception and action, and real-time constraints become ever-more strict the higher the speed of the platform. Furthermore, the modeling complexity of the system, while relatively simple around hover conditions, becomes increasingly complex at high speeds due to difficult-to-model aerodynamic effects such as drag, interaction between rotors, and turbulences.



**Figure 1.1** – An autonomous quadrotor traveling at high speeds during an autonomous drone race. Drone racing is an emerging sport where pilots race against each other with remote-controlled quadrotors while being provided a first-person-view (FPV) video stream from a camera mounted on the drone. As it allows for simple and objective comparison to human expert pilots, drone racing is used as a recurring demonstrator in this work.

Data-driven approaches have the potential to overcome many of these challenges, but come with their own set of obstacles, such as: requiring large amounts of training data, limited interpretability of the predictions, and the tendency of learning-based approaches to overfit to their training domain, rendering transfer between domains challenging.

Prior work on autonomous quadrotor navigation has approached the challenge of vision-based autonomous navigation by separating the system into a sequence of independent compute modules [50, 295, 218, 311, 170, 303]: perception, mapping, planning, and control (Figure 1.3). While such modularization of the system is beneficial in terms of interpretability and allows to easily exchange modules, it induces a substantial increase in latency from perception to action. Furthermore, the sequential nature of the modules ignores the tight coupling between perception and action, which causes errors being propagated from one module to the next. Prior work on autonomous quadrotor navigation heavily relies on control techniques such as PID control [83, 78, 330], adaptive approaches such as INDI [336, 329, 269], or model predictive control (MPC) [79, 250, 162]. While these methods have demonstrated impressive feats in controlled environments [83, 91, 250, 330, 329], they require substantial amount of tuning [207], and are difficult, if not impossible, to scale to complex dynamics models without high cost in computation time.

This thesis investigates the integration of learning-based approaches into a real-world robotic system, allowing for tight coupling of perception and action. Compared to traditional perception, planning, and control approaches, learning-based sensorimotor policies have the distinct advantage of mapping potentially high-dimensional sensory observations directly to control commands, while being able to cope with arbitrary complex observation- and dynamics models. Although learning-based methods have already shown impressive performance in the simulation domain [230, 316, 353], these demonstrators have circumvented one of the fundamental challenges of real-world robotics: imperfect perception. Apart from this restriction to simulation-based demonstrators, learning-based



**Figure 1.2** – A vision-based quadrotor developed in this work, navigating autonomously in the forest. The quadrotor can navigate towards given waypoints at speeds up to  $40 \text{ km h}^{-1}$ , without the need to build an accurate 3D map of its surroundings. This capability is enabled by deploying a machine-learning approach directly on the quadrotor: a deep neural network is fed observations from an onboard camera, an estimate of the platform state, and a reference direction to compute a collision-free receding-horizon trajectory. Employing such a data-driven methodology allows to achieve a low-latency perception-action loop while maintaining robustness to sensor noise.

methods are often believed to be too compute demanding for successful deployment on resource-constrained platforms such as small autonomous aerial vehicles. In this thesis, I demonstrate that such sensorimotor policies can very well be deployed on small aerial vehicles, and in fact outperform their traditional counterparts with respect to robustness and peak performance, resulting in the first autonomous drone that achieves speeds of  $40 \text{ km h}^{-1}$  in cluttered environments such as forests (Figure 1.2), and outperforms a human expert pilot in a vision-based drone race.

Even though machine learning appears to trump expert knowledge in some domains [154], I want to emphasize that in my experience, learning-based algorithms do not make traditional robotics knowledge or hardware design obsolete. In fact, the exact opposite seems to be true: the most capable systems that I have worked on throughout my journey as a Ph.D. student were the ones where an elegant combination of data-driven approaches, traditional algorithms, and careful hardware design has been achieved. One example of this interplay of traditional robotics techniques and novel learning-based methods is deep reinforcement learning: Control policies can be trained via deep reinforcement learning without in-depth understanding of the physics governing the quadrotor platform. However, to design a learning environment that allows successful simulation-to-reality transfer requires the capability to accurately model the physical system, which in turn necessitates a deep understanding of the robotic system. Similarly, system-oriented hardware design is an integral part of designing autonomous robots. The co-design of software and hardware greatly benefits the capability of the final robot, and simply optimizing sensor choice and sensor placement can often result in an entirely different problem to be solved.

The contributions presented in this thesis are grouped into three topics. Each topic is discussed in its own part of the thesis. In the first part, I present methodologies to tightly

couple perception and action in the context of high-speed autonomous flight. These methods combine learning-based perception with traditional planning and control and are demonstrated on the task of autonomous drone racing. In the second part, I propose approaches to efficiently transfer learning-based sensorimotor policies between domains, for example between simulation and the real world. I demonstrate how the choice of input and output representation of a learning-based policy impacts its performance and robustness when exposed to variations in the observations and dynamics. These experiments are conducted on a set of tasks such as high-speed navigation in cluttered environments, acrobatic flight, and drone racing. Finally, I present methods to perform data-driven modeling of quadrotor vehicles and its application for accurate simulation, real-time predictive control, and real-world adaptation of learning-based policies trained via deep reinforcement learning.

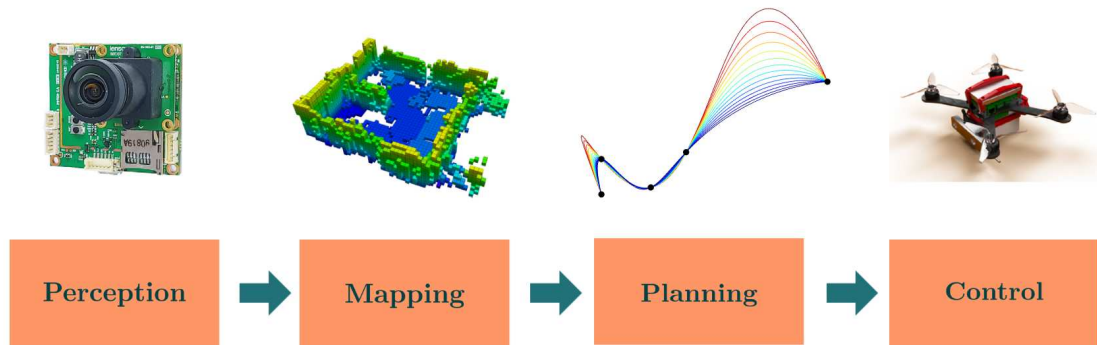
All topics address the fundamental research question of this work - *How can the capabilities of machine-learning algorithms be leveraged to push autonomous vision-based quadrotors to a new level of agility?* - from a different perspective but with the same objective: creating autonomous drones that can navigate at the level of professional human pilots.

This thesis is structured in the form of a collection of papers. An introductory section that highlights the concepts and ideas of each paper is followed by self-contained publications in the appendix. Section 1.1 states and motivates the research objectives of this work. Section 1.2 connects this research to prior work. Chapter 2 presents the contributions of the papers in the appendix and illustrates their connections with respect to each other. Finally, Chapter 3 highlights possible future research directions.

### 1.1 Motivation

Autonomous aerial vehicles have a huge industry potential, with an estimated market value of over \$80 billion US-Dollar by the year 2025 according to Forbes [8]. Already during the last decade, commercial drones – both autonomous and remote-controlled – have conquered new markets ranging from agriculture to transport, security, surveillance, inspection, entertainment, and search and rescue [333, 131]. Compared to grounded robots, aerial robots have the distinct advantage of being able to cover large distances in short time, which is an essential capability for tasks where a quick response time is required. In contrast to fixed-wing aerial robots, multirotors, and especially quadrotors, combine the ability to hover in-place with mechanical simplicity, allowing to build highly agile vehicles with a relatively simple mechanical setup. Unfortunately, this unique level of maneuverability comes at a price: even while hovering, multirotors need to excerpt constant mechanical power to stay airborne. This results in limited battery life and creates the need to push autonomous drones to high speeds in order to maximize the utility of every single battery charge [167, 22]. The desired level of agility is impressively demonstrated by professional human drone pilots, who manage to navigate through complex environments and pass through small openings even while traveling at high speeds.



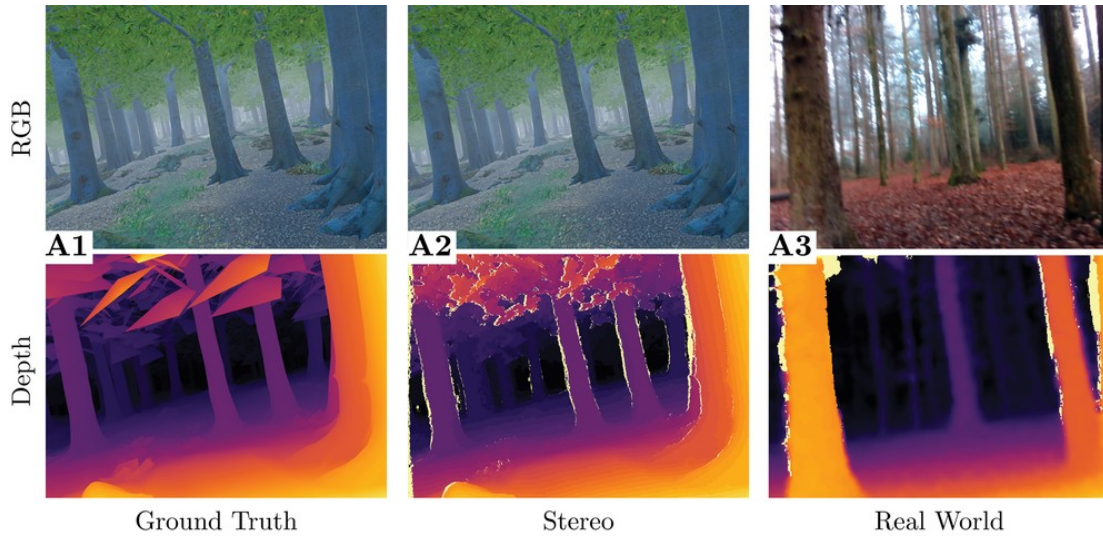


**Figure 1.3** – Prior work on autonomous quadrotor flight has approached the task of vision-based flight by separating the system into a sequence of independent compute modules consisting of perception, mapping, planning, and control. While such modularization of the system is beneficial in terms of interpretability, enables engineering teams to work on multiple modules in parallel, and allows to easily exchange modules, it results in a substantial increase in latency from perception to action. Additionally, the sequential nature of the modules ignores the tight coupling between perception and action, which causes errors being propagated from one module to the next.

Despite progress in recent years, today’s autonomous drones are still far from the performance demonstrated by human operators. Two main factors have blocked progress towards human-level performance in autonomous drone flight: (i) the hardship to effectively perceive the robot’s surroundings and (ii) the difficulty to capture an accurate model of the platform. Both challenges emerge from the high speeds and extreme accelerations experienced during agile flight.

Prior work on autonomous quadrotor flight has approached the task of vision-based flight by separating the system into a sequence of independent compute modules consisting of perception, mapping, planning, and control, as illustrated in Figure 1.3. While such modularization of the system is beneficial in terms of interpretability, enables engineering teams to work on multiple modules in parallel, and allows to easily exchange modules, it results in a substantial increase in latency from perception to action. Furthermore, the sequential nature of the modules ignores the tight coupling between perception and action, which results in errors being propagated from one module to the next. Finally, current approaches require to make simplifying assumptions about the robot dynamics and observation models to maintain real-time capability. These assumptions constrain the maximum achievable agility of the robot, as both dynamics models as well as observation models become increasingly erroneous the higher the platform speed.

In contrast to the traditional approach that divides the autonomy pipeline in a sequence of independent modules, recent learning-based approaches instead propose to replace parts of – or even the entire – pipeline with data-driven approaches that potentially directly map from sensory observation to command. Such approaches come with a set of advantages but also challenges, which are discussed in the next part.



**Figure 1.4** – All sensory information in the real world is corrupted by noise and sensor-specific failure modes. This figure illustrates simulated and real-world sensory data recorded from an RGB camera (top row), and a depth camera (bottom row). In the real world, data from the RGB camera is corrupted by motion blur and limited dynamic range. The information from the depth sensor suffers from missing information along the boundaries of objects due to occlusion. Data-driven approaches allow to be robust to such failure cases without compromising agility by training specifically on such corrupted data.

### 1.1.1 Advantages

Data-driven methods represent a holistic approach for tight coupling of perception and action, as they can directly process high-dimensional perception input. As such, enhancing, or even replacing, parts of the robotic cycle with data-driven components potentially offers substantial improvements with respect to peak performance, required compute power, and robustness to imperfect sensing and modeling.

**Representational Capacity.** Compared to traditional planning and control approaches, learning-based policies, trained via imitation learning or reinforcement learning, have greater flexibility in terms of input representations and underlying dynamics models. They can map potentially high-dimensional sensory observations such as images and IMU readings to control commands, without the need to feed the image information through a sequence of modules such as mapping, planning, and control. Furthermore, due to their universal approximation capabilities, policies represented by deep neural networks can be designed to operate with arbitrary complex underlying dynamics models: a policy trained via deep reinforcement learning does not require differentiable dynamics, and a policy trained via behavioural cloning can imitate an arbitrary complex algorithm that might even not run in real time. This is in stark contrast to state of the art model-based planning and control, which either requires to make simplifying assumptions of the dynamics to be run in real-time, or resorts to sampling-based approaches, which greatly increase the computation time.



**Figure 1.5** – Neural network-based approaches can be efficiently computed on specialized hardware such as graphics processing units (GPUs), tensor processing units (TPUs), or even field programmable gate arrays (FPGAs). Examples of such hardware accelerators include the NVIDIA Jetson TX2 (A), which contains a built-in GPU; the Intel Movidius Neural Compute Stick (B), which contains a specialized vision processing unit optimized for neural network inference; the Google Coral board (C), which contains an integrated TPU; and the TySOM development board (D) combining an FPGA with an ARM Cortex processor.

**Robustness to Imperfect Perception.** The perfect sensor on a mobile robot does not exist. Be it an RGB camera mounted on a quadrotor, or an IMU mounted on a ground robot, all sensors exhibit their distinct failure modes. In traditional robotics, these failure modes are typically modeled using Gaussian noise. While being an effective approach that can be elegantly combined with optimal estimation approaches such as Kalman filters, the Gaussian noise assumption makes an oversimplification to the sensor’s error characteristics, which ultimately results in degraded performance when the system is pushed to its limits. In contrast, data-driven approaches can be trained on such corrupted sensory observation, either from scratch or by fine-tuning on limited real-world data. Since the failure modes of sensors generally have a systematic nature, sensorimotor policies can learn to account for them by identifying the regularities in the data, resulting in robust policies that can be deployed in the real world. I made use of this property of learning-based approaches to train deep sensorimotor policies that can perform acrobatics (Paper E) or high-speed flight in unknown, unstructured environments (Paper F).

**Low Latency.** High-speed navigation in previously unknown environment requires a fast perception-action loop. Traditional methods that separate the autonomy stack into a sequence of modules generally suffer from high latency, as these modules are executed *in sequence*, which results in compound latency. In contrast, data-driven approaches can directly map raw sensory observations to navigation commands, shortcutting the sequential stack of modules typical to the traditional approaches. This direct mapping allows for substantially faster execution times and lower latency, as shown in Paper F. The effect of reduced latency of learning-based approaches is magnified for deep neural network-based approaches, which can be efficiently computed on graphics processing units or other specialized hardware such as field programmable gate arrays (FPGAs) or tensor processing units (TPUs), as illustrated in Figure 1.5.

### 1.1.2 Challenges

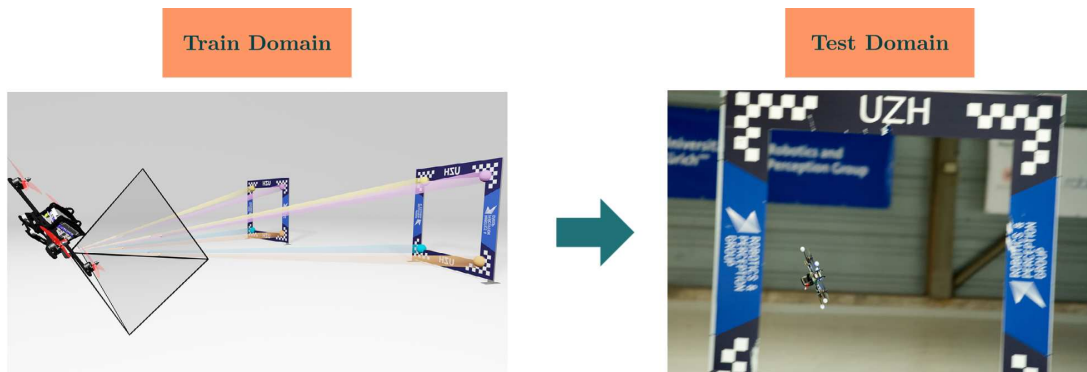
Although learning-based systems have promising capabilities compared to traditional methods for autonomous navigation, they also come with several new challenges that need to be resolved in order to exploit their full potential. The most prominent challenge arises from the sample complexity of these methods. Without access to a sufficiently large and diverse dataset, neural network-based approaches tend to overfit to their training dataset, leading to poor performance when deployed in a test scenario. However, even when a sufficiently large training set is available, transferring data-driven algorithms between domains is known to be hard and requires careful consideration regarding input and output modalities, training strategy, and network architecture.

**Training Data.** Learning-based approaches require a substantial amount of data to be trained. In the case of imitation learning, generating a sufficient amount of high-quality labeled data is a tedious and time-consuming process, often requiring manual labeling by a human expert. Conversely, reinforcement learning does not require such labeled training samples, but requires even more extensive interaction with the robot to be trained. To overcome these challenges, the community has shifted towards training data-driven approaches entirely in simulation. Simulation is fast, cheap, and safe. It allows to generate data using multiple simulated platforms in parallel, crashes do not require to repair a physical robot and no human operator is in danger by operating a partially trained policy on a real platform. However, training a sensorimotor policy in simulation and deploying it in the real world requires to bridge the simulation-to-reality gap, a challenge laid out in the next paragraph.

I have tackled the problem of training data generation by focusing on training policies entirely in simulation (Paper E, Paper D, Paper F, Paper G). Furthermore, I contributed towards ongoing efforts to improve the accuracy of quadrotor aerodynamics simulation (Paper J) in order to facilitate simulation-to-reality transfer of even the most agile control policies (Paper H).

**Domain Transfer.** Due to the high sample complexity of learning-based policies, they are often trained in simulation, which then requires transferring the policy from simulation to the real world. This transfer between domains is known to be hard and is typically approached by increasing the simulation fidelity [337, 23], by randomization of dynamics [234, 12] or rendering properties [298, 341] at training time, or by abstraction of the policy inputs [172, 209]. Apart from simulation enhancements and input abstractions, also the choice of action space of the learned policy itself can facilitate transfer. Policies that generate high-level commands, such as desired linear velocity or future waypoints [209], have a reduced simulation-to-reality gap, as they abstract the task of flying by relying on an existing underlying control stack. However, while facilitating transfer, such abstractions also constrain the maneuverability of the platform [169].

Enabling domain transfer for deep sensorimotor policies is one of the defining subjects of this thesis. I propose to tackle the problem by leveraging abstractions of the sensory inputs (Paper E, Paper F), to perform real-world adaptation of the policy using limited data from the test domain (Paper H), and to optimize the action space of the policy to

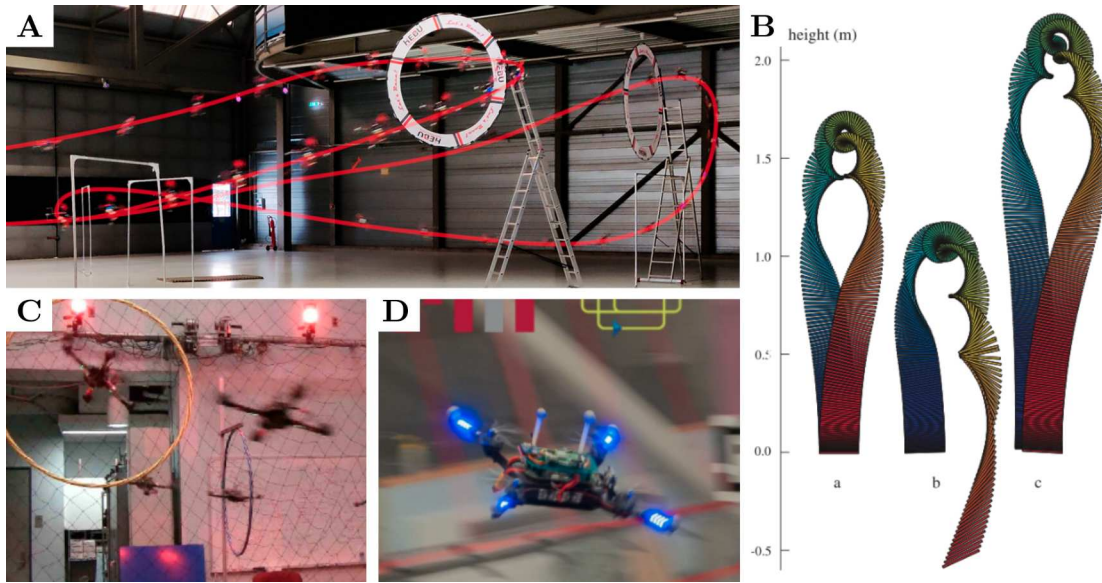


**Figure 1.6** – Training sensorimotor policies typically requires large and diverse datasets for training. Generating such datasets in the real world is a tedious process, often requiring human experts to annotate data. Generating such data in simulation is an appealing alternative, as it allows to generate virtually unlimited data without the need for manual annotation. However, training sensorimotor policies in simulation requires to bridge the simulation-to-reality gap, as even the most versatile and accurate simulators cannot exactly reproduce data observed on the real robot.

maximize robustness without compromising agility (Paper G).

**Interpretability.** Designing and developing a robotic system is often an iterative task, where algorithm hyperparameters and design decisions are refined based on experimental data. The ability to interpret not only raw measurements from the robot, but also the decisions taken by an algorithm are of great utility to the developer, as it allows to have a fine-grained understanding of potential failure modes and therefore allows to efficiently make use of experimental data. Data-driven approaches often make use of an end-to-end paradigm, which means they directly map from sensory observations to a form of control commands. This approach allows to learn complex mappings from sensory inputs to commands, but comes at the cost of reduced interpretability.

I have approached this problem of interpretability in the context of autonomous drone racing by extending the predictions of the neural network with an uncertainty component, allowing for efficient integration into an optimal estimator (Paper B).



**Figure 1.7** – The impressive agility of quadrotors has been demonstrated in a large body of prior work that has heavily relied on external tracking systems or pre-built maps of the environment. Foehn et al. [91] (A) demonstrated superhuman performance in autonomous drone racing by computing a time-optimal quadrotor trajectory and tracking it using MPC and an external motion capture system. Mellinger et al. [228] (C) used polynomial trajectories to perform precise maneuvers through a set of given waypoints. Ryou et al. [296] (D) utilized data-driven approaches to design very fast trajectories, accounting for these factors by optimizing for them during a set of carefully selected experiments. Lupashin et al. [214] (B) leveraged an iterative learning strategy to perform multiple flips with a quadrotor.

## 1.2 Related Work

This section summarizes prior work in the context of autonomous navigation of aerial robots. To this end, related work is grouped according to the amount of information that is required about the environment, i.e. if approaches assume the environment to be known, make use of an external tracking system, or if no such assumptions are made. Even though dynamics modeling is an integral part of any robotic system and definitely an instrumental component of many approaches in the first two sections, I dedicate a separate section to the topic of data-driven modeling to highlight its potential impact. To improve reproducibility and lower the bar of entry for new students, a robotics researcher needs to have access to development and testing environments, both in simulation and in the real world. There is still substantial work required to reach a sufficient level of reproducibility in robotics research, which I illustrate in individual sections on existing simulators and hardware platforms. Finally, as this thesis utilizes the recurring demonstrator of autonomous drone racing, I summarize recent progress on that task in its own section.

### 1.2.1 Autonomous Navigation: External Sensing

Assuming reliable access to accurate state estimation, an entire line of work has investigated high-speed navigation of aerial vehicles [214, 2, 243, 240, 228, 346, 321, 234], even reaching superhuman performance in drone racing [91]. These works are summarized here and put into context to demonstrate the level of performance achievable assuming perception challenges are solved.

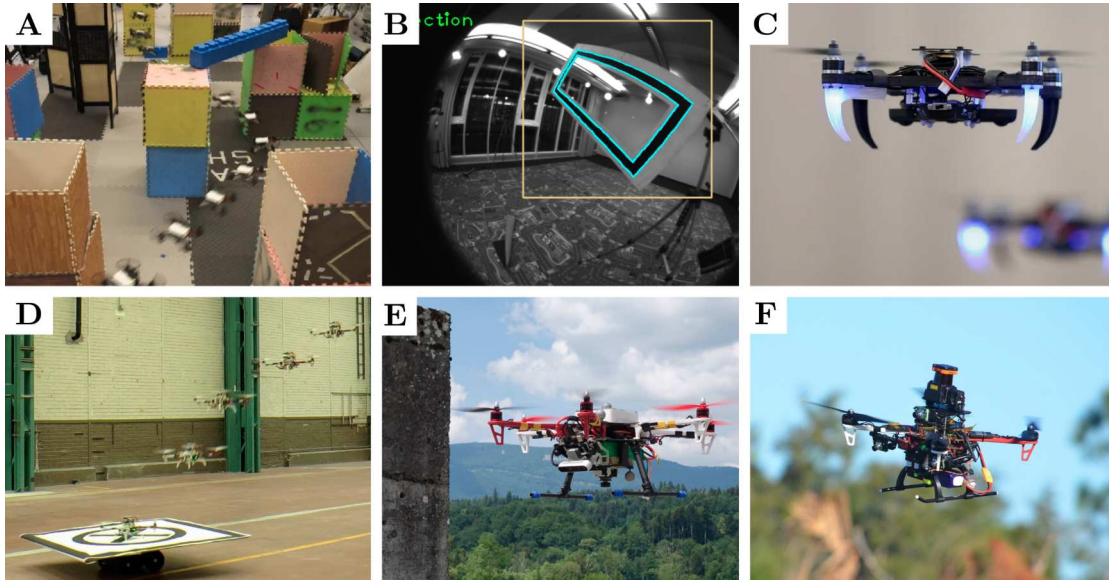
**Traditional Methods.** The impressive agility of quadrotors has been demonstrated in a large body of prior work that has either relied on external tracking systems [214, 90, 91, 228] or pre-built maps of the environment [37]. Lupashin et al. [214] proposed iterative learning of control strategies to enable platforms to perform multiple flips. Mellinger et al. [228] used a similar strategy to autonomously fly quadrotors through a tilted window [228]. By switching between two controller settings, Chen et al. [48] also demonstrated multi-flip maneuvers. Abbeel et al. [2] learned to perform a series of acrobatic maneuvers with autonomous helicopters. Their algorithm leverages expert pilot demonstrations to learn task-specific controllers. In [91], Foehn et al. proposed a trajectory generation method that allows to compute time-optimal quadrotor trajectories, which can then be tracked by a model-predictive controller. Their approach was the first to achieve superhuman performance in autonomous drone racing, but did so by relying on an external tracking system that provided highly accurate state estimation at a high rate.

**Learning-Based Methods.** One of the earliest demonstrators of learning-based approaches for real-world quadrotor control has been proposed by Hwangbo et al. [145]. They used model-free reinforcement learning to train a policy capable of maintaining stable hover flight, even when initialized in difficult-to-recover states such as a throw by a human operator. Molchanov et al. [234] showed training of a stabilizing quadrotor control policy from scratch in simulation and deployment on multiple real platforms. In [322], Song et al. trained a policy to perform autonomous drone racing. The policy directly maps from a measurement of the platform state and the relative position of the next gate to be passed to a control command in the form of collective thrust and bodyrates. The policy is shown to be robust to changes in the track layout.

**Summary.** While these works, both traditional and learning-based, showed impressive examples of agile flight, they focused purely on planning and control. The issues of unreliable perception and state estimation during agile maneuvers instead 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 or by relying on GPS or active sensors such as LIDAR in combination with a pre-built map.

### 1.2.2 Autonomous Navigation: Onboard Sensing and Computation

Autonomous navigation of a drone in an unknown environment raises fundamental robotics challenges in estimation, planning, and control. Due to their limited payload capacity, autonomous drones often rely on vision-based state estimation in the form of visual-inertial odometry [96, 32, 278, 75, 241]. While providing accurate estimation at

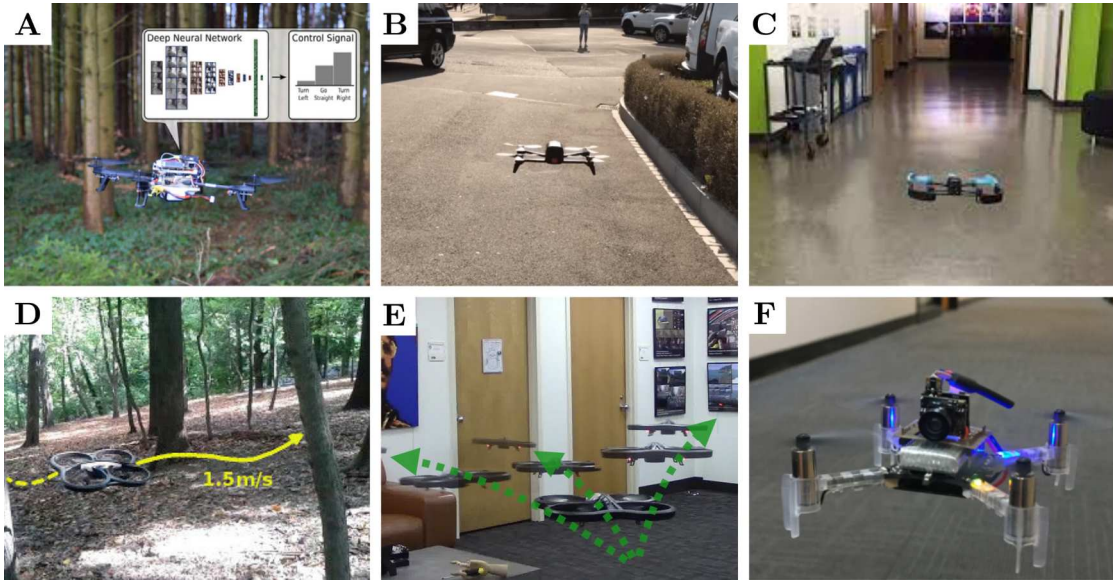


**Figure 1.8** – Traditional approaches demonstrating fully autonomous navigation using only onboard sensing and computation. **A**: Zhou et al. propose RAPTOR [374], a method for perception-aware trajectory replanning. **B,C**: Falanga et al. [83] and Loianno et al. [203] both demonstrate vision-based quadrotor flight in previously known environments. **D,E,F**: Approaches that make use of substantially more onboard computation for the task of landing on a moving platform [84] and autonomous flight in unknown environments [256, 233].

a very low weight when deployed in ideal conditions, cameras suffer from fundamental limitations such as limited dynamic range, motion blur, and restricted field of view. In contrast to traditional cameras, utilizing LIDAR [37] or novel sensors such as event cameras [104] on autonomous drones [288] has the potential to overcome the limitations of frame-based cameras. However, they are often too heavy or too expensive to be deployed on consumer-grade products [64]. In the following, I present an overview of traditional and learning-based methods that perform autonomous navigation while only relying on onboard sensing and computation.

**Traditional Methods.** To tackle the challenges of imperfect perception when operating with onboard sensory measurements, traditional methods have typically resorted to a separation of the autonomy stack into individual compute modules consisting of: perception, mapping, planning, and control. This separation paradigm of the autonomy stack has been deployed on a large range of multirotor platforms, ranging from small-scale quadrotors [203], over medium-sized quadrotors [373], up to large multirotor platforms [256, 233]. Some works tackle only perception and build high-quality maps from imperfect measurements [128, 103, 303, 76, 33], while others focus on planning without considering perception errors [39, 282, 6, 202]. Numerous systems that combine online mapping with traditional planning algorithms have been proposed to achieve autonomous flight in previously unknown environments [257, 261, 233, 20, 372, 295, 345, 50, 371, 149, 122, 114, 191]. Although all these approaches demonstrate successful navigation in previously unknown environments, their maximum achieved speed when deployed in challenging environments is below  $4 \text{ m s}^{-1}$ . Approaches that achieve higher speeds make use of active





**Figure 1.9** – Recent approaches investigate the usage of learning-based methodologies for autonomous vision-based navigation. While such an approach has the potential to substantially reduce latency and increase robustness against imperfect sensing, it requires to collect a large and diverse dataset to train. While [112, 206, 290, 106] (A,B,D,E) collect this dataset in the real world, Sadeghi et al. [298] (C) propose to train the navigation policy *entirely* in simulation and transfer it to the real system. Kang et al. [163] (F) instead investigate how data from both simulation and the real world can be combined. None of these systems achieved agile flight, with the top speeds being reached staying below  $2\text{ m s}^{-1}$ .

sensors such as LIDAR [37] or are deployed in environments with very low obstacle density [20]. In contrast, approaches that only rely on passive sensors and are deployed in challenging environments only achieve speeds below  $4\text{ m s}^{-1}$ . This limitation can be attributed mainly to two reasons, also illustrated in Figure 1.12: (i) The available onboard compute resources do not allow to run the modularized autonomy stack at sufficient frequency to enable high-speed flight [203]. (ii) Even though the platform carries sufficient compute resources, the resulting robot is too heavy and therefore constraining available operating modes to only near-hover conditions [233, 256, 114].

Both these reasons are caused by the division of the navigation task into separate compute modules: perception, mapping, planning, and control. Such strategy leads to pipelines that largely neglect interactions between the different stages and thus compound errors [371]. The sequential nature of these pipelines ignores the tight coupling between perception and action and introduces additional latency, making high-speed and agile maneuvers difficult to impossible [80]. 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 [204].

**Learning-Based Methods.** Deviating from the traditional separation approach of the autonomy stack, recent works propose to use learning-based approaches for the task of

vision-based navigation [49]. These approaches train policies directly from data without explicit mapping and planning stages [290, 112, 298, 106, 206, 163], resulting in potentially lower latency compared to their traditional counterparts. While the policies proposed in [290, 206] are trained by imitating a human, [298, 24] train their policy purely on simulated data, [112, 106] train directly on real-world data, and [163] propose to combine simulated data with few real-world samples for training. 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 [206, 106, 290, 163] and/or discrete actions [298], at the cost of reduced maneuverability and agility.

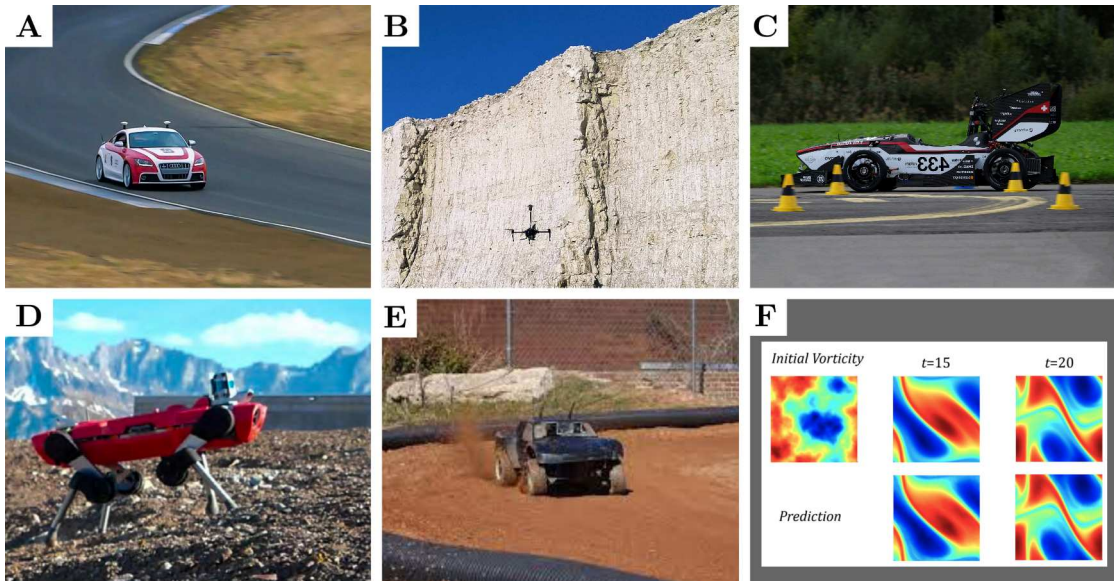
**Summary.** When moving away from the assumption of perfect perception and state estimation, the performance of autonomous aerial systems substantially drops. While this drop in performance can be observed for both traditional and learning-based approaches, they arise due to different reasons. In the case of traditional approaches, the main cause for reduced performance is the large computational overhead introduced by the mapping and planning system. This computational overhead either directly results in a reduced maximum speed due to limited onboard computation, or it is counteracted with more onboard compute resources, which in turn gives rise to large and heavy platforms that are no longer capable of agile flight. For learning-based approaches, the main cause for reduced performance is the lack of training data. Recent work has circumvented this problem by reducing the task complexity, such as restricting the drone’s motion to planar, near-hover motion. Although such restriction allowed to deploy data-driven approaches on real-world systems, it did not exploit the agility of the platform.

There exists a clear need to close the performance gap between approaches leveraging external sensing and those using only onboard sensing and computation. This thesis contributes methodologies towards this goal and demonstrates the first approach to perform vision-based acrobatic quadrotor flight (Paper E), advances the state of the art for vision-based flight in the wild (Paper F), and demonstrates the first approach to outperform a professional human pilot in a vision-based drone race (Paper H).

### 1.2.3 Data-Driven Dynamics Models

Modeling and system identification is an instrumental part of control. Traditionally, dynamics models have been identified by using first-principles-based approaches. With the advent of deep learning, a new interest into learning-based dynamics models has emerged. By exploiting the approximation capabilities of data-driven approaches, highly complex dynamics models can be identified purely from data.

Thanks to their ability to identify patterns in large amounts of data, deep neural networks represent a promising approach to model complex dynamics. Recent works (Figure 1.10) that leverage the representational power of deep networks for such modeling tasks include small-scale ground vehicles [359] (E), aerodynamics modeling of autonomous multirotors and helicopters [343, 277], turbulence prediction [199] (F), actuator modeling [143] (D), and tire friction modeling of full-size race cars [326, 325] (A).



**Figure 1.10** – The usage of data-driven dynamics models has enabled numerous breakthroughs in a variety of robotics domains. Spielberg et al. [326, 325] (A), Hewing et al. [130, 157] (C), and Williams et al. [359] (E) used learning-based dynamics models for high-speed autonomous driving. Mehndiratta et al. [224] (B) employed Gaussian processes for estimating wind disturbances. Hwangbo et al. [143] (D) leveraged a neural-network-based model of the actuators to transfer a control policy for legged locomotion from simulation to the real world. Li et al. [199] (F) predict turbulence using a neural network model.

While deep neural networks are highly versatile function approximators, they require large and diverse datasets for training, which renders them suboptimal in settings where only few data is available. Gaussian processes in contrast represent a nonparametric approach that is especially well suited for situations where only little data is available and that directly provides a measure of model uncertainty. As a result, there exists already a substantial body of work that utilizes Gaussian processes for system modeling, including race cars [130, 157] (C) and quadrotors [224, 42, 65] (B).

**Summary.** Using a dynamics model in a robotic context includes two fundamental use cases: (i) using the model for simulation, and (ii) using the model for control. The salient difference between these two use cases is their real-time requirement; while simulation can potentially run at a fraction of real-time, a control loop running on a mobile robot has to maintain strict real-time constraints. I have investigated both use cases in the context of high-speed quadrotor flight. In Paper I, I propose the usage of Gaussian processes to model residual aerodynamic effects and directly use such residual model in a predictive controller. In Paper J, I increase the residual model complexity by replacing the Gaussian processes with a deep neural network. This model achieves substantially higher accuracy, at the cost of real-time capability. Both approaches make use of a *residual* model, i.e. the data-driven model only complements an existing model instead of learning the full dynamics from data. Such an approach has shown to achieve better generalization to unseen maneuvers.

### 1.2.4 Simulation

Simulators are invaluable tools for the robotics researcher. They allow developing and testing algorithms in a safe and inexpensive manner, without having to worry about the time-consuming and expensive process of dealing with real-world hardware. The ideal simulator is: (i) *fast*, to collect a large amount of data with limited time and compute; (ii) *physically-accurate*, to represent the dynamics of the real world with high-fidelity; and (iii) *photo-realistic*, to minimize the discrepancy between simulated and real-world sensors' observations. Realizing the potential of simulators, there has been large interest in the robotics community to push for better simulation environments that would accelerate research by not only providing researchers more powerful tools for experimentation, but also creating new widely accepted benchmarks, where robotic algorithms can be tested and evaluated in a controlled and repeatable setting.

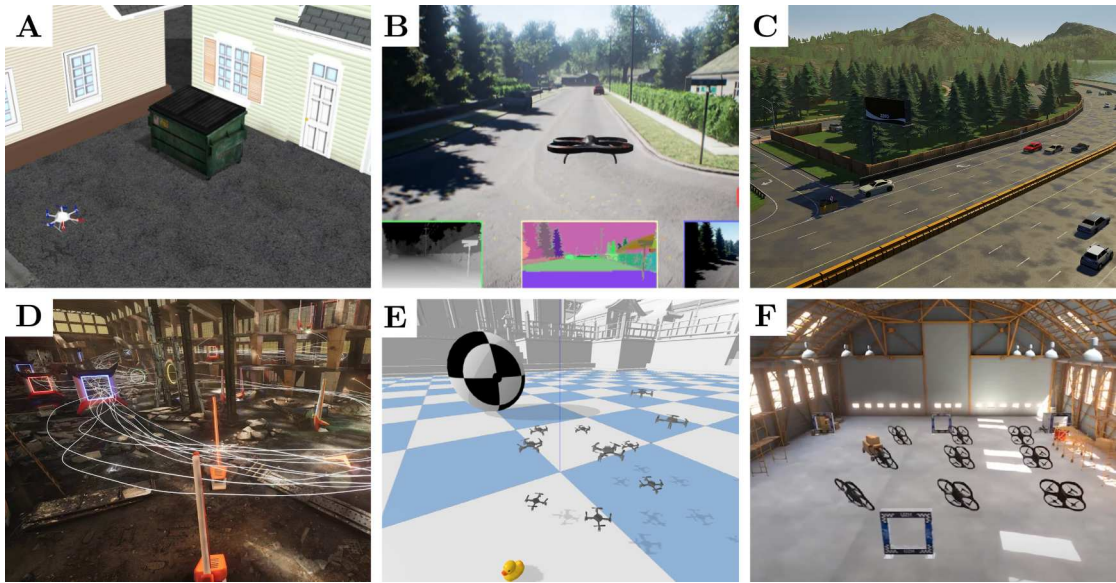
For quadrotor simulation, there exists a variety of available alternatives [102, 265, 320, 309, 342, 117, 183], illustrated in Figure 1.11, each focusing on a different use case. While [309, 117] provide near-photorealistic rendering by interfacing with state-of-the-art game engines such as Unity or Unreal Engine, [342] focuses on efficient physics simulation, allowing fast training of control policies using deep reinforcement learning. Both RotorS [102] (A) and Hector [183] are popular Micro Aerial Vehicle (MAV) simulators built on Gazebo [182], which is a general robotic simulation platform and often used with the popular Robot Operating System (ROS). Hector is a collection of open-source modules and is primarily used for autonomous mapping and navigation with rescue robots. RotorS provides several multi-rotor helicopter models such as the AscTec Hummingbird, Pelican, and Firefly. 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 difficult to develop learning-based systems.

FlightGoggles [117] (D) is a photo-realistic sensor simulator for perception-driven robotic vehicles. FlightGoggles consists of two separate components: a photo-realistic rendering engine built on Unity and a quadrotor dynamics simulation implemented in C++. In addition, it also provides an interface with real-world vehicles and actors in a motion capture system. FlightGoggles is very useful for rendering camera images given trajectories and inertial measurements from flying vehicles in real-world, in which the collected dataset [13] is used for testing vision-based algorithms.

Both AirSim <sup>1</sup> [309] (B) and CARLA [70] (C) are open-source photo-realistic simulators for autonomous vehicles built on Unreal Engine. CARLA is mainly made for autonomous driving research and only provides the dynamics of ground vehicles. Conversely, AirSim offers an interface to configure multiple vehicle models for quadrotors and supports hardware-in-the-loop (HITL) as well as software-in-the-loop (SITL) with flight controllers such as PX4. The vehicle is defined as a rigid body whose dynamics model is simulated using NVIDIA's physics engine PhysX, a popular physics engine used by the large majority of today's video games. However, this physics engine is not specialized for quadrotors

---

<sup>1</sup>AirSim also has an experimental Unity release.

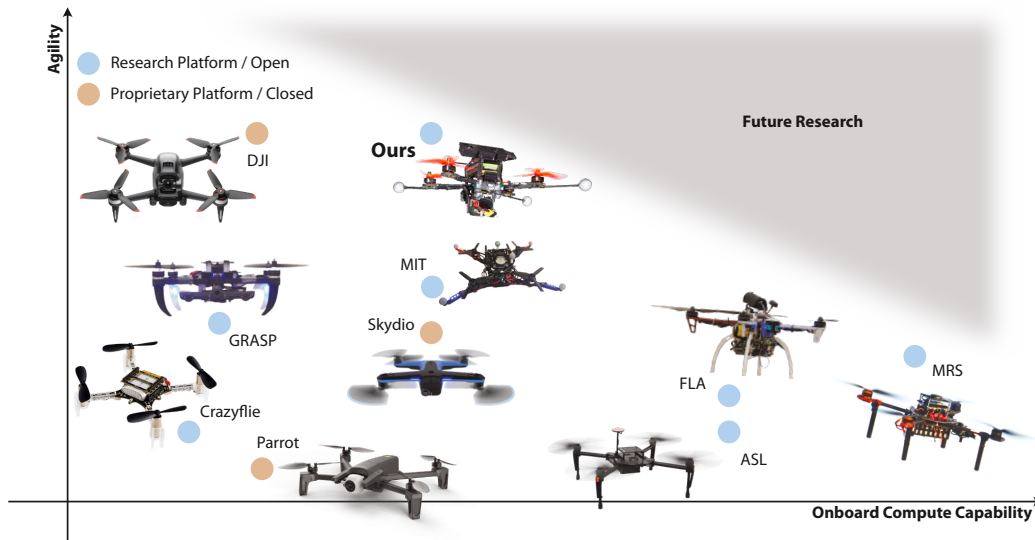


**Figure 1.11** – There already exists a wide range of simulators suitable to develop and test autonomous navigation algorithms. While some of them come with relatively accurate dynamics modeling [102] (A), others are tailored for high photorealism [309, 70] (B, C), or efficient training of reinforcement learning agents [320] (F). FlightGoggles [117] (D) is a photo-realistic sensor simulator for perception-driven robotic vehicles. An interesting alternative is proposed by [265] (E), a versatile simulator that unifies efficient physics simulation using the Bullet physics engine with realistic rendering, and standardized interfaces for reinforcement learning. I have contributed to build Flightmare [320] (F), a simulator which offers accurate dynamics modeling, efficient training routines for machine learning tasks, and offers photorealistic rendering by interfacing to Unity, a state-of-the-art game engine.

(or robots), and it is tightly coupled with the rendering engine to allow simulating environment dynamics. Because of this rigid connection between rendering and physics simulation, AirSim can achieve only limited simulation speeds. This limitation makes it difficult to apply the simulator to challenging model-free reinforcement learning tasks.

An interesting alternative is proposed in [265] (E), which proposes a versatile simulator that unifies efficient physics simulation using the Bullet physics engine with realistic rendering, and standardized interfaces for reinforcement learning. Apart from the aforementioned simulators, there are many more existing simulators that have been widely adopted by other research communities. For example, MuJoCo [342] has been widely used by the reinforcement learning community for benchmark comparisons. Similarly, RaiSim [144] is a physics engine for robotics and AI research written in C++, that supports massive parallel dynamics simulation. However, both simulators do not support complex 3D environments and photo-realistic image rendering. Sim4CV [244] is a photo-realistic simulator but made solely for computer vision applications.

**Summary.** One of the main limitations of current quadrotor simulators is their rigid nature. The trade-off between accuracy and speed has always been in the hands of the simulator developers, not of the end-users. However, this paradigm leaves some questions



**Figure 1.12** – A qualitative comparison of different available consumer and research platforms with respect to available onboard compute capability and agility. The open-source frameworks FLA [233], ASL [297], and MRS [14] have relatively large weight and low agility. The DJI [69], Skydio [317], and Parrot [300] are closed-source commercial products that are not intended for research purposes. The Crazyflie [109] does not allow for sufficient onboard compute or sensing, while the MIT [13] and GRASP [203] platforms are not available open-source. Finally, the *Agilicious* (“Ours”) platform to which I contributed during my Ph.D., provides agile flight performance, onboard GPU-accelerated compute capabilities, as well as open-source and open-hardware availability.

open: What if we want to *dynamically* change the underlying physics model? What if we want to *actively* trade-off photo-realism for speed? During my Ph.D., I have contributed to a novel quadrotor simulator, called Flightmare [320] (F), that is specifically designed to put the speed vs. accuracy trade-off in the hands of the users. Flightmare is composed of two main blocks: a rendering engine, based on Unity [153], and a physics model. These blocks are completely decoupled and can run independently from each other. Besides, each block is flexible by design. Indeed, the rendering block can be used within a wide range of 3D realistic environments and generate visual information from low to high photo-realism. With minimal additional computational costs, it is also possible to simulate sensor noise, *e.g.* motion-blur, environment dynamics, *e.g.* wind, and lens distortions [153]. Similarly, the physics block offers full control to the user in terms of the desired robot dynamics and associated sensing. Depending on the application, the users can easily switch between a basic (noise-free) quadrotor model and a more advanced rigid-body dynamics, including friction and rotor drag, or directly use the real platform dynamics like [117]. Inertial sensing and motor encoders, which directly depend on the physics model, can also be noise-free or include different degrees of noise [102, 183].

### 1.2.5 Hardware Platforms

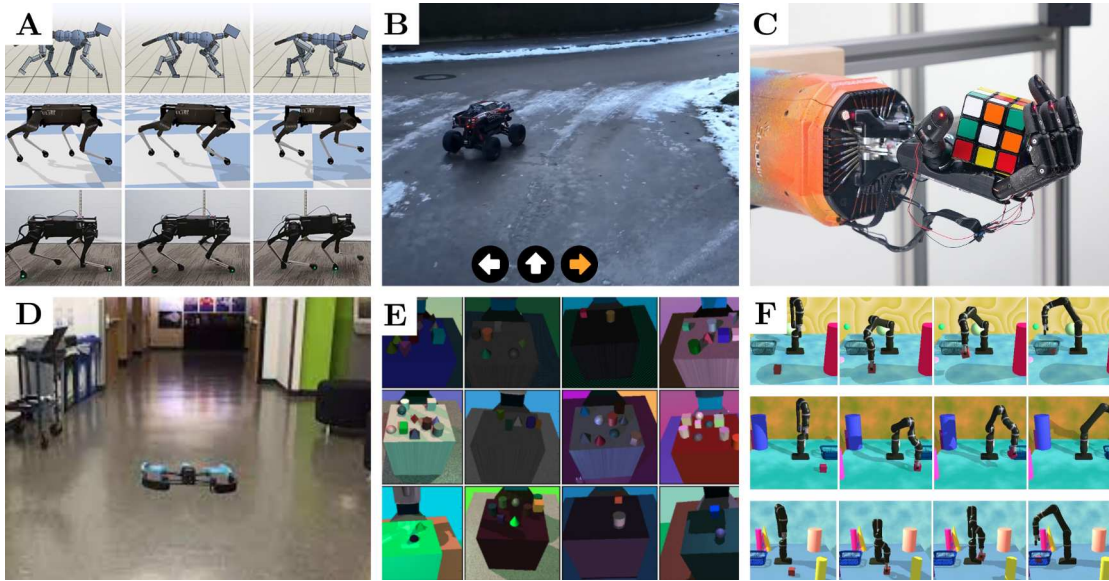
Agile flight comes with ever-increasing engineering challenges since performing faster maneuvers with an autonomous system requires more capable algorithms, specialized hardware, and proficiency in system integration. As a result, only a small number of research groups have undertaken the significant overhead of hardware and software engineering, and have developed the expertise and resources to design quadrotor platforms that fulfill the requirements on weight, sensing, and computational budget necessary for autonomous agile flight. The platforms and software stacks developed by research groups [226, 203, 233, 297, 78, 256, 14, 126], illustrated in Figure 1.12, vary strongly in their choice of hardware and software tools. This is expected, as optimizing a robot with respect to different tasks based on individual experience in a closed-source research environment leads to a fragmentation of the research community. For example, even though many research groups use the Robot Operating System middleware to accelerate development, publications are often difficult to reproduce or verify since they build on a plethora of previous implementations of the authoring research group. In the worst case, building on an imperfect or even faulty closed-source foundation can lead to wrong or non-reproducible conclusions, slowing down research progress.

To break this vicious cycle and to democratize research on fast autonomous flight, the robotics community needs an open-source and open-hardware quadrotor platform that provides the versatility and performance needed for a wide range of agile flight tasks. Such an open and agile platform does not yet exist, which is why I contributed throughout my Ph.D. to a novel quadrotor platform, called *Agilicious*.

### 1.2.6 Simulation-to-Reality Transfer

The hardship to generate data in the real world led the research community to embrace the idea of simulation-to-reality transfer: training a data-driven sensorimotor policy entirely in simulation and transferring it to the real robot. Compared to data collection in the real world, simulation is especially appealing due to its ability to generate data faster, cheaper, safer, and more informative than real-world experimentation. Prior work on simulation-to-reality transfer has studied challenging real-world robotic problems related to grasping [341, 118, 362, 34, 148, 294, 299, 281], in-hand manipulation [5], and navigation [298, 247, 267].

Methodologies to facilitate simulation-to-reality transfer include domain randomization of visual features [341, 148, 298], abstraction of the policy inputs [247], or generative models to translate simulated images into realistic ones [281]. OpenAI et al. [5] achieved complex real-world in-hand manipulation by training a policy purely in simulation using domain randomization on physics properties such as gravity vector, inertia values, and friction coefficients. Peng et al. [267] investigate adaptation using real-world reference data for efficient simulation to reality transfer of locomotion policies. Similarly, Smith et al. [319] investigate fine-tuning of locomotion policies using data obtained from the real robot. Finally, Kumar et al. [186] condition the policy on a latent encoding that characterizes dynamics properties of the robot that is trained in simulation.



**Figure 1.13** – Prior work on simulation-to-reality transfer has studied challenging real-world problems related to legged locomotion (A), navigation (B, D), grasping (E, F), in-hand manipulation (C).

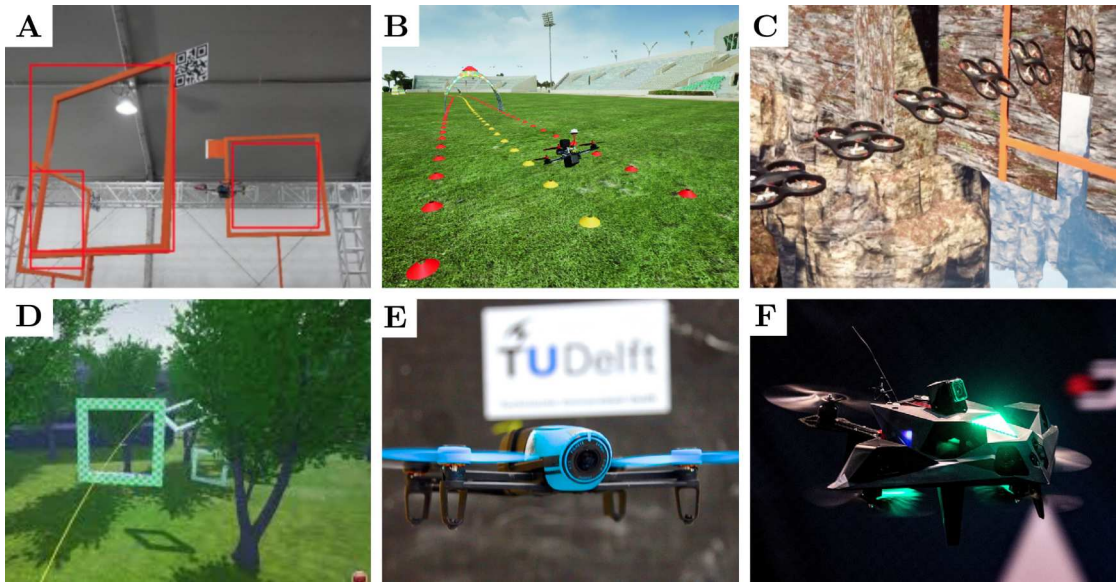
Although all these works achieve successful simulation-to-reality transfer, none of them considered the challenge of high-speed navigation. In my thesis, I extend the methodologies of domain randomization, abstraction, and real-world adaptation to the challenge of agile drone flight and validate them on the task of drone racing (Paper D, Paper H), acrobatic flight (Paper E), and high-speed flight in the wild (Paper F).

### 1.2.7 Autonomous Drone Racing

Autonomous drone racing has gained a lot of attention from the research community [235, 217] in recent years and is a recurrent demonstrator in this thesis. This focus is due to the fact that autonomous racing presents unique opportunities and challenges in designing algorithms and hardware that can operate at the limits of perception, planning, and control.

Early works investigating the challenge of autonomous drone racing include Jung et al. [155, 156] (Figure 1.14A), who consider the problem of autonomous drone navigation in a previously unseen track. They use line-of-sight guidance combined with a deep-learning-based gate detector. As a consequence, the next gate to be traversed has to be in view at all times. Müller et al. [246] (Figure 1.14B) instead proposed to directly train an end-to-end policy that maps from image observations to control commands. While this approach has been successfully deployed in simulation, it was never tested on a real drone platform. Organized during the 2019 NeurIPS conference, the *Game of Drones* competition [217] (Figure 1.14D) considered the task of autonomous drone racing in the photorealistic simulator AirSim [309]. This competition resulted in increased interest in drone racing by the research community and new approaches and





**Figure 1.14** – Autonomous drone racing has received a lot of attention from the research community. Notable works investigating autonomous drone racing in the real world include Jung et al. [156] (A), Li et al. [197] (E), and De Wagter et al. [62] (F), who proposed the winning approach to the 2019 AlphaPilot challenge. Prior work also investigates the task of drone racing in the simulation domain [246, 121] (B, C), including the simulation-only Game of Drones competition [217] (D).

baselines being proposed, such as the optimization-based trajectory generation approach of Han et al. [121] (Figure 1.14C).

The largest competition about autonomous drone racing held to date was the 2019 AlphaPilot competition organized by Lockheed Martin and the Drone Racing League. Over 400 international teams participated in the competition, with the best eight teams proceeding to the finals, where races with real drone platforms were held. The winning approach of this competition, proposed by De Wagter et al. [62] (Figure 1.14F) demonstrated speeds up to  $10 \text{ m s}^{-1}$ , but was still clearly outperformed by a human pilot who managed to fly the same race track in just half the time.

In this thesis, I present approaches and methodologies to push the level of performance of autonomous racing drones to human-level performance, as demonstrated in Paper H. This demonstrator builds on methodologies developed and insights obtained in my prior works, including the identification of a suitable control modality (Paper G), accurate modeling of the platform physics (Paper J), and robust perception of racing gates (Paper C).



## 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.

In total, this research has been published in five journal publications (two in *Science Robotics*, one in *Springer: Autonomous Robots*, two in *IEEE Robotics and Automation Letters*) and six peer-reviewed conference publications. A complete list of all publications can be found on page [xi](#).

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

### Awards:

- *Robotics: Science and Systems (RSS) 2020*, **Best System Paper Award** for Paper [C](#) and invitation to publish in *Springer: Autonomous Robots*.
- *IEEE Transactions on Robotics, 2020*, **King-Sun Fu Memorial Best Paper Award (Honorable Mention)** for Paper [D](#)
- *Robotics: Science and Systems (RSS) 2020*, **Best Paper Award (Honorable Mention)** for Paper [E](#)
- *AlphaPilot Challenge, 2019*, organized by *Lockheed Martin* and the *Drone Racing League*, **2nd Place out of 430 participants worldwide** with the approach presented in Paper [C](#).
- *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2018* in the *Autonomous Drone Racing (ADR)* challenge, **Winner 1st Place** with the approach presented in Paper [B](#).
- *Conference on Robot Learning (CoRL) 2018*, **Best System Paper Award**, for Paper [A](#) and invitation to publish in *IEEE Transactions on Robotics*.
- *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2017*, **Best Search and Rescue Robotics Paper Award (Honorable Mention)**, for Paper [L](#).

### Software:

- **Agilicious Flightstack:** Paper [K](#)  
available at <https://agilicious.dev>
- **Learning High-Speed Flight in the Wild:** Paper [F](#)  
available at [https://github.com/uzh-rpg/agile\\_autonomy](https://github.com/uzh-rpg/agile_autonomy)
- **Deep Drone Acrobatics:** Paper [E](#)  
available at [https://github.com/uzh-rpg/deep\\_drone\\_acrobatics](https://github.com/uzh-rpg/deep_drone_acrobatics)
- **Deep Drone Racing:** Paper [D](#)  
available at [https://github.com/uzh-rpg/sim2real\\_drone\\_racing](https://github.com/uzh-rpg/sim2real_drone_racing)
- **Data-Driven MPC for Quadrotors:** Paper [I](#)  
available at [https://github.com/uzh-rpg/data\\_driven\\_mpc](https://github.com/uzh-rpg/data_driven_mpc)

### Media Coverage:

- **The New York Times** on Paper [C](#): *A.I. Is Flying Drones (Very, Very Slowly)*  
Article available at this [Link](#).
- **Forbes** on Paper [F](#): *This Hotshot AI Drone Can Speed Through Complex Environments Thanks To New Kind Of Virtual Training*  
Article available at this [Link](#).
- **WIRED** coverage about Paper [A](#): *Drones Just Learned to Fly Solo*  
Article available at this [Link](#).
- **Der Spiegel** on Paper [E](#): *Akrobatische Drohnen*  
Article available at this [Link](#).
- **IEEE Spectrum** on Paper [E](#): *AI-Powered Drone Learns Extreme Acrobatics*  
Article available at this [Link](#).
- **Tech Xplore** on Paper [F](#): *Flying high-speed drones into the unknown with AI*  
Article available at this [Link](#).
- **Focus Online** on Paper [F](#): *Geschulte Drohnen sausen ins Unbekannte*  
Article available at this [Link](#).
- **Metro UK** on Paper [F](#): *Artificial intelligence means drones can dodge through forests at 25mph*  
Article available at this [Link](#).
- **Robohub** on Paper [E](#): *Drones learn acrobatics by themselves*  
Article available at this [Link](#).

## 2.1 Tight Coupling of Learning-Based Perception and Optimal Planning and Control

In this part of the thesis, I consider the challenge of effectively interfacing learning-based perception with traditional planning and control stacks. While fully learned sensorimotor policies have potentially superior capabilities in terms of compute requirements and latency compared to such hybrid approaches, for many industry applications fully learned systems are not well suited due to their limited interpretability and verifiability.

Combining data-driven perception with traditional planning and control requires to find suitable *intermediate representations* that are appropriate for learning and integrate well with state-of-the-art planning and control frameworks. Prior work in the context of autonomous navigation proposed intermediate representations in the form of high-level velocity commands [206, 298]. Such intermediate representation can be readily integrated into an existing planning and control system, but excessively constrains the maneuverability of the platform.

The demonstrator considered in this part of the thesis is autonomous drone racing. In autonomous drone racing, we assume having access to an approximate estimate of the race track layout. The drone is then tasked with navigating through the track as fast as possible, while only relying on onboard sensory observations such as images and measurements from accelerometers and gyroscopes. While there already exist approaches that combine a learned perception system with a classical control stack [156, 206, 298, 106], none of these do either fully exploit the dynamical capabilities of quadrotors or use the network predictions also for state estimation.

This thesis contributes to addressing the challenges of effective combination of data-driven perception with optimal planning and control via intermediate representations. In Paper A, I propose to use receding-horizon waypoints in combination with efficient state-to-state trajectory generation. Paper B extends this methodology to arbitrary race track layouts by replacing the receding-horizon waypoint formulation with a probabilistic relative pose prediction of the next gate. Finally, Paper C addresses cases where multiple gates are visible by directly performing prediction in the image plane.



**Figure 2.1** – By combining a convolutional neural network with state-of-the-art trajectory generation and control methods, our vision-based, autonomous quadrotor is able to successfully navigate a race track with moving gates with high agility.

### 2.1.1 Paper A: Deep Drone Racing: Learning Agile Flight in Dynamic Environments

Elia Kaufmann\*, Antonio Loquercio\*, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. “Deep Drone Racing: Learning Agile Flight in Dynamic Environments”. In: *Conf. on Robotics Learning (CoRL)*. 2018, **Best System Paper Award**

Autonomous agile flight brings up fundamental challenges in robotics, such as coping with unreliable state estimation, reacting optimally to dynamically changing environments, and coupling perception and action in real time under severe resource constraints. I have investigated these challenges in the context of autonomous, vision-based drone racing in dynamic environments. The approach presented in this work combines a convolutional neural network (CNN) with a state-of-the-art path-planning and control system. The CNN directly maps raw images into a robust representation in the form of a waypoint and desired speed. This information is then used by an efficient trajectory generation method to produce state-to-state minimum-jerk trajectories that are tracked by a cascaded control pipeline. The method is demonstrated in autonomous agile flight scenarios, in which a vision-based quadrotor traverses drone-racing tracks with possibly moving gates. The method does not require any explicit map of the environment and runs fully onboard.

My contribution to this work includes the conceptualization, development, and implementation of the data generation procedure, the choice of intermediate representation in the form of state-to-state trajectories, the design of the hardware platform, the integration of the state estimation, and the evaluation of the approach in both simulation and in real-world experiments.

**Supplementary Video:** <https://youtu.be/8RILnqPxo1s>

## 2.1. Tight Coupling of Learning-Based Perception and Optimal Planning and Control

---



**Figure 2.2** – Our quadrotor flies through an indoor track. Our approach uses optimal filtering to incorporate estimates from a deep perception system. It can race a new track after a single demonstration.

### 2.1.2 Paper B: Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. “Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)* (2019), pp. 690–696. DOI: [10.1109/ICRA.2019.8793631](https://doi.org/10.1109/ICRA.2019.8793631), **This approach won the 2018 IROS Autonomous Drone Race.**

In this paper, I present a system for autonomous, vision-based drone racing combining a learned perception system with model predictive control that can be deployed on novel track layouts after only a single demonstration flight. Our approach represents the global track layout with coarse gate locations, which can be easily estimated from a single demonstration flight. At test time, a convolutional network predicts the poses of the closest gates along with their uncertainty. These predictions are incorporated by an extended Kalman filter to maintain optimal maximum-a-posteriori estimates of gate locations. This allows the framework to cope with misleading high-variance estimates that could stem from poor observability or lack of visible gates. Given the estimated gate poses, we use model predictive control to quickly and accurately navigate through the track. We conduct extensive experiments in the physical world, demonstrating agile and robust flight through complex and diverse previously-unseen race tracks. The presented approach was used to win the IROS 2018 Autonomous Drone Race Competition, outracing the second-placing team by a factor of two.

My contribution to this work includes the design of the network architecture and training loss formulation, implementation of the data generation procedure, development of the simulation environment, design of the hardware setup, and the experimental evaluation in simulation and the real world.

**Supplementary Video:** <https://youtu.be/UuQvijZcUSc>



**Figure 2.3** – Our *AlphaPilot* drone waiting on the start podium to autonomously race through the gates ahead.

### 2.1.3 Paper C: AlphaPilot: Autonomous Drone Racing

Philipp Foehn\*, Dario Brescianini\*, Elia Kaufmann\*, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. “AlphaPilot: Autonomous Drone Racing”. In: *Robotics: Science and Systems (RSS)* (2020), **Best System Paper Award, 2nd Place at 2019 AlphaPilot Competition**

In this paper, I present a novel system for autonomous, vision-based drone racing combining learned data abstraction, non-linear filtering, and time-optimal trajectory planning. The system has successfully been deployed at the first autonomous drone racing world championship: the *2019 AlphaPilot Challenge*. Contrary to traditional drone racing systems, which only detect the next gate, our approach makes use of any visible gate and takes advantage of multiple, simultaneous gate detections to compensate for drift in the state estimate and build a global map of the gates. The global map and drift-compensated state estimate allow the drone to navigate through the race course even when the gates are not immediately visible and further enable to plan a near time-optimal path through the race course in real time based on approximate drone dynamics. The proposed system has been demonstrated to successfully guide the drone through tight race courses reaching speeds up to 8 m/s and ranked second at the *2019 AlphaPilot Challenge*.

My contribution to this work includes the conceptualization, development, and implementation of the perception system to detect racing gates in RGB images. I furthermore contributed to the entire code base, including the adaption of the VIO frontend to account for camera occlusions. I performed tests with the physical platform to identify system parameters such as delay, inertia, drag, and thrust curves.

**Supplementary Video:** <https://youtu.be/DGjwm5PZQT8>



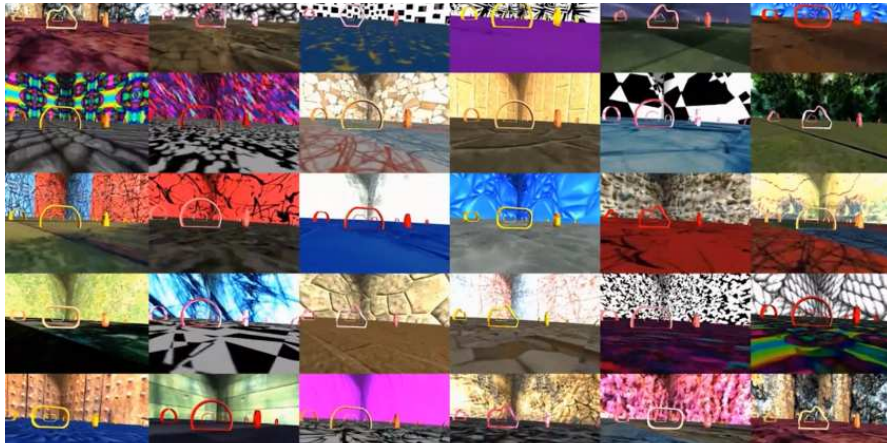
## 2.2 Simulation-to-Reality Transfer for Agile Drone Flight

Even though learning-based sensorimotor policies are a promising approach to push the limits of agile drone flight, training such policies is reliant on a substantial labeled dataset. Obtaining such data in the real world can either be done by instrumenting the environment and using an automated data generation strategy, or by crowd-sourcing human annotators. However, while instrumenting of the environment requires expensive hardware, crowd-sourcing human annotators is often impossible, as the labelling process itself requires expert knowledge when considering quadrotor flight at the limits of handling.

The hardship of obtaining high-quality, potentially labeled, data from real-world systems has led the robotics community to investigate approaches that perform data collection entirely in simulation. While this shift to simulated data allows for virtually unlimited labeled data generation, it comes with its own particular challenges. The most prominent arises when a policy trained in simulation is deployed on a real robot. Due to the distribution shift between simulated data and real-world data, the policy is deployed on out-of-distribution data, potentially with catastrophic consequences.

Prior work investigates approaches to facilitate transfer of sensorimotor policies from simulation to the real world by using randomization [341, 148, 5], abstraction [247], and adaptation [186, 319]. Although these approaches achieve successful transfer in their domains, no prior work investigated simulation-to-reality transfer for high-speed navigation.

The contributions presented in this thesis investigate the challenge of simulation-to-reality transfer in the context of high-speed flight. I investigate how domain randomization and abstraction of visual features enables successful transfer of sensorimotor policies for high-speed drone flight. I apply the methodologies on multiple complex navigation tasks, including autonomous drone racing (Paper D), acrobatic flight (Paper E), and high-speed flight in the wild (Paper F). By extending the transfer studies from the policy observation space to the action space, I identify in Paper G the optimal action space for agile flight that combines robust transferability with high agility. Finally, I combine the insights from these transfer studies with a novel methodology to real-world fine-tuning and demonstrate the first approach to vision-based autonomous drone racing that outperforms a professional human pilot in a drone race (Paper H).



**Figure 2.4** – 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 fine-tuning.

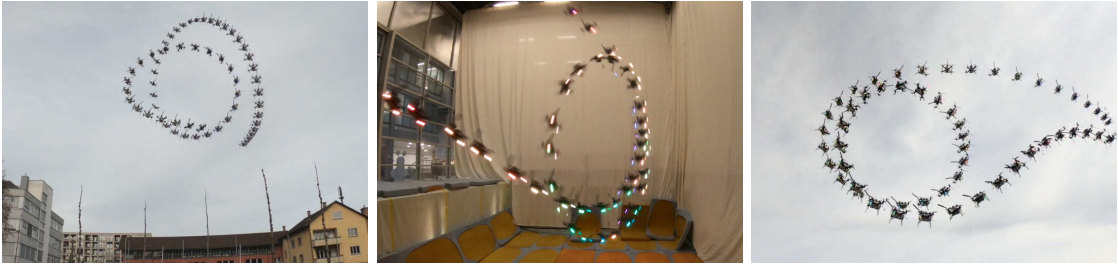
### 2.2.1 Paper D: Deep Drone Racing: From Simulation to Reality with Domain Randomization

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. Robot.* 36.1 (2019), pp. 1–14. DOI: [10.1109/TRO.2019.2942989](https://doi.org/10.1109/TRO.2019.2942989), **Best Paper Award Honorable Mention**

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. The approach presented in this work combines a convolutional neural network (CNN) with a state-of-the-art path-planning and control system. Instead of relying on tedious real-world data collection, this work investigates the feasibility of training the perception system entirely in simulation and transferring it to the real robot. 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 the system robust to changes of illumination and gate appearance.

My contribution to this work includes the conceptualization, development, and implementation of the data generation procedure, the choice of intermediate representation in the form of state-to-state trajectories, the design of the hardware platform, the integration of the state estimation, and the evaluation of the approach in both simulation and in real-world experiments.

**Supplementary Video:** <https://youtu.be/vdxB89lgZhQ>



**Figure 2.5** – 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.

### 2.2.2 Paper E: Deep Drone Acrobatics

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

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 data-driven approaches can not only be used in combination with traditional planning and control, but instead can directly map from sensory observations to low-level commands. Specifically, I propose to train a sensorimotor policy that enables an autonomous quadrotor to fly extreme acrobatic maneuvers with only onboard sensing and computation. The policy is trained entirely in simulation by leveraging demonstrations from an optimal controller that has access to privileged information. I show that successful policy transfer to a real robot can be achieved by using appropriate abstractions of the visual input. 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  $3g$ .

My contribution to this work includes the design of the training objective, implementation of the training procedure, generation of the reference maneuvers, implementation of the system on the real robot, and the evaluation of the approach in both simulation and in real-world experiments.

**Supplementary Video:** [https://youtu.be/2N\\_wKXQ6MXA](https://youtu.be/2N_wKXQ6MXA)



**Figure 2.6** – Deployment of the proposed approach in a challenging real-world scenario featuring snow-covered branches. The sensorimotor policy was trained entirely in simulation and has access to only onboard sensing and computation.

### 2.2.3 Paper F: Learning High-Speed Flight in the Wild

Antonio Loquercio\*, Elia Kaufmann\*, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Learning High-Speed Flight in the Wild”. In: *Science Robotics*. 2021

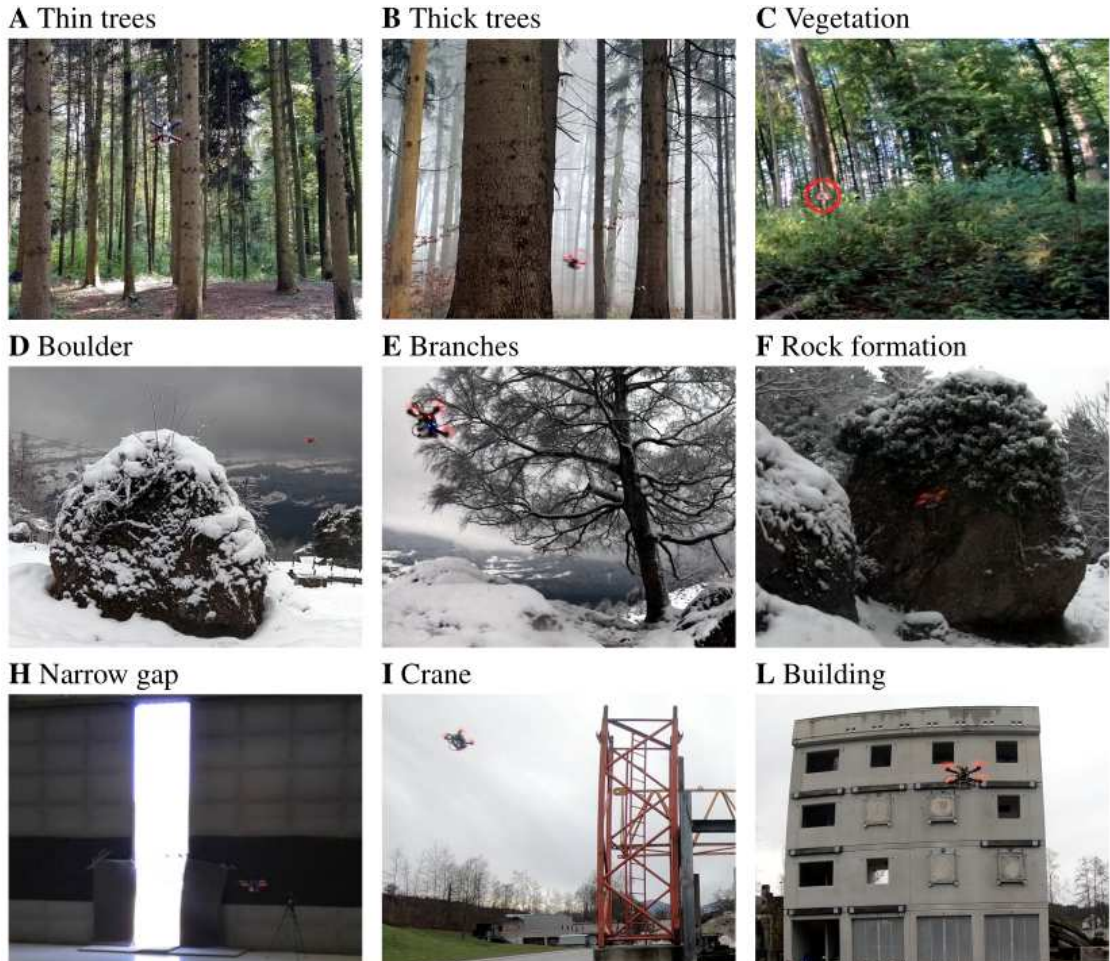
In this work, I extend the methodologies presented in Paper E to one of the grand challenges of robotics [366]: navigation in previously unknown environments with only onboard sensing and computation. To date, only expert human pilots have truly mastered that task and are able to navigate at high speeds through unknown, cluttered environments. In contrast, 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. In this work, I 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. Following the approaches presented in Paper E, this 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, the 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.

My contribution to this work includes the conceptualization and implementation of the

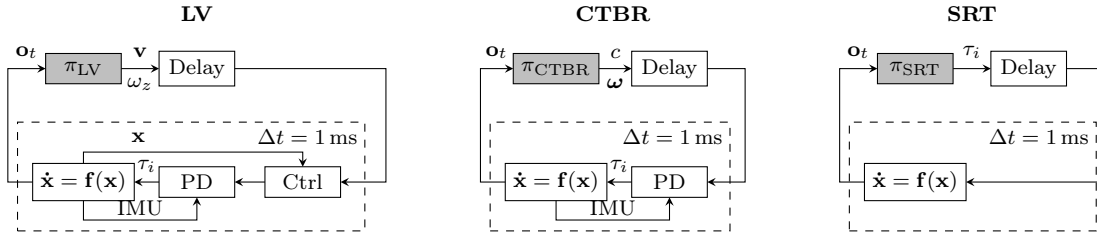
## 2.2. Simulation-to-Reality Transfer for Agile Drone Flight

expert policy in simulation, desing of the simulation and data generation procedure, identification of suitable input modalities, implementation of realistic noise in the depth input, implementation of the reactive planning baseline, integration of the system on the real robot, and the evaluation of the approach in both simulation and in real-world experiments.

**Supplementary Video:** <https://youtu.be/m89bNn6RFoQ>



**Figure 2.7** – Deployment of the proposed approach in a set of challenging environments. All environments are previously unknown. The reader is encouraged to watch the supplementary video to better understand the speed and agility of the proposed approach.



**Figure 2.8** – In this paper, we compare three different classes of control policies for the task of agile quadrotor flight. From left to right: policies specifying desired linear velocities (LV) (they rely on a control stack that maps the output velocities to individual rotor thrusts), policies commanding collective thrust and bodyrates (CTBR) (they rely on a low-level controller that maps the output bodyrates to individual rotor thrusts), policies directly outputting single-rotor thrust (SRT).

### 2.2.4 Paper G: A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. “A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022

Quadrotors are highly nonlinear dynamical systems that require carefully tuned controllers to be pushed to their physical limits. Recently, learning-based control policies have been proposed for quadrotors, as they would potentially allow learning direct mappings from high-dimensional raw sensory observations to actions. Due to sample inefficiency, training such learned controllers on the real platform is impractical or even impossible. Training in simulation is attractive but requires to transfer policies between domains, which demands trained policies to be robust to such domain gap. In this work, I make two contributions: (i) I perform the first benchmark comparison of existing learned control policies for agile quadrotor flight and show that training a control policy that commands body-rates and thrust results in more robust sim-to-real transfer compared to a policy that directly specifies individual rotor thrusts, (ii) I demonstrate for the first time that such a control policy trained via deep reinforcement learning can control a quadrotor in real-world experiments at speeds over 45km/h.

My contribution to this work includes the implementation of the training environment, integration and tuning of the reinforcement learning algorithm for the task of high-speed trajectory tracking, generation of the reference maneuvers, design and implementation of domain randomization of dynamics properties, integration of the system on the real robot, and the evaluation of the approach in both simulation and in real-world experiments.

**Supplementary Video:** <https://youtu.be/zqdfVq2uWUA>



**Figure 2.9** – In this work, I present a novel approach for autonomous drone racing that only relies on onboard sensory observations and onboard computation. The approach combines model-free reinforcement learning with residual models identified from real data to achieve champion-level performance on the task of vision-based drone racing.

### 2.2.5 Paper H: Champion-Level Performance in Drone Racing using Deep Reinforcement Learning

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Champion-Level Drone Racing using Deep Reinforcement Learning”. In: *Nature* (2023)

Vision-based drone racing is one of the most challenging problems for autonomous robots owing to its enormous state space, the strict real-time requirements, and the very thin slack for errors. By leveraging insights from Paper J and Paper G, I propose a novel approach for training neural networks for vision-based drone racing using deep reinforcement learning. The core of the proposed approach is a fine-tuning procedure that enables improvement through real-world experience. Using this algorithm, the approach outperformed existing methods for vision-based drone racing by a factor of two, and scored multiple times against three professional drone racing pilots, including the world champions of two drone racing leagues. This marks the first time that a computer program has defeated a human professional in a real-world sport designed by and for humans, a feat previously thought to be at least a decade away.

My contribution to this work includes the conceptualization, development, and implementation of the perception system to detect racing gates in RGB images, the design and implementation of the Kalman filter, development and implementation of the reinforcement learning environment for the task of vision-based drone racing, the design and implementation of the reward, the conceptualization and implementation of the adaptation procedure for both the dynamics and observation, the implementation of the fine-tuning stage, integration of the system on the real robot, and the evaluation of the approach in both simulation and in real-world experiments.

### 2.3 Data-Driven Dynamics Models

Dynamics models are a core component to any simulation or state-of-the-art model-based estimation, planning, or control algorithm. Having access to accurate dynamics models therefore greatly benefits the performance of a robotic system.

Traditionally, dynamics models have been identified by using first-principles-based approaches. Such models are typically efficient to compute, generalize well, but provide limited modeling accuracy. With the advent of deep learning, a new interest into learning-based dynamics models has emerged. By exploiting the approximation capabilities of data-driven approaches, highly complex dynamics models can be identified purely from data.

When designing data-driven models, the application domain imposes a set of constraints on modeling complexity, which results in different design decisions for different tasks. As such, in this thesis two different approaches to data-driven dynamics model learning for high-speed quadrotor flight are presented: (i) Paper I presents an approach to aerodynamic residual model learning using Gaussian processes that is tailored for real-time capability when deployed in the optimization loop of a model predictive controller. (ii) Paper J proposes instead the usage of deep neural networks for dynamics modeling, which purely maximizes for modeling accuracy at the cost of computational efficiency. Both approaches use a data-driven approach to identify a *residual* dynamics model, which is then complemented by a simple first-principle dynamics model (Paper I), or by a state-of-the-art propeller model based on blade-element-momentum theory (Paper J).





**Figure 2.10** – Our quadrotor platform reaches its physical limits at a pitch angle of 80 degrees while performing a lemniscate trajectory in our experiments. Throughout the trajectory, the platform reaches speeds of up to  $14\text{m s}^{-1}$  and accelerations beyond  $4g$ .

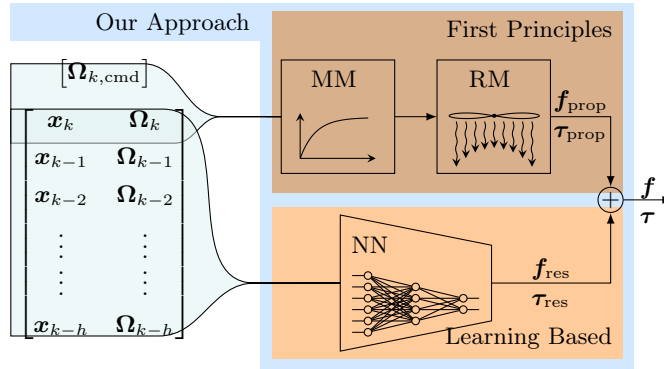
### 2.3.1 Paper I: Data-Driven MPC for Quadrotors

Guillem Torrente\*, Elia Kaufmann\*, Philipp Foehn, and Davide Scaramuzza. “Data-driven mpc for quadrotors”. In: *IEEE Robot. Autom. Lett.* 6.2 (2021), pp. 3769–3776

Aerodynamic forces render accurate high-speed trajectory tracking with quadrotors extremely challenging. These complex aerodynamic effects become a significant disturbance at high speeds, introducing large positional tracking errors, and are extremely difficult to model. To fly at high speeds, feedback control must be able to account for these aerodynamic effects in real-time. This necessitates a modeling procedure that is both accurate and efficient to evaluate. In this work, I present an approach to model residual aerodynamic effects using Gaussian Processes, which is incorporated into a Model Predictive Controller to achieve efficient and precise real-time feedback control. The method is verified by extensive comparison to a state-of-the-art linear drag model in synthetic and real-world experiments at speeds of up to  $50\text{km/h}$  and accelerations beyond  $4g$ .

This work was done in context of the master thesis of Guillem Torrente, who I supervised. My contribution to this work includes the conceptualization of the approach, the design of the experiments and the reference trajectories, and the implementation of the system on the real-world platform.

**Supplementary Video:** <https://youtu.be/FHvDghUUQtc>



**Figure 2.11** – Overview of the proposed architecture to predict aerodynamic forces and torques. The physical modeling pipeline (upper part) consists of a motor model (MM) and a rotor model (RM). It takes the current state  $\mathbf{x}_k$ , current motor speeds  $\Omega_k$ , and the motor speed command  $\Omega_{k,\text{cmd}}$  as an input. Combined with the estimate of the residual forces and torques predicted by the neural network (NN) using the current and past  $h$  states, the acting force  $\mathbf{f}$  and torque  $\boldsymbol{\tau}$  are calculated.

### 2.3.2 Paper J: NeuroBEM: Hybrid Aerodynamic Quadrotor Model

Leonard Bauersfeld\*, Elia Kaufmann\*, Philipp Foehn, Sihao Sun, and Davide Scaramuzza. “NeuroBEM: Hybrid Aerodynamic Quadrotor Model”. In: *RSS: Robotics, Science, and Systems* (2021)

Extending the methodology of Paper I, this work investigates the limits of modeling accuracy when no real-time constraints are imposed. To this end, I propose to combine a state-of-the-art propeller model based on blade-element-momentum theory with a residual wrench component predicted by a deep neural network that observes a history of state observations. The resulting hybrid approach fusing first principles and learning achieves unprecedented accuracy in modeling quadrotors and their aerodynamic effects over the entire performance envelope of the platform. Our hybrid approach unifies *and outperforms* both first-principles blade-element momentum theory and learned residual dynamics. It is identified and tested using autonomous-quadrotor-flight data at speeds up to 65 km/h. The resulting model captures the aerodynamic thrust, torques, and parasitic effects with astonishing accuracy, outperforming existing models with 50% reduced prediction errors, and shows strong generalization capabilities beyond the training set.

This work was done in context of the master thesis of Leonard Bauersfeld, who I supervised. My contribution to this work includes the conceptualization of the approach, the design and implementation of the neural network that predicts residual forces and torques, and the design and implementation of the reference maneuvers used for model identification.

**Supplementary Video:** <https://youtu.be/Nze1wfmzTQ>

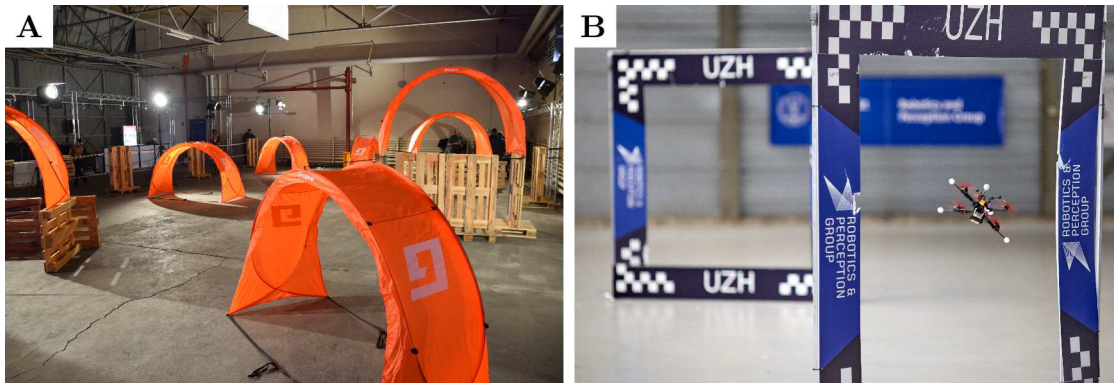


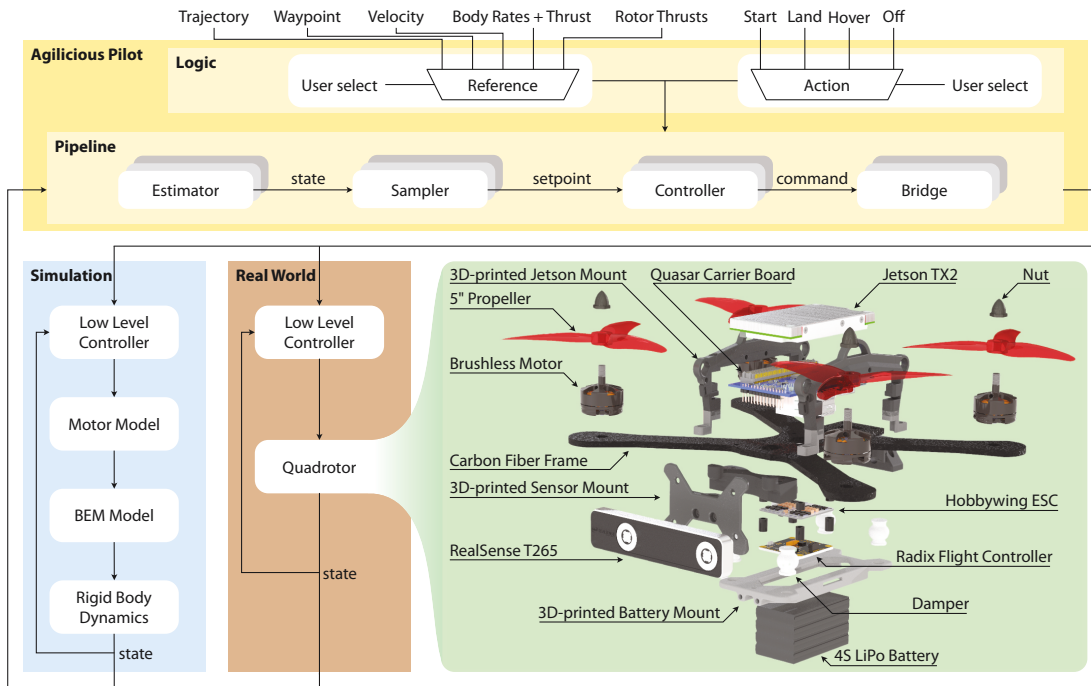
Figure 2.12 – Autonomous drone racing demonstrations given at the inauguration of the Switzerland Innovation Park (A) in March 2018 and the Scientifica event (B) in September 2021.

## 2.4 Additional Contributions

During my time at the Robotics and Perception Group, I contributed to some notable projects unrelated to the main topic of the thesis. These projects include *Agilicious*, an open-source and open-hardware quadrotor platform, as well as a publication on high-speed exploration using quadrotors.

**Real-World Demonstrators.** Throughout my journey as a Ph.D. student, I have participated in several public demonstrations of my research. Such live demonstrations are one of the best ways to communicate research in an approachable way to the general public. Additionally, it helps to push the development of robust systems that are lightweight and repeatably perform well in real-world conditions. Notable examples of such public demonstrations include the inauguration of the Switzerland Innovation Park<sup>1</sup> (Figure 2.12A) in March 2018 and the Scientifica event (Figure 2.12B) in September 2021. While the inauguration of the Switzerland Innovation Park hosted mostly representatives from industry and the Swiss government, Scientifica is an event held every two years that targets the general public and especially children and young adults. The primary goal of Scientifica is to communicate research as well as enthusiasm for science to visitors in an attractive and understandable way. During both events, I demonstrated autonomous drone racing with only onboard sensing and computation on a challenging race track to an audience of over 300 people.

<sup>1</sup><https://www.switzerland-innovation.com/zurich/>



**Figure 2.13** – Illustration of the Agilicious software and hardware design. Both software and hardware are tailored for agile flight while featuring powerful onboard compute capabilities through an NVIDIA Jetson TX2. As a key feature, the software of Agilicious is built in a modular fashion, allowing rapid software prototyping in simulation and seamless transition to real-world experiments.

### 2.4.1 Paper K: Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

Philipp Foehn\*, Elia Kaufmann\*, Angel Romero, Robert Penicka, Sihao Sun, Leonard Bauersfeld, Thomas Laengle, Yunlong Song, Antonio Loquercio, and Davide Scaramuzza. “Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight”. In: *Science Robotics* (2021). under review

Agile flight comes with ever-increasing engineering challenges since performing faster maneuvers with an autonomous system requires more capable algorithms, specialized hardware, and proficiency in system integration. As a result, only a small number of research groups have undertaken the significant overhead of hardware and software engineering, and have developed the expertise and resources to design quadrotor platforms that fulfill the requirements on weight, sensing, and computational budget necessary for autonomous agile flight.

In this work, I present an open-source and open-hardware quadrotor platform designed for research on agile vision-based flight. Developed over two years as a collaborative project of multiple Ph.D. students and engineers at the Robotics and Perception Group, the Agilicious software and hardware stack has been extensively tested in simulation scenarios, in real-world flight tests in motion capture settings, and vision-based outdoor



**Figure 2.14** – Illustration of the quadrotor autonomously exploring a forest. The quadrotor reached speeds up to  $2 \text{ m s}^{-1}$ .

experiments.

My contribution to this work includes contributions to the core-library, such as an interface for learning-based policies, a safety controller to catch the robot when an experiment fails, the implementation of polynomial and sampled trajectories, and others. Furthermore, I contributed key design decisions to the hardware platform, such as crash-resilient sensor mountings, selection of the main compute board, and identification and integration of breakout boards and the primary camera. I provided several experimental demonstrators in the tracking arena and the real world, including acrobatic flight, high-speed flight in the wild, and high-speed trajectory tracking.

#### 2.4.2 Paper L: Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

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, **Best Search and Rescue Paper Award Finalist**

Exploring and mapping previously unknown environments while avoiding collisions with obstacles is a fundamental task for autonomous robots. In scenarios where this needs to be done rapidly, multi-rotors are a good choice for the task, as they can cover ground at potentially very high velocities. Flying at high velocities, however, implies the ability to rapidly plan trajectories and to react to new information quickly. In this paper, I propose an extension to classical frontier-based exploration that facilitates exploration at high speeds. The extension consists of a reactive mode in which the multi-rotor rapidly selects a goal frontier from its field of view. The goal frontier is selected in a way that minimizes the change in velocity necessary to reach it. While this approach can increase the total path length, it significantly reduces the exploration time, since the multi-rotor can fly at consistently higher speeds.

## Chapter 2. Contributions

---

Although this work did not directly contribute to the main topic of this thesis, it allowed me to better understand the assumptions and limitations of traditional approaches for vision-based flight. This work was done in context of my master thesis under the supervision of Titus Cieslewski. My contribution to this work includes the conceptualization of the approach, its implementation in simulation and the real world, and its evaluation in simulation and real-world experiments in the forest.

## 3 Future Directions

At the time this doctoral work started, autonomous vision-based quadrotors were mostly research platforms that were constrained to slow, near-hover maneuvers. As of the time this thesis is completed, the entire field of autonomous quadrotor flight has substantially progressed, especially when considering situations where external sensing is available, such as instrumented tracking volumes. Even though this thesis makes contributions towards achieving the same level of agility while only relying on onboard sensing and computation, this challenge is far from being solved. In that context, I present some limitations of the proposed approaches and interesting avenues for future work.

**Generalization.** Although this thesis presents promising approaches to push the frontier of vision-based navigation, there are still open challenges that need to be addressed. One example of such a challenge is *generalization*: even though this thesis presents methods that allow for high-speed vision-based flight in scenarios such as acrobatic flight, agile flight in cluttered environments, and drone racing, it is not clear how these sensorimotor policies can be efficiently transferred to novel task formulations, sensor configurations, or physical platforms. Following the methodologies presented in this thesis, transferring to any of these new scenarios would require to retrain the policy in simulation, or perform adaptation using learned residual models. While the former suffers from the need to re-identify observation models and dynamics models, the latter is restricted to policy transfer between simulation and reality for the same task.

Possible avenues for future work to address this challenge include hierarchical learning [119] or approaches that optimize policy parameters on a learned manifold [267].

**Continual Learning.** The approaches to sensorimotor policy training proposed in this thesis are *static* in their nature: after the policy is trained in simulation via imitation learning or reinforcement learning, its parameters are frozen and applied on the real robot. In contrast, natural agents interact fundamentally different with their environment; they continually adapt to new experience and improve their performance in a much more dynamic fashion. Designing an artificial agent with similar capabilities would greatly increase the utility of robots in the real world and is an interesting direction for future work.

The adaptation method proposed in Paper H represents a first step towards this direction, although the adaptation is still performed in an iterative fashion. Recent work has

proposed methods to enable artificial agents to perform few-shot learning of new tasks and scenarios using techniques such as adaptive learning [319, 186] or meta learning [11, 87]. These methods have shown promising results on simple manipulation and locomotion tasks but remain to be demonstrated on complex navigation tasks such as high-speed flight in the real world.

**Autonomous Drone Racing.** The approaches presented in this thesis addressing the challenge of autonomous drone racing are limited to single-agent time trial races. To achieve true racing behaviour, these approaches need to be extended to a multi-agent setting, which raises novel challenges in perception, planning, and control. Regarding perception, multiagent drone racing requires to detect opponent agents, which is a challenging task when navigating at high speeds and when onboard observations are subject to motion blur. The planning challenges arise from the need to design game-theoretic strategies that involve maneuvers such as strategic blocking, which are potentially not time-optimal but still dominant approaches when competing in a multi-agent setting. Additionally, the limited field of view of the onboard camera renders opponents potentially unobservable, which requires a mental model of the trajectory of opponents. Specifically, it requires to predict the behaviour of opponents. Finally, racing simultaneously with other drones through a race track poses new challenges in modeling and control, as aerodynamic effects induced by other agents need to be accounted for.

I envision that many of these challenges can be addressed using deep reinforcement learning in combination with *self-play*, where agents improve by competing against each other [16, 316].



# A Deep Drone Racing: Learning Agile Flight in Dynamic Environments

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

Elia Kaufmann\*, Antonio Loquercio\*, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. “Deep Drone Racing: Learning Agile Flight in Dynamic Environments”. In: *Conf. on Robotics Learning (CoRL)*. 2018

# Deep Drone Racing: Learning Agile Flight in Dynamic Environments

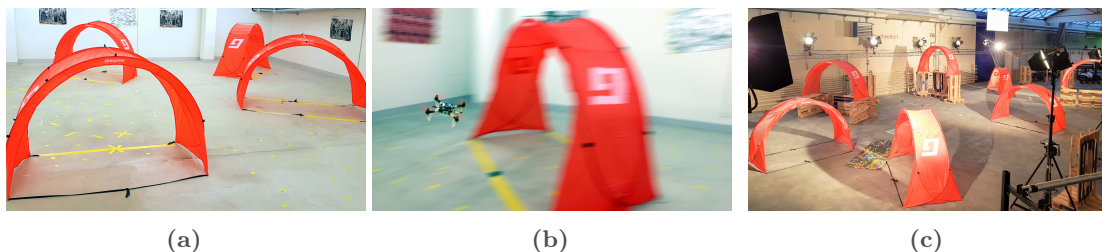
Elia Kaufmann\*, Antonio Loquercio\*, René Ranftl, Alexey Dosovitskiy,  
Vladlen Koltun, Davide Scaramuzza

**Abstract** — Autonomous agile flight brings up fundamental challenges in robotics, such as coping with unreliable state estimation, reacting optimally to dynamically changing environments, and coupling perception and action in real time under severe resource constraints. In this paper, we consider these challenges in the context of autonomous, vision-based drone racing in dynamic environments. Our approach combines a convolutional neural network (CNN) with a state-of-the-art path-planning and control system. The CNN directly maps raw images into a robust representation in the form of a waypoint and desired speed. This information is then used by the planner to generate a short, minimum-jerk trajectory segment and corresponding motor commands to reach the desired goal. We demonstrate our method in autonomous agile flight scenarios, in which a vision-based quadrotor traverses drone-racing tracks with possibly moving gates. Our method does not require any explicit map of the environment and runs fully onboard. We extensively test the precision and robustness of the approach in simulation and in the physical world. We also evaluate our method against state-of-the-art navigation approaches and professional human drone pilots.

## A.1 Introduction

Drone racing has become a popular televised sport with high-profile international competitions. In a drone race, each vehicle is controlled by a human pilot, who receives a first-person-view live stream from an onboard camera and flies the drone via a radio transmitter. Human drone pilots need years of training to master the advanced navigation and control skills that are required to be successful in international competitions. Such skills would also be valuable to autonomous systems that must quickly and safely fly through complex environments, in applications such as disaster response, aerial delivery, and inspection of complex structures.

We imagine that in the near future fully autonomous racing drones will compete against human pilots. However, developing a fully autonomous racing drone is difficult due to challenges that span dynamics modeling, onboard perception, localization and mapping, trajectory generation, and optimal control.



**Figure A.1** – By combining a convolutional neural network with state-of-the-art trajectory generation and control methods, our vision-based, autonomous quadrotor is able to successfully navigate a race track with moving gates with high agility.

A racing drone must complete a track in the shortest amount of time. One way to approach this problem is to accurately track a precomputed global trajectory passing through all gates. However, this requires highly accurate state estimation. Simultaneous Localization and Mapping (SLAM) systems [41] can provide accurate pose estimates against a previously-generated, globally-consistent map. These approaches may fail, however, when localizing against a map that was created in significantly different conditions, or during periods of high acceleration (because of motion blur and loss of feature tracking). Additionally, enforcing global consistency leads to increased computational demands and significant difficulties in coping with dynamic environments. Indeed, SLAM methods enable navigation only in a predominantly-static world, where waypoints and (optionally) collision-free trajectories can be statically defined. In contrast, drone races (and related applications of flying robots) can include moving gates and dynamic obstacles.

In this paper, we take a step towards autonomous, vision-based drone racing in dynamic environments. Our proposed approach is driven by the insight that methods relying on global state estimates in the form of robot poses are problematic due to the inherent difficulties of pose estimation at high speed along with inability to adequately cope with dynamic environments. As an alternative, we propose a hybrid system that combines the perceptual awareness of a convolutional neural network (CNN) with the speed and accuracy of a state-of-the-art trajectory generation and tracking pipeline. Our method does not require an explicit map of the environment. The CNN interprets the scene,

## Appendix A. Deep Drone Racing: Learning Agile Flight in Dynamic Environments

---

extracts information from a raw image, and maps it to a robust representation in the form of a waypoint and desired speed. This information is then used by the planning module to generate a short trajectory segment and corresponding motor commands to reach the desired local goal specified by the waypoint. The resulting approach combines the advantages of both worlds: the perceptual awareness and simplicity of CNNs and the precision offered by state-of-the-art controllers. The approach is both powerful and extremely lightweight: all computations run fully onboard.

Our experiments, performed in simulation and on a physical quadrotor, show that the proposed approach yields an integrated perception and control system that is able to cope with highly dynamic environments and severe occlusions, while being compact and efficient enough to be executed entirely onboard. The presented approach can perform complex navigation tasks, such as seeking a moving gate or racing through a track, with higher robustness and precision than state-of-the-art, highly engineered systems.

### A.2 Related Work

Pushing a robotic platform to high speeds poses a set of fundamental problems. Motion blur, challenging lighting conditions, and perceptual aliasing can cause severe drifts in any state estimation system. Additionally, state-of-the-art state estimation pipelines may require expensive sensors [37], have high computational costs [248], or be subject to drift due to the use of compressed maps [216]. Therefore, real-time performance is generally hindered when operating with resource constrained platforms, such as small quadrotors. This makes it extremely difficult to fully exploit the properties of popular minimum-snap or minimum-jerk trajectories [227, 242] for small, agile quadrotors using only onboard sensing and computing. Moreover, state-of-the-art state estimation methods are designed for a *predominantly-static world*, where no dynamic changes to the environment, or to the track to follow, occur.

In order to cope with dynamic environments, it is necessary to develop methods that tightly couple the perception and action loops. For the specific case of drone racing, this entails the capability to look for the target (the next gate) and localize relative to this while maintaining visual contact with it [83, 302]. However, traditional, handcrafted gate detectors quickly become unreliable in the presence of occlusions, partial visibility, and motion blur. The classical solution to this problem is visual servoing, where a robot is given a set of target locations in the form of reference images [335]. However, visual servoing only works well when the difference between the current and the reference images is small (which cannot be guaranteed at high speed) and is not robust to occlusions and motion blur.

An alternative solution consists of deriving actions directly from images using end-to-end trainable machine learning systems [283, 298, 159, 71, 156]. While being independent of any global map and position estimate, these methods are not directly applicable to our specific problem due to their high computational complexity [298], their low maximum speed [156] or the inherent difficulties of generalizing to 3D motions [283, 71]. Furthermore, the optimal output representation for learning-based algorithms that couple perception

and control is an open question. Known output representations range from predicting discrete navigation commands [178, 206]—which enables high robustness but leads to low agility—to direct control [159]—which can lead to highly agile control, but suffers from high sample complexity.

Taking the best of both worlds, this paper combines the benefits of agile trajectories with the ability of deep neural networks to learn highly expressive perception models, which are able to reason on high-dimensional, raw sensory inputs. The result is an algorithm that enables a resource-constrained, vision-based quadrotor to navigate a race track with possibly moving gates with high agility. While the supervision to train our neural network comes from global trajectory methods, the learned policy only operates on raw perceptual input, i.e., images, without requiring any information about the system’s global state. In addition, the “learner” acquires an ability that the “teacher” it imitates does not possess: it can cope with dynamic environments.

### A.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 from a forward-facing camera. The control system then uses these outputs to generate a minimum jerk trajectory [242] that is tracked by a low-level controller [77]. In what follows we describe the subsystems in more detail.

**Perception system** The goal of the perception system is to analyze the image and provide the flight direction to the control system. We implement the perception system by a convolutional network. The network takes as input a  $300 \times 200$  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 the efficient ResNet-based architecture of Loquercio et al. [206] (see the supplement for details). With our hardware setup, the network can process roughly 10 frames per second onboard. The system is trained by imitating an automatically computed expert policy, as explained in Section A.3.1.

**Control system** Given the tuple  $\{\vec{x}, v\}$ , the control system generates low-level control 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 A.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

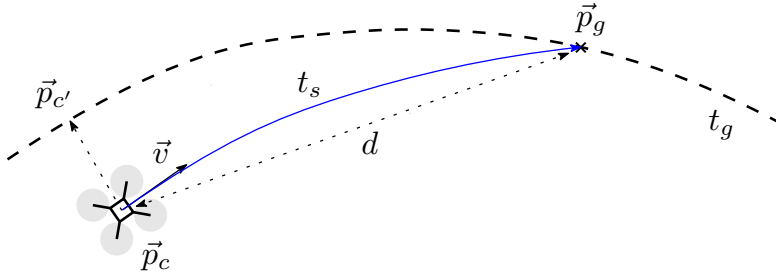


Figure A.2 – Notation used for the control system.

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 A.2). Since we run all computations onboard, we use computationally efficient minimum jerk trajectories [242] to generate  $t_s$ . To track  $t_s$ , i.e., to compute the low-level control commands, we deploy the control scheme proposed by Faessler et al. [77].

### A.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 this 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 A.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 [227]. 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, whose distance 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. The network is not directly imitating the demonstrated behavior, and therefore the performance of the learned policy is not upper-bounded by

the performance of the provided demonstrations.

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 manifest itself when the quadrotor flies far from the reference trajectory  $t_g$ .) In simulation, we employed a variant of DAgger [289], 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.

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

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

where  $\vec{x}_g$  denotes the groundtruth image coordinates and  $v_g$  denotes the groundtruth 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 the supplement 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 a variety of layouts, generated by slightly moving the gates from their initial position. We generate 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.

## A.4 Experiments in Simulation

We perform an extensive evaluation, both in simulation and on a physical system and compare our approach to a set of state-of-the-art baselines. We first present experiments in a controlled, simulated environment. The aim of these experiments is to get a sense of the capabilities of the presented approach, and compare to a direct end-to-end deep learning approach that regresses body rates based on image data. We use RotorS [102] and Gazebo for all simulation experiments.

### A.4.1 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 A.3a). We use this track to compare the

## Appendix A. Deep Drone Racing: Learning Agile Flight in Dynamic Environments

---



**Figure A.3** – Illustration of the small (a) and large (b) simulated tracks. The small track consists of 4 gates placed in a planar configuration and spans a total length of 43 meters. The large track consists of 8 gates placed at different heights and spans a total length of 116 meters.

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 [77].

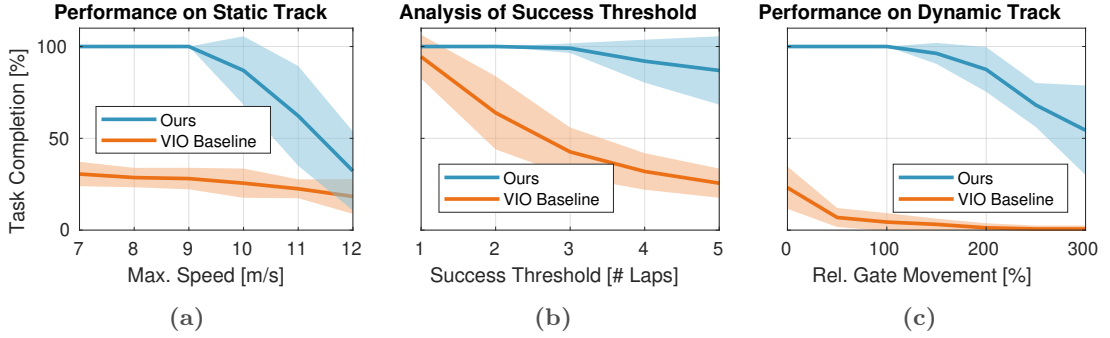
While our approach was always able to successfully complete the track, the naive baseline could never pass through more than one gate. Training on more data (35K samples, as compared to 5K samples used by our method) did not noticeably improve the performance of the baseline. In contrast to previous work [246], we believe that end-to-end learning of low-level controls is suboptimal for the task of drone navigation when operating in the real world. Indeed, the network has to learn the basic notion of stability from scratch in order to control an unstable platform such as a quadrotor [251]. This leads to high sample complexity, and gives no mathematical guarantees on the platform's stability. 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 than the camera images arrive and process them at a rate that is computationally infeasible with existing onboard hardware. In contrast, our approach can benefit from years of study in the field of control theory [133]. Because stability is handled by a classic controller, the network can focus on the main task of robust navigation, 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 the low-level controller, which enables 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.

### A.4.2 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 (Figure A.3b) with 8 gates and a length of 116 meters. 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 metric 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 baseline, we use a pure feedforward setting by





**Figure A.4 – 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. For each speed 10 runs were performed. A task completion rate of 100% is achieved if the drone can complete five consecutive laps without crashing. **b)** Analysis of the influence of the choice of success threshold. The experimental setting is the same as in Figure A.4a, but the performance is reported for a fixed maximum speed of  $10 \text{ m s}^{-1}$  and different success thresholds. The  $y$ -axis is shared with Figure A.4a. **c)** Result of our approach when flying through a simulated track with moving gates. Every gate independently moves with a sinusoidal pattern whose amplitude is equal to its base size (1.3 m) times 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. The  $y$ -axis is shared with Figure A.4a. The reader is encouraged to watch the supplementary video to better understand the experimental setup and task difficulty.

following the global trajectory  $t_g$  using visual inertial odometry [97].

The results of this experiment are shown in Figure A.4a. We can observe that the VIO baseline performs inferior compared to our approach, on both the static and dynamic track. On the static track, the VIO baseline fails due to the accumulated drift, as shown in Figure A.4b. While the VIO baseline performs comparably when one single lap is considered a success, the performance degrades rapidly if the threshold for success is raised to more laps. Our approach reliably works up to a maximum speed of  $9 \text{ m s}^{-1}$ , while the performance gracefully degrades 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.

### A.4.3 Generalization to dynamic environments

The learned policy has a characteristic that the expert policy lacks: coping with dynamic environments. In those, waypoints and collision-free trajectories cannot be defined *a priori*. To quantitatively test this ability, we reuse the track layout from the previous experiment (Figure A.3b), but dynamically move each gate according to a sinusoidal pattern in each dimension independently. Figure A.4c 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 A.4.2. For this experiment,

## Appendix A. Deep Drone Racing: Learning Agile Flight in Dynamic Environments

---



Figure A.5 – 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 A.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 \*.

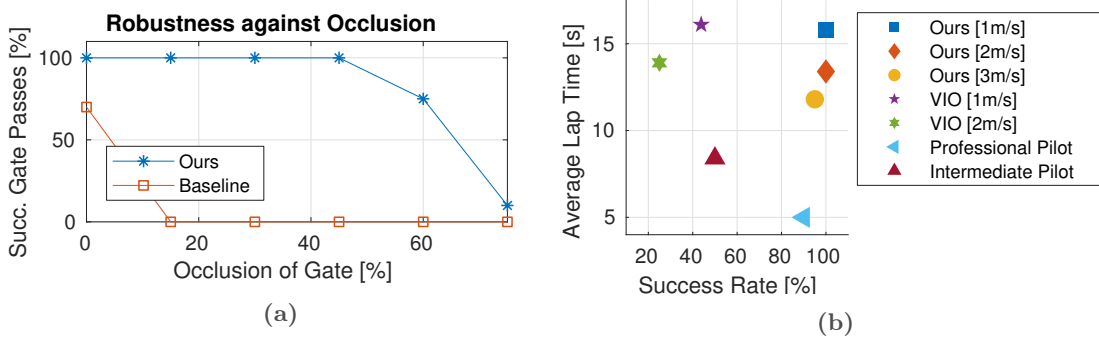
we kept the maximum platform velocity  $v_{max}$  constant at  $8 \text{ m s}^{-1}$ . Despite the high speed, our approach can handle dynamic gate movements of up to 1.5 times the gates’ diameter without crashing. In contrast, the VIO baseline (*i.e.* the expert policy) cannot adapt to changes in the environment, and fails even for tiny gate motions. The performance of our approach gracefully degrades for gate movements larger than 1.5 times the gates’ 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 our approach reactively adapts to dynamic changes and generalizes well to cases where the track remains roughly similar to the one collected data from.

### A.5 Experiments in the Physical World

To show the ability of our approach to control real quadrotors, we performed experiments on a physical platform. We compare our approach to state-of-the-art classic approaches to robot navigation, as well as to human drone pilots of different skill levels. For these experiments, we collected data in the real world. Technical details on the platform used can be found in the supplement.

In a first 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 A.5. We compare our approach to the handcrafted window detector of Falanga et al. [83] by replacing our perception system (Section A.3) with the handcrafted detector and leaving the control system (Section A.3) unchanged.

## A.5. Experiments in the Physical World



**Figure A.6** – **a)** Success rate for different amount of occlusion of the gate. The area is calculated on the entire size of the gate. At more than 60% occlusion, the platform has barely any space to pass through the gap. **b)** 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 drone pilot. We evaluate success rate using the same metric as explained in Section A.4.2.

Table A.1 shows a comparison between our approach and the baseline. We test the robustness of both approaches to the initial position of the quadrotor. To do so we place 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 measure average success rate to pass the gate without crashing. The experiments indicate that our approach is robust to initial conditions. The drone is able to pass the gate consistently, even if the gate is only partially visible. By contrast, the handcrafted baseline cannot detect the gate if it’s not entirely in the field of view. The baseline sometimes fails even if the gate is fully visible because the window detector loses tracking due to platform vibrations.

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 never seen these configurations during training. Figure A.6a shows that our approach is robust to occlusions of up to 50% of the total area of the gate (Figure A.5), 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.

### A.5.1 Experiments on a race track

In the last set of experiments we challenge the system to race through a track with either static or dynamics gates. The track is shown in Figure A.1a. 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 diverse set of baselines, such as a classic approach based

## Appendix A. Deep Drone Racing: Learning Agile Flight in Dynamic Environments

---

on planning and tracking [203] and human pilots of different skill levels. Note that due to the smaller size of the real track compared to the simulated one, the maximum speed achieved in real world experiments is lower than in simulation. For our baseline, we use a state-of-the-art visual-inertial odometry approach [203] to provide global state estimates in order to track the global reference trajectory.

Figure A.6b summarizes the quantitative results of our evaluation, where we measure success rate (completing five consecutive laps without crashing), as well as the best lap time. Our learning-based approach outperforms the visual odometry-based baseline, whose drift at high speeds inevitably leads to poor performance. By generating waypoint commands in body frame, instead, our approach is insensitive to state estimation drift, and can complete 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. It is evident from Figure A.6b that both the professional and the intermediate pilots were able to complete the track faster than the autonomous systems. The high speed and aggressive flight by human pilots comes at the cost of increased failure rates, however. 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 in sharp turns. Such maneuvers require a level of reasoning about the environment that our autonomous system still lacks.

In a last qualitative experiment, we manually moved gates while the quadrotor navigated through the track. This requires the navigation system to be able to reactively respond to dynamic changes. Note that moving gates break the main assumption of traditional high-speed navigation approaches [39, 101], 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<sup>1</sup>. As in the simulation experiments, the system can generalize to dynamically moving gates on the real track. It is worth noting that training data was collected by changing the position of only a single gate, but the network is able to cope with movement of any gate at test time.

### A.6 Discussion

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 a desired speed directly from raw images. These high-level commands are then executed by a classic control stack. To enable agile and fast flight, we train the network to follow a global reference trajectory. The system combines the robust perceptual awareness of modern machine learning pipelines with the stability and speed of well-known control algorithms.

---

<sup>1</sup>Available from: <http://youtu.be/8RILnqPxo1s>

We demonstrated the capabilities of this integrated approach to perception and control in an extensive set of experiments on real drones and in simulation. Our experiments show that the resulting system is able to robustly 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.

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.

Scaling such hybrid approaches to more general environments is an exciting avenue for future work that poses several challenges. First, while the ability of our system to navigate through moving or partially occluded gates is promising, performance will degrade if the appearance of the environment changes substantially beyond what was observed during training. Second, in order to train the perception system, our current approach requires a significant amount of data in the application environment. This might be acceptable in some scenarios, but not practical when fast adaptation to previously unseen environments is needed. This could be addressed with techniques such as few-shot learning. Third, in the cases where trajectory optimization cannot provide a policy to be imitated, for instance in the presence of extremely tight turns, the learner is also likely to fail. This issue could be alleviated by integrating learning deeper into the control system.



# B Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

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

Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. “Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)* (2019), pp. 690–696. DOI: [10.1109/ICRA.2019.8793631](https://doi.org/10.1109/ICRA.2019.8793631)

# Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy,  
Vladlen Koltun, Davide Scaramuzza

**Abstract** — Autonomous micro aerial vehicles still struggle with fast and agile maneuvers, dynamic environments, imperfect sensing, and state estimation drift. Autonomous drone racing brings these challenges to the fore. Human pilots can fly a previously unseen track after a handful of practice runs. In contrast, state-of-the-art autonomous navigation algorithms require either a precise metric map of the environment or a large amount of training data collected in the track of interest. To bridge this gap, we propose an approach that can fly a new track in a previously unseen environment without a precise map or expensive data collection. Our approach represents the global track layout with coarse gate locations, which can be easily estimated from a single demonstration flight. At test time, a convolutional network predicts the poses of the closest gates along with their uncertainty. These predictions are incorporated by an extended Kalman filter to maintain optimal maximum-a-posteriori estimates of gate locations. This allows the framework to cope with misleading high-variance estimates that could stem from poor observability or lack of visible gates. Given the estimated gate poses, we use model predictive control to quickly and accurately navigate through the track. We conduct extensive experiments in the physical world, demonstrating agile and robust flight through complex and diverse previously-unseen race tracks. The presented approach was used to win the IROS 2018 Autonomous Drone Race Competition, outracing the second-placing team by a factor of two.





**Figure B.1** – A quadrotor flies through an indoor track. Our approach uses optimal filtering to incorporate estimates from a deep perception system. It can race a new track after a single demonstration.

## B.1 Introduction

First-person view (FPV) drone racing is a fast-growing sport, in which human pilots race micro aerial vehicles (MAVs) through tracks via remote control. Drone racing provides a natural proving ground for vision-based autonomous drone navigation. This has motivated competitions such as the annual IROS Autonomous Drone Race [237] and the recently announced AlphaPilot Innovation Challenge, an autonomous drone racing competition with more than 2 million US dollars in cash prizes.

To successfully navigate a race track, a drone has to continually sense and interpret its environment. It has to be robust to cluttered and possibly dynamic track layouts. It needs precise planning and control to support the aggressive maneuvers required to traverse a track at high speed. Drone racing thus crystallizes some of the central outstanding issues in robotics. Algorithms developed for drone racing can benefit robotics in general and can contribute to areas such as autonomous transportation, delivery, and disaster relief.

Traditional localization-based approaches for drone navigation require precomputing a precise 3D map of the environment against which the MAV is localized. Thus, while previous works demonstrated impressive results in controlled settings [240], these methods are difficult to deploy in new environments where a precise map is not available. Additionally, they fail in the presence of dynamic objects such as moving gates, have inconsistent computational overhead, and are prone to failure under appearance changes such as varying lighting.

## Appendix B. Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

---

Recent work has shown that deep networks can provide drones with robust perception capabilities and facilitate safe navigation even in dynamic environments [171, 156]. However, current deep learning approaches to autonomous drone racing require a large amount of training data collected in the same track. This stands in contrast to human pilots, who can quickly adapt to new tracks by leveraging skills acquired in the past.

In this paper, we develop a deep-learning-aided approach to autonomous drone racing capable of fast adaptation to new tracks, without the need for building precise maps or collecting large amounts of data from the track. We represent a track by coarse locations of a set of gate, which can be easily acquired in a single demonstration flight through the track. These recorded gate represent the rough global layout of the track. At test time, the local track configuration is estimated by a convolutional network that predicts the location of the closest gate together with its uncertainty, given the currently observed image. The network predictions and uncertainties are continuously incorporated using an extended Kalman filter (EKF) to derive optimal maximum-a-posteriori estimates of gate locations. This allows the framework to cope with misleading high-variance estimates that could stem from bad observability or complete absence of visible gate. Given these estimated gate locations, we use model predictive control to quickly and accurately navigate through them.

We evaluate the proposed method in simulation and on a real quadrotor flying fully autonomously. Our algorithm runs onboard on a computationally constrained platform. We show that the presented approach can race a new track after only a single demonstration, without any additional training or adaptation. Integration of the estimated gate positions is crucial to the success of the method: a purely image-based reactive approach only shows non-trivial performance in the simplest tracks. We further demonstrate that the proposed method is robust to dynamic changes in the track layout induced by moving gates.

The presented approach was used to win the IROS Autonomous Drone Race Competition, held in October 2018. An MAV controlled by the presented approach placed first in the competition, traversing the eight gates of the race track in 31.8 seconds. In comparison, the second-place entry completed the track in 61 seconds, and the third in 90.1 seconds.

### B.2 Related Work

Traditional approaches to autonomous MAV navigation build on visual inertial odometry (VIO) [97, 32, 194, 349] or simultaneous localization and mapping (SLAM) [249, 306], which are used to provide a pose estimate of the drone relative to an internal metric map [203, 86]. While these methods can be used to perform visual teach and repeat [86], they are not concerned with trajectory generation [227, 243]. Furthermore, teach and repeat assumes a static world and accurate pose estimation: assumptions that are commonly violated in the real world.

The advent of deep learning has inspired alternative solutions to autonomous navigation that aim to overcome these limitations. These approaches typically predict actions directly

from images. Output representations range from predicting discrete navigation commands (classification in action space) [178, 112, 206] to direct regression of control signals [245]. A different line of work combines network predictions with model predictive control by regressing the cost function from a single image [71].

In the context of drone racing, Kaufmann et al. [171] proposed an intermediate representation in the form of a goal direction and desired speed. The learned policy imitates an optimal trajectory [227] through the track. An advantage of this approach is that it can navigate even when no gate is in view, by exploiting track-specific context and background information. A downside, however, is the need for a large amount of labeled data collected directly in the track of interest in order to learn this contextual information. As a result, the approach is difficult to deploy in new environments.

Jung et al. [156] consider the problem of autonomous drone navigation in a previously unseen track. They use line-of-sight guidance combined with a deep-learning-based gate detector. As a consequence, the next gate to be traversed has to be in view at all times. Additionally, gates cannot be approached from an acute angle since the algorithm does not account for gate rotation. The method is thus applicable only to relatively simple environments, where the next gate is always visible.

Our approach addresses the limitations of both works [171, 156]. It operates reliably even when no gate is in sight, while eliminating the need to retrain the perception system for every new track. This enables rapid deployment in complex novel tracks.

### B.3 Methodology

We address the problem of robust autonomous flight through a predefined, ordered set of possibly spatially perturbed gate. Our approach comprises three subsystems: perception, mapping, and combined planning and control. The perception system takes as input a single image from a forward-facing camera and estimates both the relative pose of the next gate and a corresponding uncertainty measure. The mapping system receives the output of the perception system together with the current state estimate of the quadrotor and produces filtered estimates of gate poses. The gate poses are used by the planning system to maintain a set of waypoints through the track. These waypoints are followed by a control pipeline that generates feasible receding-horizon trajectories and tracks them.

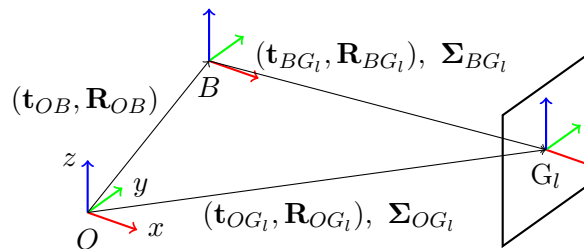


Figure B.2 – Relation of odometry  $O$ , body  $B$ , and gate frame  $G_l$ .

## Appendix B. Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

---

### B.3.1 Notation and Frame Convention

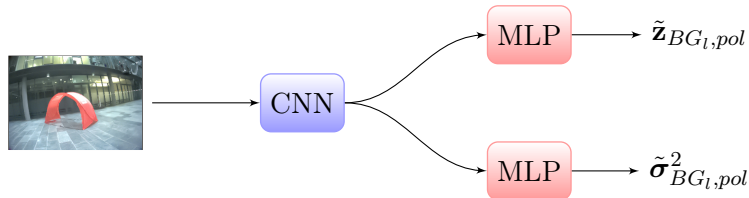
We denote all scalars by lowercase letters  $x$ , vectors by lowercase bold letters  $\mathbf{x}$ , and matrices by bold uppercase letters  $\mathbf{X}$ . Estimated values are written as  $\hat{x}$ , measured values as  $\tilde{x}$ .

The relevant coordinate frames are the odometry frame  $O$ , the body frame  $B$ , and the gate frames  $G_l$ , where  $l \in \{1, \dots, N_l\}$  and  $N_l$  is the number of gate. A schematic overview of the relation between coordinate frames is shown in Figure B.2. The odometry frame  $O$  is the global VIO reference frame. The relation between the body frame  $B$  and the odometry frame  $O$  is given by the rotation  $\mathbf{R}_{OB}$  and translation  $\mathbf{t}_{OB}$ . This transform is acquired through a visual inertial pose estimator. The prediction  $(\tilde{\mathbf{t}}_{BG_l}, \tilde{\mathbf{R}}_{BG_l})$  is provided together with a corresponding uncorrelated covariance in polar coordinates  $\tilde{\Sigma}_{BG_l, pol} = \text{diag}(\tilde{\sigma}_{BG_l, pol}^2)$  of the gate’s pose in the body frame. In parallel, we maintain an estimate of each gate pose  $(\hat{\mathbf{t}}_{OG_l}, \hat{\mathbf{R}}_{OG_l})$  along with its covariance  $\hat{\Sigma}_{OG_l} = \text{cov}(\hat{\mathbf{t}}_{OG_l}, \hat{\mathbf{R}}_{OG_l})$  in the odometry frame. This has the advantage that gate poses can be updated independently of each other.

### B.3.2 Perception System

#### Architecture

The deep network takes as input a  $320 \times 240$  RGB image and regresses both the mean  $\tilde{\mathbf{z}}_{BG_l, pol} = [\tilde{r}, \tilde{\theta}, \tilde{\psi}, \tilde{\phi}]^T \in \mathbb{R}^4$  and the variance  $\tilde{\sigma}_{BG_l, pol}^2 \in \mathbb{R}^4$  of a multivariate normal distribution that describes the current estimate of the next gate’s pose. Our choice of output distribution is motivated by the fact that we use an EKF to estimate the joint probability distribution of a gate’s pose, which is known to be optimal for identical and independently distributed white noise with known covariance. The mean represents the prediction of the relative position and orientation of the gate with respect to the quadrotor in spherical coordinates. We found this to be advantageous compared to a Cartesian representation since it decouples distance estimation from the position of the gate in image coordinates. We use a single angle  $\tilde{\phi}$  to describe the relative horizontal orientation of the gate, since the gravity direction is known from the IMU. Furthermore, we assume that gate are always upright and can be traversed horizontally along the normal direction. Specifically,  $\tilde{\phi}$  is measured between the quadrotor’s current heading and the gate’s heading.



**Figure B.3** – Schematic illustration of the network architecture. Image features are extracted by a CNN [206] and passed to two separate MLPs to regress  $\tilde{\mathbf{z}}_{BG_l, pol}$  and  $\tilde{\sigma}_{BG_l, pol}^2$ , respectively.

The overall structure of the deep network is shown in Figure B.3. First, the input

image is processed by a Convolutional Neural Network (CNN), based on the shallow DroNet architecture [206]. The extracted features are then processed by two separate multilayer perceptrons (MLPs) that estimate the mean  $\tilde{\mathbf{z}}_{BG_i, pol}$  and the variance  $\tilde{\sigma}_{BG_i, pol}^2$  of a multivariate normal distribution, respectively. A similar network architecture for mean-variance estimation was proposed in [255].

### Training Procedure

We train the network in two stages.

In the first stage, the parameters of the CNN and MLP $_{\mathbf{z}}$ , denoted by  $\theta_{\text{CNN}}$  and  $\theta_{\mathbf{z}}$ , are jointly learned by minimizing a loss over groundtruth poses for images with visible gate:

$$\{\theta_{\text{CNN}}^*, \theta_{\mathbf{z}_i}^*\} = \arg \min_{\theta_{\text{CNN}}, \theta_{\mathbf{z}_i}} \sum_{i=1}^N \|\mathbf{y}_i - \tilde{\mathbf{z}}_i\|_2^2, \quad (\text{B.1})$$

where  $\mathbf{y}_i$  denotes the groundtruth pose and  $N$  denotes the dataset size.

In the second stage, the training set is extended to also include images that do not show visible gate. In this stage only the parameters  $\theta_{\sigma^2}$  of the subnetwork MLP $_{\sigma^2}$  are trained, while keeping the other weights fixed. We minimize the loss function proposed by [255], which amounts to the negative log-likelihood of a multivariate normal distribution with uncorrelated covariance:

$$-\log p(\mathbf{y} | \tilde{\mathbf{z}}_i, \tilde{\sigma}^2) \propto \sum_{j=1}^4 \log \tilde{\sigma}_j^2 + \frac{(y_j - \tilde{z}_j)^2}{\tilde{\sigma}_j^2}. \quad (\text{B.2})$$

Our use of mean-variance estimation is motivated by studies that have shown that it is a computationally efficient way to obtain uncertainty estimates [177].

### Training Data Generation

We collect a set of images from the forward-facing camera on the drone and associate each image with the relative pose of the gate with respect to the body frame of the quadrotor. In real-world experiments, we use the quadrotor and leverage the onboard state estimation pipeline to generate training data. The platform is initialized at a known position relative to a gate and subsequently carried through the environment while collecting images and corresponding relative gate poses. To collect training data, it is not necessary to have complete tracks available. A single gate placed in different environments suffices, as the perception system only needs to estimate the relative pose with respect to the next gate at test time. Moreover, in contrast to Kaufmann et al. [171], the perception system is never trained on data from tracks and environments it is later deployed in.

## Appendix B. Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

---

### B.3.3 Mapping System

The mapping system takes as input a measurement from the perception system and outputs a filtered estimate of the current track layout. By correcting the gates with the measurements from the CNN, gate displacement and accumulated VIO drift can be compensated for. The mapping part of our pipeline can be divided into two stages: measurement assignment stage and filter stage.

#### Measurement Assignment

We maintain a map of all gate  $l = 1 \dots N_l$  with states  $\hat{\mathbf{x}}_{OG_l} = [\hat{\mathbf{t}}_{OG_l}, \hat{\phi}_{OG_l}]^\top$  corresponding to gate translation  $\hat{\mathbf{t}}_{OG_l}$  and yaw  $\hat{\phi}_{OG_l}$  with respect to the odometry frame  $O$ . The output of the perception system is used to update the pose  $\hat{\mathbf{x}}_{OG_l}$  of the next gate to be passed. To assign a measurement to a gate, the measurement is transformed into the odometry frame and assigned to the closest gate. If a measurement is assigned to a gate that is not the next gate to be passed, it is discarded as an outlier. We keep track of the next gate by detecting gate traversals. The detection of a gate traversal is done by expressing the quadrotor's current position in a gate-based coordinate frame. In this frame, the condition for traversal can be expressed as

$$G_l \hat{\mathbf{t}}_{G_l B, x} \geq 0. \quad (\text{B.3})$$

#### Extended Kalman Filter

The prediction of the network in body frame  $B$  is given by  $\tilde{\mathbf{z}}_{BG, pol} = [\tilde{r}, \tilde{\theta}, \tilde{\psi}, \tilde{\phi}]^\top$  containing the spherical coordinates  $[\tilde{r}, \tilde{\theta}, \tilde{\psi}]^\top$  and yaw  $\tilde{\phi}$  of the gate, and the corresponding variance  $\tilde{\sigma}_{BG, pol}^2$ . The transformation into the Cartesian representation  $\tilde{\mathbf{z}}_{BG}$  leads to

$$\tilde{\mathbf{z}}_{BG} = \mathbf{f}(\tilde{\mathbf{z}}_{BG, pol}) = \begin{bmatrix} \tilde{r} \sin \tilde{\theta} \cos \tilde{\psi} \\ \tilde{r} \sin \tilde{\theta} \sin \tilde{\psi} \\ \tilde{r} \cos \tilde{\theta} \\ \tilde{\phi} \end{bmatrix} \quad (\text{B.4})$$

$$\tilde{\Sigma}_{BG} = \mathbf{J}_{\mathbf{f}}|_{\tilde{\mathbf{z}}_{pol}} \tilde{\Sigma}_{BG, pol} \mathbf{J}_{\mathbf{f}}^\top|_{\tilde{\mathbf{z}}_{pol}}, \quad (\text{B.5})$$

where  $\mathbf{J}_{\mathbf{f}, i, j} = \frac{\partial f_i}{\partial x_{pol, j}}$  is the Jacobian of the conversion function  $\mathbf{f}$  and  $\mathbf{J}_{\mathbf{f}}|_{\mathbf{z}_{pol}}$  is its evaluation at  $\mathbf{z}_{pol}$ . To integrate neural network predictions reliably into a map with prior knowledge of the gate, we represent each gate with its own EKF. We treat the prediction  $\tilde{\mathbf{z}}_{BG}$  and  $\tilde{\Sigma}_{BG}$  at each time step as a measurement and associated variance, respectively. Similar to the state,  $\tilde{\mathbf{z}}_{BG} = [\tilde{\mathbf{t}}_{BG}^\top, \tilde{\phi}_{BG}]^\top$  consists of a translation  $\tilde{\mathbf{t}}_{BG}$  and rotation  $\tilde{\phi}_{BG}$  around the world  $z$ -axis. Since our measurement and states have different origin frames,

we can formulate the EKF measurement as follows:

$$\begin{aligned} \tilde{\mathbf{z}}_k &= \mathbf{H}_k \hat{\mathbf{x}}_k + \mathbf{w}, \quad \mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k) \\ \mathbb{E}[\tilde{\mathbf{z}}_k] &= \begin{bmatrix} \mathbf{R}_{OB,k}^{-1} \mathbf{o}\mathbf{t}_{OG,k} - \mathbf{R}_{OB,k}^{-1} \mathbf{o}\mathbf{t}_{OB,k} \\ \phi_{OG,k} - \phi_{OB,k} \end{bmatrix}. \end{aligned} \quad (\text{B.6})$$

Now with  $\hat{\mathbf{x}}_k = [\mathbf{o}\mathbf{t}_{OG,k}, \phi_{OG,k}]^\top$  we can write

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{R}_{OB,k}^{-1} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \quad (\text{B.7})$$

$$\boldsymbol{\mu}_k = \begin{bmatrix} -\mathbf{R}_{OB,k}^{-1} \mathbf{o}\mathbf{t}_{OB,k} \\ -\phi_{OB,k} \end{bmatrix} \quad \boldsymbol{\Sigma}_k = \tilde{\boldsymbol{\Sigma}}_{BG,k} \quad (\text{B.8})$$

and, due to identity process dynamics and process covariance  $\boldsymbol{\Sigma}_Q$ , our prediction step becomes

$$\hat{\mathbf{x}}_{k+1}^* = \hat{\mathbf{x}}_k \quad \hat{\mathbf{P}}_{k+1}^* = \hat{\mathbf{P}}_k + \boldsymbol{\Sigma}_Q. \quad (\text{B.9})$$

The a-posteriori filter update can be summarized as follows:

$$\begin{aligned} \mathbf{K}_k &= \hat{\mathbf{P}}_k^* \mathbf{H}_k \left( \tilde{\boldsymbol{\Sigma}}_{BG,k} + \mathbf{H}_k \hat{\mathbf{P}}_k^* \mathbf{H}_k^\top \right)^{-1} \\ \hat{\mathbf{x}}_{k+1} &= \hat{\mathbf{x}}_k^* + \mathbf{K}_k (\tilde{\mathbf{z}}_k - \boldsymbol{\mu}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^*) \\ \hat{\mathbf{P}}_{k+1} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \hat{\mathbf{P}}_k^* (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^\top + \mathbf{K}_k \tilde{\boldsymbol{\Sigma}}_{BG,k} \mathbf{K}_k^\top \end{aligned} \quad (\text{B.10})$$

with  $\hat{\mathbf{P}}_k^*$  as the estimated covariance and the superscript \* indicating the a-priori predictions.

### B.3.4 Planning and Control System

The planning and control stage is split into two asynchronous modules. First, low-level waypoints are generated from the estimated gate position and a desired path is generated by linearly interpolating between the low-level waypoints. Second, locally feasible control trajectories are planned and tracked using a model predictive control scheme.

#### Waypoint Generation

For each gate in our map we generate two waypoints: one lying in front of the gate relative to the current quadrotor position and one lying after the gate. Both waypoints are set with a positive and negative offset  $\mathbf{p}_{wp,l\pm}$  in the  $x$  direction with respect to the gate  $l$ :

$$\mathbf{p}_{wp,l\pm} = \mathbf{o}\mathbf{t}_{OG_l} + \mathbf{R}_{OG_l} [\pm x_G, 0, 0]^\top, \quad (\text{B.11})$$

where  $x_G$  is a user-defined constant accounting for the spatial dimension of gate  $l$ . We then linearly interpolate a path from waypoint to waypoint and use it as a reference for

## Appendix B. Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

---

our controller.

### Model Predictive Control

We formulate the control problem as a quadratic optimization problem which we solve using sequential quadratic programming as described in [79]:

$$\begin{aligned} \min_{\mathbf{u}} \int_{t_0}^{t_f} & \left( \bar{\mathbf{x}}_t^\top(t) \mathbf{Q} \bar{\mathbf{x}}_t(t) + \bar{\mathbf{u}}_t^\top(t) \mathbf{R} \bar{\mathbf{u}}_t(t) \right) dt \\ & \bar{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x}_r(t) \quad \bar{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u}_r(t) \\ \text{subject to} \quad & \mathbf{r}(\mathbf{x}, \mathbf{u}) = 0 \quad \mathbf{h}(\mathbf{x}, \mathbf{u}) \leq 0. \end{aligned}$$

The states  $\mathbf{x}$  and inputs  $\mathbf{u}$  are weighted with positive diagonal matrices  $\mathbf{Q}$  and  $\mathbf{R}$  with respect to a reference  $\mathbf{x}_r$  and  $\mathbf{u}_r$ . The equality and inequality constraints,  $\mathbf{r}$  and  $\mathbf{h}$  respectively, are used to incorporate the vehicle dynamics and input saturations. The reference is our linearly sampled path along which the MPC finds a feasible trajectory. Note that we can run the control loop independent of the detection and mapping pipeline and reactively stabilize the vehicle along the changing waypoints.

## B.4 Experimental Setup

We evaluate the presented approach in simulation and on a physical system.

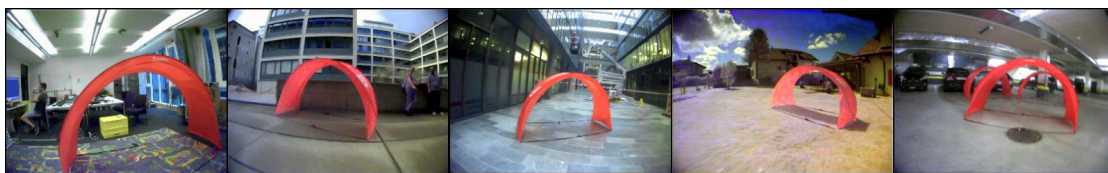
### B.4.1 Simulation

We use RotorS [102] and Gazebo [182] for all simulation experiments. To train the perception system, we generated 45,000 training images by randomly sampling camera and gate positions and computing their relative poses. For quantitative evaluation, a 100% successful trial is defined as completing 3 consecutive laps without crashing or missing a gate. If the MAV crashes or misses a gate before completing 3 laps, the success rate is measured as a fraction of completed gate out of 3 laps: for instance, completing 1



**Figure B.4** – Our platform, equipped with an Intel UpBoard and a Qualcomm Snapdragon Flight.





**Figure B.5** – We collected training data for the perception system in 5 different environments. From left to right: flying room, outdoor urban environment, atrium, outdoor countryside, garage.

lap counts as 33.3% success.

### B.4.2 Physical System

In all real-world experiments and data collection we use an in-house MAV platform with an Intel UpBoard<sup>1</sup> as the main computer running the CNN, EKF, and MPC. Additionally we use a Qualcomm Snapdragon Flight<sup>2</sup> as a visual-inertial odometry unit. The platform is shown in Fig. B.4. The CNN reaches an inference rate of  $\sim 10$  Hz while the MPC runs at 100 Hz. With a take-off weight of 950 g the platform reaches thrust-to-weight ratio of  $\sim 3$ .

We collect training data for the perception system in five different environments, both indoors and outdoors. Example images from the environments are shown in Fig. B.5. In total, we collected 32,000 images.

## B.5 Results

Results are shown in the supplementary video: <https://youtu.be/UuQvijZcUSc>

### B.5.1 Simulation

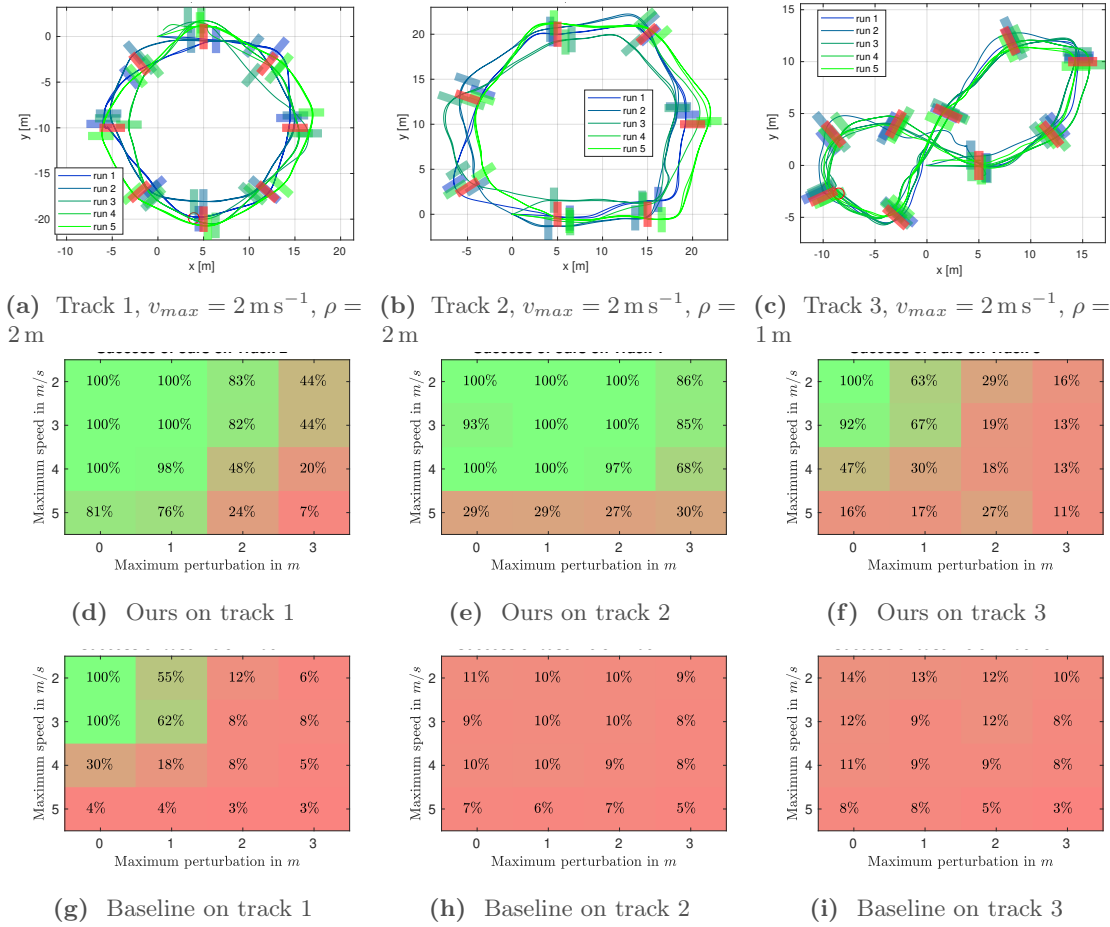
We first present experiments in a controlled, simulated environment. The aim of these experiments is to thoroughly evaluate the presented approach both quantitatively and qualitatively and compare it to a baseline – the method of Jung et al. [156]. The baseline was trained on the same data as our approach.

We evaluate the two methods on three tracks of increasing difficulty. Figs. B.6a-c show an illustration of the three race tracks and plot the executed trajectories together with the nominal gate positions in red and the actual displaced gate positions in the corresponding track color. Our approach achieved successful runs in all environments, with speeds up to  $4 \text{ m s}^{-1}$  in the first two tracks. Additionally, gate displacement was handled robustly up to a magnitude of 2 m before a significant drop in performance occurred. Figs. B.6d-i show the success rate of our method and the baseline on the three tracks, under varying speed and track perturbations. Our approach outperforms the baseline by a large margin

<sup>1</sup><https://www.up-board.org/up/>

<sup>2</sup><https://developer.qualcomm.com/hardware/qualcomm-flight>

## Appendix B. Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

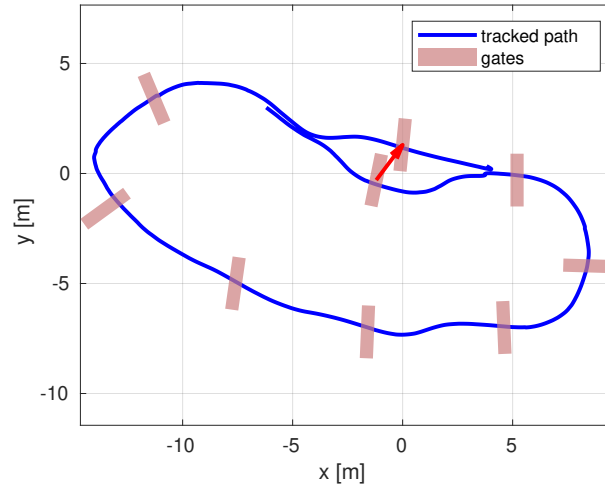


**Figure B.6** – Results of the simulation experiments. We compare the presented approach to the baseline [156] on three tracks, at different speeds and track perturbations. (a)-(c): Perturbed tracks and example trajectories flown by our approach. (d)-(f): Success rate of our method. For each data point, 5 experiments were performed with random initial gate perturbation. (g)-(i): Success rate of the baseline method.

in all scenarios. This is mainly because the baseline relies on the permanent visibility of the next gate. Therefore, it only manages to complete a lap in the simplest first track where the next gate can always be seen. In the more complex second and third tracks, the baseline passes at most one or two gates. In contrast, due to the integration of prior information from demonstration and approximate mapping, our approach is successful on all tracks, including the very challenging third one.

### B.5.2 Physical System

To show the capabilities of our approach on a physical platform, we evaluated it on a real-world track with 8 gates and a total length of 80 meters, shown in Fig. B.7. No training data for the perception system was collected in this environment. Fig. B.8 summarizes the results. As in the simulation experiments, we measure the performance with respect to the average MAV speed. As before, a success rate of 100% requires 3



**Figure B.7** – Trajectory flown through multiple gate, one of which was moved as indicated by the red arrow. For visualization, only a single lap is illustrated.

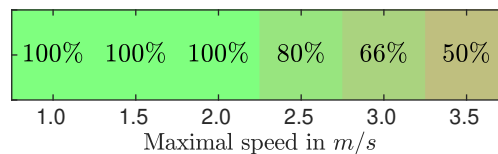
completed laps without crashing or missing a gate. Our approach confidently completed 3 laps with speeds up to  $2 \text{ m s}^{-1}$  and managed to complete the track with speeds up to  $3.5 \text{ m s}^{-1}$ . In contrast, the reactive baseline was not able to complete the full track even at  $1.0 \text{ m s}^{-1}$  (not shown in the figure).

An example recorded trajectory of our approach is shown in Fig. B.7. Note that one of the gates was moved during the experiment, but our approach was robust to this change in the environment. Our approach could handle gate displacements of up to 3.0 m and complete the full track without crashing. The reader is encouraged to watch the supplementary video for more qualitative results on real tracks.

## B.6 Conclusion

We presented an approach to autonomous vision-based drone navigation. The approach combines learning methods and optimal filtering. In addition to predicting relative gate poses, our network also estimates the uncertainty of its predictions. This allows us to integrate the network outputs with prior information via an extended Kalman filter.

We showed successful navigation through both simulated and real-world race tracks with



**Figure B.8** – Success rates of our approach in the real-world experiment. The reader is encouraged to watch the supplementary video to see the presented approach in action.

## Appendix B. Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing

---

increased robustness and speed compared to a state-of-the-art baseline. The presented approach reliably handles gate displacements of up to 2 m. In the physical track, we reached speeds of up to  $3.5 \text{ m s}^{-1}$ , outpacing the baseline by a large margin.

Our approach is capable of flying a new track with an approximate map obtained from a single demonstration flight. This approach was used to win the IROS 2018 Autonomous Drone Race Competition, where it outraced the second-placing entry by a factor of two.

# C AlphaPilot: Autonomous Drone Racing

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

Philipp Foehn\*, Dario Brescianini\*, Elia Kaufmann\*, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. “AlphaPilot: Autonomous Drone Racing”. In: *Robotics: Science and Systems (RSS)* (2020)

# AlphaPilot: Autonomous Drone Racing

Philipp Foehn\*, Dario Brescianini\*, Elia Kaufmann\*,

Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, Davide Scaramuzza

**Abstract** — This paper presents a novel system for autonomous, vision-based drone racing combining learned data abstraction, non-linear filtering, and time-optimal trajectory planning. The system has successfully been deployed at the first autonomous drone racing world championship: the *2019 AlphaPilot Challenge*. Contrary to traditional drone racing systems, which only detect the next gate, our approach makes use of any visible gate and takes advantage of multiple, simultaneous gate detections to compensate for drift in the state estimate and build a global map of the gates. The global map and drift-compensated state estimate allow the drone to navigate through the race course even when the gates are not immediately visible and further enable to plan a near time-optimal path through the race course in real time based on approximate drone dynamics. The proposed system has been demonstrated to successfully guide the drone through tight race courses reaching speeds up to 8m/s and ranked second at the *2019 AlphaPilot Challenge*.

## C.1 Introduction

### C.1.1 Motivation

Autonomous drones have seen a massive gain in robustness in recent years and perform an increasingly large set of tasks across various commercial industries; however, they are still far from fully exploiting their physical capabilities. Indeed, most autonomous drones only fly at low speeds near hover conditions in order to be able to robustly sense their environment and to have sufficient time to avoid obstacles. Faster and more agile flight could not only increase the flight range of autonomous drones, but also improve their ability to avoid fast dynamic obstacles and enhance their maneuverability in confined spaces. Human pilots have shown that drones are capable of flying through complex environments, such as race courses, at breathtaking speeds. However, autonomous drones are still far from human performance in terms of speed, versatility, and robustness, so that a lot of research and innovation is needed in order to fill this gap.

In order to push the capabilities and performance of autonomous drones, in 2019, Lockheed Martin and the Drone Racing League have launched the *AlphaPilot Challenge*<sup>1,2</sup>, an open innovation challenge with a grand prize of \$1 million. The goal of the challenge is to develop a fully autonomous drone that navigates through a race course using machine vision, and which could one day beat the best human pilot. While other autonomous drone races [237, 235] focus on complex navigation, the *AlphaPilot Challenge* pushes the limits in terms of speed and course size to advance the state of the art and enter the domain of human performance. Due to the high speeds at which drones must fly in order to beat the best human pilots, the challenging visual environments (e.g., low light, motion blur), and the limited computational power of drones, autonomous drone racing raises fundamental challenges in real-time state estimation, perception, planning, and control.

### C.1.2 Related Work

Autonomous navigation in indoor or GPS-denied environments typically relies on simultaneous localization and mapping (SLAM), often in the form of visual-inertial odometry (VIO) [41]. There exists a variety of VIO algorithms, e.g., [241, 32, 278, 97], that are based on feature detection and tracking that achieve very good results in general navigation tasks [64]. However, the performance of these algorithms significantly degrades during agile and high-speed flight as encountered in drone racing. The drone's high translational and rotational velocities cause large optic flow, making robust feature detection and tracking over sequential images difficult and thus causing substantial drift in the VIO state estimate [63].

To overcome this difficulty, several approaches exploiting the structure of drone racing with gates as landmarks have been developed, e.g., [196, 156, 170], where the drone locates itself relative to gates. In [196], a handcrafted process is used to extract gate information from images that is then fused with attitude estimates from an inertial measurement unit

---

<sup>1</sup><https://thedroneracingleague.com/airr/>

<sup>2</sup><https://www.nytimes.com/2019/03/26/technology/alphapilot-ai-drone-racing.html>



**Figure C.1** – Our *AlphaPilot* drone waiting on the start podium to autonomously race through the gates ahead.

(IMU) to compute an attitude reference that guides the drone towards the 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. In [156], a convolutional neural network (CNN) is used to 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. While this approach is successfully deployed on a real robotic system, the generated control commands do not account for the underactuated system dynamics of the quadrotor, constraining this method to low-speed flight. The approach presented in [170] also relies on relative gate data but has the advantage that it works even when no gate is visible. In particular, it uses a CNN to directly infer relative gate poses from images and fuse the results with a VIO state estimate. However, the CNN does not perform well when multiple gates are visible as it is frequently the case for drone racing.

Assuming knowledge of the platform state and the environment, there exist many approaches which can reliably generate feasible trajectories with high efficiency. The most prominent line of work exploits the quadrotor’s underactuated nature and the resulting differentially-flat output states [228, 243], where trajectories are described as polynomials in time. Other approaches additionally incorporate obstacle avoidance [372, 107] or perception constraints [79, 323]. However, in the context of drone racing, specifically the AlphaPilot Challenge, obstacle avoidance is often not needed, but time-optimal planning



is of interest. There exists a handful of approaches for time-optimal planning [127, 205, 296, 91]. However, while [127, 205] are limited to 2D scenarios and only find trajectories between two given states, [296] requires simulation and real-world data obtained on the track, and the method of [91] is not applicable due to computational constraints.

### C.1.3 Contribution

The approach contributed herein builds upon the work of [170] and fuses VIO with a robust CNN-based gate corner detection using an extended Kalman filter (EKF), achieving high accuracy at little computational cost. The gate corner detections are used as static features to compensate for the VIO drift and to align the drone’s flight path precisely with the gates. Contrary to all previous works [196, 156, 170], which only detect the next gate, our approach makes use of any gate detection and even profits from multiple simultaneous detections to compensate for VIO drift and build a global gate map. The global map allows the drone to navigate through the race course even when the gates are not immediately visible and further enables the usage of sophisticated path planning and control algorithms. In particular, a computationally efficient, sampling-based path planner (see e.g., [190], and references therein) is employed that plans near time-optimal paths through multiple gates ahead and is capable of adjusting the path in real time if the global map is updated.

This paper extends our previous work [92] by including a more detailed elaboration on our gate corner detection in Sec. C.4 with an ablation study in Sec. C.8.1, further details on the fusion of VIO and gate detection in Sec. C.5, and a description of the path parameterization in Sec. C.6, completed by an ablation study on the planning horizon length in Sec. C.8.3.

## C.2 AlphaPilot Race Format and Drone

### C.2.1 Race Format

From more than 400 teams that participated in a series of qualification tests including a simulated drone race [117], the top nine teams were selected to compete in the *2019 AlphaPilot Challenge*. The challenge consists of three qualification races and a final championship race at which the six best teams from the qualification races compete for the grand prize of \$1 million. Each race is implemented as a time trial competition in which each team is given three attempts to fly through a race course as fast a possible without competing drones on the course. Taking off from a start podium, the drones have to autonomously navigate through a sequence of gates with distinct appearances in the correct order and terminate at a designated finish gate. The race course layout, gate sequence, and position are provided ahead of each race up to approximately  $\pm 3$  m horizontal uncertainty, enforcing teams to come up with solutions that adapt to the real gate positions. Initially, the race courses were planned to have a lap length of approximately 300 m and required the completion up to three laps. However, due to technical difficulties, no race required to complete multiple laps and the track length at

the final championship race was limited to about 74 m.

### C.2.2 Drone Specifications

All teams were provided with an identical race drone (Fig. C.1) that was approximately 0.7 m in diameter, weighed 3.4 kg, and had a thrust-to-weight ratio of 1.4. The drone was equipped with a NVIDIA Jetson Xavier embedded computer for interfacing all sensors and actuators and handling all computation for autonomous navigation onboard. The sensor suite included two  $\pm 30^\circ$  forward-facing stereo camera pairs (Fig. C.2), an IMU, and a downward-facing laser rangefinder (LRF). All sensor data were globally time stamped by software upon reception at the onboard computer. Detailed specifications of the available sensors are given in Table C.1. The drone was equipped with a flight controller that controlled the total thrust  $f$  along the drone’s  $z$ -axis (see Fig. C.2) and the angular velocity,  $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$ , in the body-fixed coordinate frame  $\mathcal{B}$ .

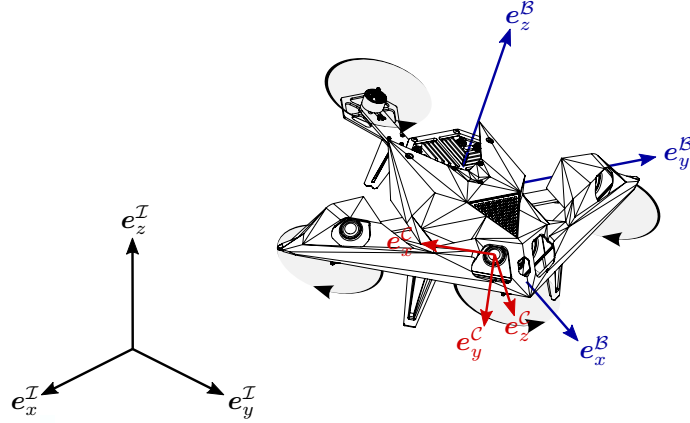
### C.2.3 Drone Model

Bold lower case and upper case letters will be used to denote vectors and matrices, respectively. The subscripts in  ${}^{\mathcal{I}}\mathbf{p}_{CB} = {}^{\mathcal{I}}\mathbf{p}_B - {}^{\mathcal{I}}\mathbf{p}_C$  are used to express a vector from point  $C$  to point  $B$  expressed in frame  $\mathcal{I}$ . Without loss of generality,  $I$  is used to represent the origin of frame  $\mathcal{I}$ , and  $B$  represents the origin of coordinate frame  $\mathcal{B}$ . For the sake of readability, the leading subscript may be omitted if the frame in which the vector is expressed is clear from context.

The drone is modelled as a rigid body of mass  $m$  with rotor drag proportional to its velocity acting on it [160]. The translational degrees-of-freedom are described by the position of its center of mass,  $\mathbf{p}_B = (p_{B,x}, p_{B,y}, p_{B,z})$ , with respect to an inertial frame  $\mathcal{I}$  as illustrated in Fig. C.2. The rotational degrees-of-freedom are parametrized using a unit quaternion,  $\mathbf{q}_{\mathcal{I}\mathcal{B}}$ , where  $\mathbf{R}_{\mathcal{I}\mathcal{B}} = \mathbf{R}(\mathbf{q}_{\mathcal{I}\mathcal{B}})$  denotes the rotation matrix mapping a vector from the body-fixed coordinate frame  $\mathcal{B}$  to the inertial frame  $\mathcal{I}$  [313]. A unit quaternion,  $\mathbf{q}$ , consists of a scalar  $q_w$  and a vector  $\tilde{\mathbf{q}} = (q_x, q_y, q_z)$  and is defined as  $\mathbf{q} = (q_w, \tilde{\mathbf{q}})$  [313].

Table C.1 – Sensor specifications.

Sensor	Model	Rate	Details
Cam	Leopard Imaging IMX 264	60 Hz	global shutter, color resolution: $1200 \times 720$
IMU	Bosch BMI088	430 Hz	range: $\pm 24g, \pm 34.5 \text{ rad/s}$ resolution: $7e^{-4}g, 1e^{-3} \text{ rad/s}$
LRF	Garmin LIDAR-Lite v3	120 Hz	range: 1-40 m resolution: 0.01 m



**Figure C.2** – Illustration of the race drone with its body-fixed coordinate frame  $\mathcal{B}$  in blue and a camera coordinate frame  $\mathcal{C}$  in red.

The drone's equations of motion are

$$m\ddot{\mathbf{p}}_B = \mathbf{R}_{IB}f\mathbf{e}_z^{\mathcal{B}} - \mathbf{R}_{IB}\mathbf{D}\mathbf{R}_{IB}^T\mathbf{v}_B - m\mathbf{g}, \quad (\text{C.1})$$

$$\dot{\mathbf{q}}_{IB} = \frac{1}{2} \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \otimes \mathbf{q}_{IB}, \quad (\text{C.2})$$

where  $f$  and  $\boldsymbol{\omega}$  are the force and bodyrate inputs,  $\mathbf{e}_z^{\mathcal{B}} = (0, 0, 1)$  is the drone's  $z$ -axis expressed in its body-fixed frame  $\mathcal{B}$ ,  $\mathbf{D} = \text{diag}(d_x, d_y, 0)$  is a constant diagonal matrix containing the rotor drag coefficients,  $\mathbf{v}_B = \dot{\mathbf{p}}_B$  denotes the drone's velocity,  $\mathbf{g}$  is gravity and  $\otimes$  denotes the quaternion multiplication operator [313]. The drag coefficients were identified experimentally to be  $d_x = 0.5 \text{ kg/s}$  and  $d_y = 0.25 \text{ kg/s}$ .

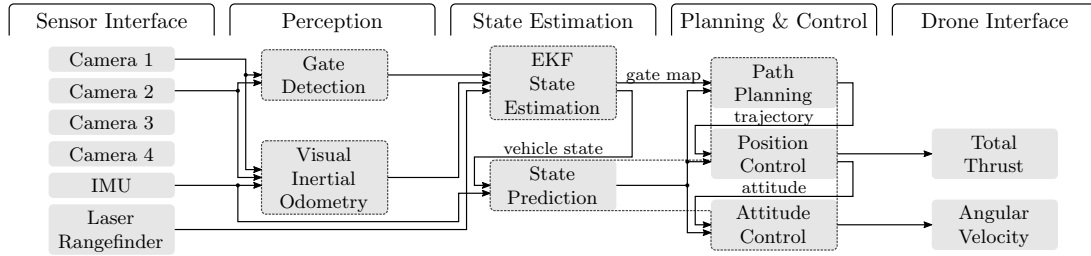
## C.3 System Overview

The system is composed of five functional groups: Sensor interface, perception, state estimation, planning and control, and drone interface (see Fig. C.3). In the following, a brief introduction to the functionality of our proposed perception, state estimation, and planning and control system is given.

### C.3.1 Perception

Of the two stereo camera pairs available on the drone, only the two central forward-facing cameras are used for gate detection (see Section C.4) and, in combination with IMU measurements, to run VIO. The advantage is that the amount of image data to be processed is reduced while maintaining a very large field of view. Due to its robustness, multi-camera capability and computational efficiency, ROVIO [32] has been chosen as VIO pipeline. At low speeds, ROVIO is able to provide an accurate estimate of the quadrotor vehicle's pose and velocity relative to its starting position, however, at larger speeds the state estimate suffers from drift.

## Appendix C. AlphaPilot: Autonomous Drone Racing



**Figure C.3** – Overview of the system architecture and its main components. All components within a dotted area run in a single thread.

### C.3.2 State Estimation

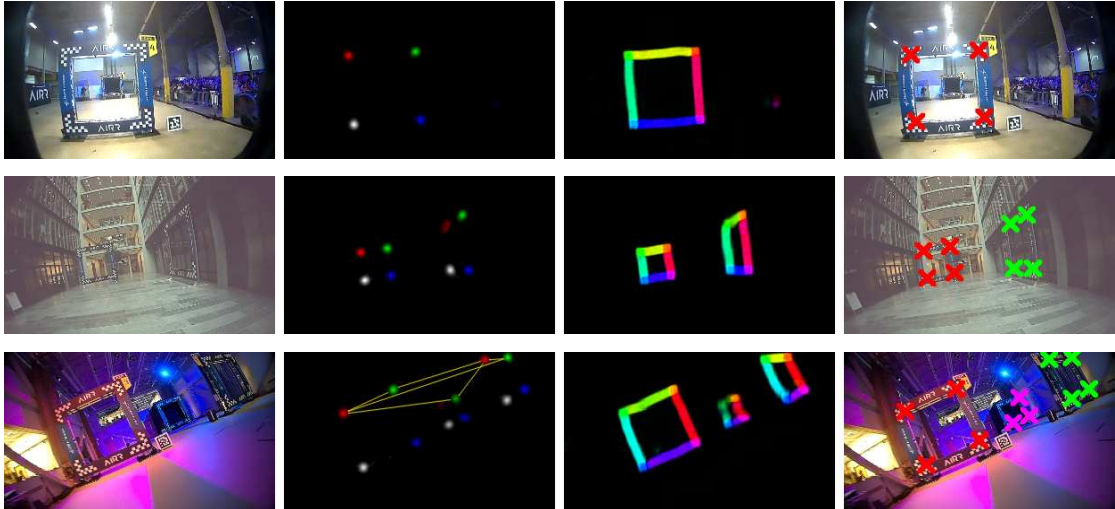
In order to compensate for a drifting VIO estimate, the output of the gate detection and VIO are fused together with the measurements from the downward-facing laser rangefinder (LRF) using an EKF (see Section C.5). The EKF estimates a global map of the gates and, since the gates are stationary, uses the gate detections to align the VIO estimate with the global gate map, i.e., compensates for the VIO drift. Computing the state estimate, in particular interfacing the cameras and running VIO, introduces latency in the order of 130 ms to the system. In order to be able to achieve a high bandwidth of the control system despite large latencies, the vehicle’s state estimate is predicted forward to the vehicle’s current time using the IMU measurements.

### C.3.3 Planning and Control

The global gate map and the latency-compensated state estimate of the vehicle are used to plan a near time-optimal path through the next  $N$  gates starting from the vehicle’s current state (see Section C.6). The path is re-planned every time (i) the vehicle passes through a gate, (ii) the estimate of the gate map or (iii) the VIO drift are updated significantly, i.e., large changes in the gate positions or VIO drift. The path is tracked using a cascaded control scheme (see Section C.7) with an outer proportional-derivative (PD) position control loop and an inner proportional (P) attitude control loop. Finally, the outputs of the control loops, i.e., a total thrust and angular velocity command, are sent to the drone.

### C.3.4 Software Architecture

The NVIDIA Jetson Xavier provides eight CPU cores, however, four cores are used to run the sensor and drone interface. The other four cores are used to run the gate detection, VIO, EKF state estimation, and planning and control, each in a separate thread on a separate core. All threads are implemented asynchronously to run at their own speed, i.e., whenever new data is available, in order to maximize data throughput and to reduce processing latency. The gate detection thread is able to process all camera images in real time at 60 Hz, whereas the VIO thread only achieves approximately 35 Hz. In order



**Figure C.4** – The gate detection module returns sets of corner points for each gate in the input image (fourth column) using a two-stage process. In the first stage, a neural network transforms an input image,  $I_{w \times h \times 3}$  (first column), into a set of confidence maps for corners,  $C_{w \times h \times 4}$  (second column), and Part Affinity Fields (PAFs) [43],  $E_{w \times h \times (4 \cdot 2)}$  (third column). In the second stage, the PAFs are used to associate sets of corner points that belong to the same gate. For visualization, both corner maps,  $C$  (second column), and PAFs,  $E$  (third column), are displayed in a single image each. While color encodes the corner class for  $C$ , it encodes the direction of the 2D vector fields for  $E$ . The yellow lines in the bottom of the second column show the six edge candidates of the edge class ( $TL, TR$ ) (the  $TL$  corner of the middle gate is below the detection threshold), see Section C.4.2. Best viewed in color.

to deal with the asynchronous nature of the gate detection and VIO thread and their output, all data is globally time stamped and integrated in the EKF accordingly. The EKF thread runs every time a new gate or LRF measurement is available. The planning and control thread runs at a fixed rate of 50 Hz. To achieve this, the planning and control thread includes the state prediction which compensates for latencies introduced by the VIO.

## C.4 Gate Detection

To correct for drift accumulated by the VIO pipeline, the gates are used as distinct landmarks for relative localization. In contrast to previous CNN-based approaches to gate detection, we do not infer the relative pose to a gate directly, but instead segment the four corners of the observed gate in the input image. These corner segmentations represent the likelihood of a specific gate corner to be present at a specific pixel coordinate. To represent a value proportional to the likelihood, the maps are trained on Gaussians of the corner projections. This allows the detection of an arbitrary amount of gates, and allows for a more principled inclusion of gate measurements in the EKF through the use of reprojection error. Specifically, it exhibits more predictable behavior for partial gate observations and overlapping gates, and allows to suppress the impact of Gaussian noise by having multiple measurements relating to the same quantity. Since the exact shape of the gates is known, detecting a set of characteristic points per gate allows to

## Appendix C. AlphaPilot: Autonomous Drone Racing

---

constrain the relative pose. For the quadratic gates of the *AlphaPilot Challenge*, these characteristic points are chosen to be the inner corner of the gate border (see Fig. C.4, 4th column). However, just detecting the four corners of all gates is not enough. If just four corners of several gates are extracted, the association of corners to gates is undefined (see Fig. C.4, 3rd row, 2nd column). To solve this problem, we additionally train our network to extract so-called Part Affinity Fields (PAFs), as proposed by [43]. These are vector fields, which, in our case, are defined along the edges of the gates, and point from one corner to the next corner of the same gate, see column three in Figure C.4. The entire gate detection pipeline consists of two stages: 1) predicting corner maps and PAFs by the neural network, 2) extracting single edge candidates from the network prediction and assembling them to gates. In the following, both stages are explained in detail.

### C.4.1 Stage 1: Predicting Corner Maps and Part Affinity Fields

In the first detection stage, each input image,  $\mathbf{I}_{w \times h \times 3}$ , is mapped by a neural network into a set of  $N_C = 4$  corner maps,  $\mathbf{C}_{w \times h \times N_C}$ , and  $N_E = 4$  PAFs,  $\mathbf{E}_{w \times h \times (N_E \cdot 2)}$ . Predicted corner maps as well as PAFs are illustrated in Figure C.4, 2nd and 3rd column. The network is trained in a supervised manner by minimizing the Mean-Squared-Error loss between the network prediction and the ground-truth maps. In the following, ground-truth maps for both map types are explained in detail.

#### Corner Maps

For each corner class,  $j \in \mathcal{C}_j$ ,  $\mathcal{C}_j := \{TL, TR, BL, BR\}$ , a ground-truth corner map,  $\mathbf{C}_j^*(\mathbf{s})$ , is represented by a single-channel map of the same size as the input image and indicates the existence of a corner of class  $j$  at pixel location  $\mathbf{s}$  in the image. The value at location  $\mathbf{s} \in \mathbf{I}$  in  $\mathbf{C}_j^*$  is defined by a Gaussian as

$$\mathbf{C}_j^*(\mathbf{s}) = \exp\left(-\frac{\|\mathbf{s} - \mathbf{s}_j^*\|_2^2}{\sigma^2}\right), \quad (\text{C.3})$$

where  $\mathbf{s}_j^*$  denotes the ground truth image position of the nearest corner with class  $j$ . The choice of the parameter  $\sigma$  controls the width of the Gaussian. We use  $\sigma = 7$  pixel in our implementation. Gaussians are used to account for small errors in the ground-truth corner positions that are provided by hand. Ground-truth corner maps are generated for each individual gate in the image separately and then aggregated. Aggregation is performed by taking the pixel-wise maximum of the individual corner maps, as this preserves the distinction between close corners.

#### Part Affinity Fields

We define a PAF for each of the four possible classes of edges, defined by its two connecting corners as  $(k, l) \in \mathcal{E}_{KL} := \{(TL, TR), (TR, BR), (BR, BL), (BL, TL)\}$ . For each edge class,  $(k, l)$ , the ground-truth PAF,  $\mathbf{E}_{(k,l)}^*(\mathbf{s})$ , is represented by a two-channel map of the same

size as the input image and points from corner  $k$  to corner  $l$  of the same gate, provided that the given image point  $\mathbf{s}$  lies within distance  $d$  of such an edge. We use  $d = 10$  pixel in our implementation. Let  $\mathcal{G}^*$  be the set of gates  $g$  and  $\mathcal{S}_{(k,l),g}$  be the set of image points that are within distance  $d$  of the line connecting the corner points  $\mathbf{s}_k^*$  and  $\mathbf{s}_l^*$  belonging to gate  $g$ . Furthermore, let  $\mathbf{v}_{k,l,g}$  be the unit vector pointing from  $\mathbf{s}_k^*$  to  $\mathbf{s}_l^*$  of the same gate. Then, the part affinity field,  $\mathbf{E}_{(k,l)}^*(\mathbf{s})$ , is defined as:

$$\mathbf{E}_{(k,l)}^*(\mathbf{s}) = \begin{cases} \mathbf{v}_{k,l,g} & \text{if } \mathbf{s} \in \mathcal{S}_{(k,l),g}, \quad g \in \mathcal{G}^* \\ \mathbf{0} & \text{otherwise.} \end{cases} \quad (\text{C.4})$$

As in the case of corner maps, PAFs are generated for each individual gate in the image separately and then aggregated. In case a point  $\mathbf{s}$  lies in  $\mathcal{S}_{(k,l),g}$  of several gates, the  $\mathbf{v}_{k,l,g}$  of all corresponding gates are averaged.

### C.4.2 Stage 2: Corner Association

At test time, discrete corner candidates,  $\mathbf{s}_j$ , for each corner class,  $j \in \mathcal{C}_j$ , are extracted from the predicted corner map using non-maximum suppression and thresholding. For each corner class, there might be several corner candidates, due to multiple gates in the image or false positives. These corner candidates allow the formation of an exhaustive set of edge candidates,  $\{(\mathbf{s}_k, \mathbf{s}_l)\}$ , see the yellow lines in Fig. C.4. Given the corresponding PAF,  $\mathbf{E}_{(k,l)}(\mathbf{s})$ , each edge candidate is assigned a score which expresses the agreement of that candidate with the PAF. This score is given by the line integral

$$\mathcal{S}((\mathbf{s}_k, \mathbf{s}_l)) = \int_{u=0}^{u=1} \mathbf{E}_{(k,l)}(\mathbf{s}(u)) \cdot \frac{\mathbf{s}_l - \mathbf{s}_k}{\|\mathbf{s}_l - \mathbf{s}_k\|} du, \quad (\text{C.5})$$

where  $\mathbf{s}(u)$  linearly interpolates between the two corner candidate locations  $\mathbf{s}_k$  and  $\mathbf{s}_l$ . In practice,  $\mathcal{S}$  is approximated by uniformly sampling the integrand.

The line integral  $\mathcal{S}$  is used as metric to associate corner candidates to gate detections. The goal is to find the optimal assignment for the set of all possible corner candidates to gates. As described in [43], finding this optimal assignment corresponds to a  $K$ -dimensional matching problem that is known to be NP-Hard [356]. Following [43], the problem is simplified by decomposing the matching problem into a set of bipartite matching subproblems. Matching is therefore performed independently for each edge class. Specifically, the following optimization problem represents the bipartite matching subproblem for edge class  $(k, l)$ :

$$\max \mathcal{S}_{(k,l)} = \sum_{k \in \mathcal{D}_k} \sum_{l \in \mathcal{D}_l} \mathcal{S}((\mathbf{s}_k, \mathbf{s}_l)) \cdot z_{kl} \quad (\text{C.6})$$

$$\text{s.t. } \forall k \in \mathcal{D}_k, \sum_{l \in \mathcal{D}_l} z_{kl} \leq 1, \quad (\text{C.7})$$

$$\forall l \in \mathcal{D}_l, \sum_{k \in \mathcal{D}_k} z_{kl} \leq 1, \quad (\text{C.8})$$

## Appendix C. AlphaPilot: Autonomous Drone Racing

---

where  $\mathcal{S}_{(k,l)}$  is the cumulative matching score and  $\mathcal{D}_k, \mathcal{D}_l$  denote the set of corner candidates for edge class  $(k, l)$ . The variable  $z_{kl} \in \{0, 1\}$  indicates whether two corner candidates are connected. Equations (C.7) and (C.8) enforce that no two edges share the same corner. Above optimization problem can be solved using the Hungarian method [185], resulting in a set of edge candidates for each edge class  $(k, l)$ .

With the bipartite matching problems being solved for all edge classes, the pairwise associations can be extended to sets of associated edges for each gate.

### C.4.3 Training Data

The neural network is trained in a supervised fashion using a dataset recorded in the real world. Training data is generated by recording video sequences of gates in 5 different environments. Each frame is annotated with the corners of all gates visible in the image using the open source image annotation software labelme<sup>3</sup>, which is extended with KLT-Tracking for semi-automatic labelling. The resulting dataset used for training consists of 28k images and is split into 24k samples for training and 4k samples for validation. At training time, the data is augmented using random rotations of up to 30° and random changes in brightness, hue, contrast and saturation.

### C.4.4 Network Architecture and Deployment

The network architecture is designed to optimally trade-off between computation time and accuracy. By conducting a neural network architecture search, the best performing architecture for the task is identified. The architecture search is limited to variants of U-Net [287] due to its ability to perform segmentation tasks efficiently with a very limited amount of labeled training data. The best performing architecture is identified as a 5-level U-Net with [12, 18, 24, 32, 32] convolutional filters of size [3, 3, 3, 5, 7] per level and a final additional layer operating on the output of the U-Net containing 12 filters. At each layer, the input feature map is zero-padded to preserve a constant height and width throughout the network. As activation function, LeakyReLU with  $\alpha = 0.01$  is used. For deployment on the Jetson Xavier, the network is ported to TensorRT 5.0.2.6. To optimize memory footprint and inference time, inference is performed in half-precision mode (FP16) and batches of two images of size  $592 \times 352$  are fed to the network.

## C.5 State Estimation

The non-linear measurement models of the VIO, gate detection, and laser rangefinder are fused using an EKF [161]. In order to obtain the best possible pose accuracy relative to the gates, the EKF estimates the translational and rotational misalignment of the VIO origin frame,  $\mathcal{V}$ , with respect to the inertial frame,  $\mathcal{I}$ , represented by  $\mathbf{p}_V$  and  $\mathbf{q}_{\mathcal{I}\mathcal{V}}$ , jointly with the gate positions,  $\mathbf{p}_{G_i}$ , and gate heading,  $\varphi_{\mathcal{I}G_i}$ . It can thus correct for an imprecise initial position estimate, VIO drift, and uncertainty in gate positions. The EKF's state

---

<sup>3</sup><https://github.com/wkentaro/labelme>



space at time  $t_k$  is  $\mathbf{x}_k = \mathbf{x}(t_k)$  with covariance  $\mathbf{P}_k$  described by

$$\mathbf{x}_k = (\mathbf{p}_{\mathcal{V}}, \mathbf{q}_{\mathcal{I}\mathcal{V}}, \mathbf{p}_{G_0}, \varphi_{\mathcal{I}G_0}, \dots, \mathbf{p}_{G_{N-1}}, \varphi_{\mathcal{I}G_{N-1}}). \quad (\text{C.9})$$

The drone's corrected pose,  $(\mathbf{p}_B, \mathbf{q}_{\mathcal{I}B})$ , can then be computed from the VIO estimate,  $(\mathbf{p}_{\mathcal{V}B}, \mathbf{q}_{\mathcal{V}B})$ , by transforming it from frame  $\mathcal{V}$  into the frame  $\mathcal{I}$  using  $(\mathbf{p}_{\mathcal{V}}, \mathbf{q}_{\mathcal{I}\mathcal{V}})$  as

$$\mathbf{p}_B = \mathbf{p}_{\mathcal{V}} + \mathbf{R}_{\mathcal{I}\mathcal{V}} \cdot \mathbf{p}_{\mathcal{V}B}, \quad \mathbf{q}_{\mathcal{I}B} = \mathbf{q}_{\mathcal{I}\mathcal{V}} \cdot \mathbf{q}_{\mathcal{V}B}. \quad (\text{C.10})$$

All estimated parameters are expected to be time-invariant but subject to noise and drift. This is modelled by a Gaussian random walk, simplifying the EKF process update to:

$$\mathbf{x}_{k+1} = \mathbf{x}_k, \quad \mathbf{P}_{k+1} = \mathbf{P}_k + \Delta t_k \mathbf{Q}, \quad (\text{C.11})$$

where  $\mathbf{Q}$  is the random walk process noise. For each measurement  $\mathbf{z}_k$  with noise  $\mathbf{R}$  the predicted *a priori* estimate,  $\mathbf{x}_k^-$ , is corrected with measurement function,  $\mathbf{h}(\mathbf{x}_k^-)$ , and Kalman gain,  $\mathbf{K}_k$ , resulting in the *a posteriori* estimate,  $\mathbf{x}_k^+$ , as

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_k^- \mathbf{H}_k^\top (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^\top + \mathbf{R})^{-1}, \\ \mathbf{x}_k^+ &= \mathbf{x}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{h}(\mathbf{x}_k^-)), \\ \mathbf{P}_k^+ &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^-, \end{aligned} \quad (\text{C.12})$$

with  $\mathbf{h}(\mathbf{x}_k^-)$ , the measurement function with Jacobian  $\mathbf{H}_k$ .

However, the filter state includes a rotation quaternion constrained to unit norm,  $\|\mathbf{q}_{\mathcal{I}\mathcal{V}}\| \stackrel{!}{=} 1$ . This is effectively an over-parameterization in the filter state space and can lead to poor linearization as well as underestimation of the covariance. To apply the EKFs linear update step on the over-parameterized quaternion, it is lifted to its tangent space description, similar to [95]. The quaternion  $\mathbf{q}_{\mathcal{I}\mathcal{V}}$  is composed of a reference quaternion,  $\mathbf{q}_{\mathcal{I}\mathcal{V}_{\text{ref}}}$ , which is adjusted after each update step, and an error quaternion,  $\mathbf{q}_{\mathcal{V}_{\text{ref}}\mathcal{V}}$ , of which only its vector part,  $\tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}$ , is in the EKF's state space. Therefore we get

$$\mathbf{q}_{\mathcal{I}\mathcal{V}} = \mathbf{q}_{\mathcal{I}\mathcal{V}_{\text{ref}}} \cdot \mathbf{q}_{\mathcal{V}_{\text{ref}}\mathcal{V}} \quad \mathbf{q}_{\mathcal{V}_{\text{ref}}\mathcal{V}} = \begin{bmatrix} \sqrt{1 - \tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}^\top \cdot \tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}} \\ \tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}} \end{bmatrix} \quad (\text{C.13})$$

from which we can derive the Jacobian of any measurement function,  $\mathbf{h}(\mathbf{x})$ , with respect to  $\mathbf{q}_{\mathcal{I}\mathcal{V}}$  by the chain rule as

$$\frac{\partial}{\partial \tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}} \mathbf{h}(\mathbf{x}) = \frac{\partial}{\partial \mathbf{q}_{\mathcal{I}\mathcal{V}}} \mathbf{h}(\mathbf{x}) \cdot \frac{\partial \mathbf{q}_{\mathcal{I}\mathcal{V}}}{\partial \tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}} \quad (\text{C.14})$$

$$= \frac{\partial}{\partial \tilde{\mathbf{q}}_{\mathcal{I}\mathcal{V}}} \mathbf{h}(\mathbf{x}) \cdot [\mathbf{q}_{\mathcal{I}\mathcal{V}_{\text{ref}}}]_{\times} \begin{bmatrix} -\tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}^\top \\ \sqrt{1 - \tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}^\top \cdot \tilde{\mathbf{q}}_{\mathcal{V}_{\text{ref}}\mathcal{V}}} \\ \mathbf{I}^{3 \times 3} \end{bmatrix} \quad (\text{C.15})$$

where we arrive at (C.15) by using (C.13) in (C.14) and use  $[\mathbf{q}_{\mathcal{I}\mathcal{V}_{\text{ref}}}]_{\times}$  to represent the matrix resulting from a lefthand-side multiplication with  $\mathbf{q}_{\mathcal{I}\mathcal{V}_{\text{ref}}}$ .

### C.5.1 Measurement Modalities

All measurements up to the camera frame time  $t_k$  are passed to the EKF together with the VIO estimate,  $\mathbf{p}_{VB,k}$  and  $\mathbf{q}_{VB,k}$ , with respect to the VIO frame  $\mathcal{V}$ . Note that the VIO estimate is assumed to be a constant parameter, not a filter state, which vastly simplifies derivations and computation, leading to an efficient yet robust filter.

#### Gate Measurements

Gate measurements consist of the image pixel coordinates,  $\mathbf{s}_{Co_{ij}}$ , of a specific gate corner. These corners are denoted with top left and right, and bottom left and right, as in  $j \in \mathcal{C}_j$ ,  $\mathcal{C}_j := \{TL, TR, BL, BR\}$  and the gates are enumerated by  $i \in [0, N - 1]$ . All gates are of equal width,  $w$ , and height,  $h$ , so that the corner positions in the gate frame,  $\mathcal{G}_i$ , can be written as  $\mathbf{p}_{G_i Co_{ij}} = \frac{1}{2}(0, \pm w, \pm h)$ . The measurement equation can be written as the pinhole camera projection [334] of the gate corner into the camera frame. A pinhole camera maps the gate corner point,  $\mathbf{p}_{Co_{ij}}$ , expressed in the camera frame,  $\mathcal{C}$ , into pixel coordinates as

$$\mathbf{h}_{\text{Gate}}(\mathbf{x}) = \mathbf{s}_{Co_{ij}} = \frac{1}{[\mathbf{p}_{Co_{ij}}]_z} \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \end{bmatrix} \mathbf{p}_{Co_{ij}}, \quad (\text{C.16})$$

where  $[\cdot]_z$  indicates the scalar  $z$ -component of a vector,  $f_x$  and  $f_y$  are the camera's focal lengths and  $(c_x, c_y)$  is the camera's optical center. The gate corner point,  $\mathbf{p}_{Co_{ij}}$ , is given by

$$\mathbf{p}_{Co_{ij}} = \mathbf{R}_{IC}^T (\mathbf{p}_{G_i} + \mathbf{R}_{IG_i} \mathbf{p}_{G_i Co_{ij}} - \mathbf{p}_C), \quad (\text{C.17})$$

with  $\mathbf{p}_C$  and  $\mathbf{R}_{IC}$  being the transformation between the inertial frame  $\mathcal{I}$  and camera frame  $\mathcal{C}$ ,

$$\mathbf{p}_C = \mathbf{p}_V + \mathbf{R}_{IV} (\mathbf{p}_{VB} + \mathbf{R}_{VB} \mathbf{p}_{BC}), \quad (\text{C.18})$$

$$\mathbf{R}_{IC} = \mathbf{R}_{IV} \mathbf{R}_{VB} \mathbf{R}_{BC}, \quad (\text{C.19})$$

where  $\mathbf{p}_{BC}$  and  $\mathbf{R}_{BC}$  describe a constant transformation between the drone's body frame  $\mathcal{B}$  and camera frame  $\mathcal{C}$  (see Fig. C.2). The Jacobian with respect to the EKF's state space is derived using the chain rule,

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{h}_{\text{Gate}}(\mathbf{x}) = \frac{\partial \mathbf{h}_{\text{Gate}}(\mathbf{x})}{\partial \mathbf{p}_{Co_{ij}}(\mathbf{x})} \cdot \frac{\partial \mathbf{p}_{Co_{ij}}(\mathbf{x})}{\partial \mathbf{x}}, \quad (\text{C.20})$$

where the first term representing the derivative of the projection, and the second term represents the derivative with respect to the state space including gate position and orientation, and the frame alignment, which can be further decomposed using (C.14).

### Gate Correspondences

The gate detection (see Figure C.4) provides sets of  $m$  measurements,

$$\mathbf{S}_i = \{\mathbf{s}_{Co_{ij}0}, \dots, \mathbf{s}_{Co_{ij}m-1}\},$$

corresponding to the unknown gate  $\hat{i}$  at known corners  $j \in \mathcal{C}_j$ . To identify the correspondences between a detection set  $\mathbf{S}_i$  and the gate  $G_i$  in our map, we use the square sum of reprojection error. For this, we first compute the reprojection of all gate corners,  $\mathbf{s}_{Co_{ij}}$ , according to (C.16) and then compute the square error sum between the measurement set,  $\mathbf{S}_i$ , and the candidates,  $\mathbf{s}_{Co_{ij}}$ . Finally, the correspondence is established to the gate  $G_i$  which minimizes the square error sum, as in

$$\underset{i \in [0, N-1]}{\operatorname{argmin}} \sum_{\mathbf{s}_{Co_{ij}} \in \mathbf{S}_i} (\mathbf{s}_{Co_{ij}} - \mathbf{s}_{Co_{ij}})^T (\mathbf{s}_{Co_{ij}} - \mathbf{s}_{Co_{ij}}). \quad (\text{C.21})$$

### Laser Rangefinder Measurement

The drone's laser rangefinder measures the distance along the drones negative  $z$ -axis to the ground, which is assumed to be flat and at a height of 0 m. The measurement equation can be described by

$$h_{\text{LRF}}(\mathbf{x}) = \frac{[\mathbf{p}_B]_z}{[\mathbf{R}_{IB} \mathbf{e}_z^B]_z} = \frac{[\mathbf{p}_V + \mathbf{R}_{TV} \mathbf{p}_{VB}]_z}{[\mathbf{R}_{TV} \mathbf{R}_{VB} \mathbf{e}_z^B]_z}. \quad (\text{C.22})$$

The Jacobian with respect to the state space is again derived by  $\frac{\partial h_{\text{LRF}}}{\partial \mathbf{p}_V}$  and  $\frac{\partial h_{\text{LRF}}}{\partial \mathbf{q}_{TV}}$  and further simplified using (C.14).

## C.6 Path Planning

For the purpose of path planning, the drone is assumed to be a point mass with bounded accelerations as inputs. This simplification allows for the computation of time-optimal motion primitives in closed-form and enables the planning of approximate time-optimal paths through the race course in real time. Even though the dynamics of the quadrotor vehicle's acceleration cannot be neglected in practice, it is assumed that this simplification still captures the most relevant dynamics for path planning and that the resulting paths approximate the true time-optimal paths well. In order to facilitate the tracking of the approximate time-optimal path, polynomials of order four are fitted to the path which yield smoother position, velocity and acceleration commands, and can therefore be better tracked by the drone.

In the following, time-optimal motion primitives based on the simplified dynamics are first introduced and then a path planning strategy based on these motion primitives is presented. Finally, a method to parameterize the time-optimal path is introduced.

### C.6.1 Time-Optimal Motion Primitive

The minimum times,  $T_x^*$ ,  $T_y^*$  and  $T_z^*$ , required for the vehicle to fly from an initial state, consisting of position and velocity, to a final state while satisfying the simplified dynamics  $\ddot{\mathbf{p}}_B(t) = \mathbf{u}(t)$  with the input acceleration  $\mathbf{u}(t)$  being constrained to  $\underline{\mathbf{u}} \leq \mathbf{u}(t) \leq \bar{\mathbf{u}}$  are computed for each axis individually. Without loss of generality, only the  $x$ -axis is considered in the following. Using Pontryagin’s maximum principle [27], it can be shown that the optimal control input is bang-bang in acceleration, i.e., has the form

$$u_x^*(t) = \begin{cases} \underline{u}_x, & 0 \leq t \leq t^*, \\ \bar{u}_x, & t^* < t \leq T_x^*, \end{cases} \quad (\text{C.23})$$

or vice versa with the control input first being  $\bar{u}_x$  followed by  $\underline{u}_x$ . In order to control the maximum velocity of the vehicle, e.g., to constrain the solutions to ranges where the simplified dynamics approximate the true dynamics well or to limit the motion blur of the camera images, a velocity constraint of the form  $\underline{\mathbf{v}}_B \leq \mathbf{v}_B(t) \leq \bar{\mathbf{v}}_B$  can be added, in which case the optimal control input has bang-singular-bang solution [222]

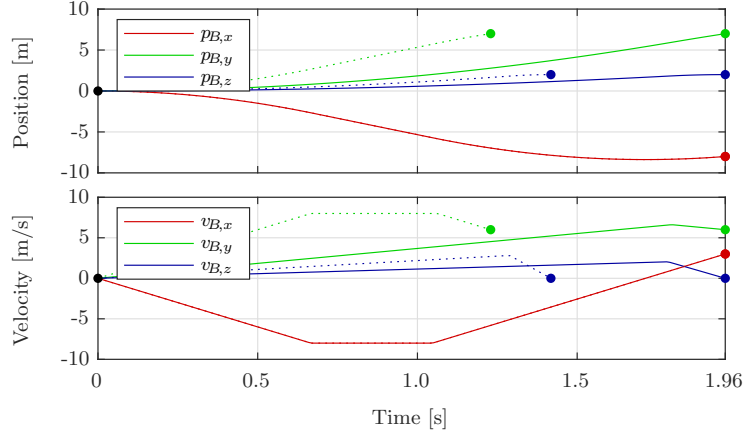
$$u_x^*(t) = \begin{cases} \underline{u}_x, & 0 \leq t \leq t_1^*, \\ 0, & t_1^* < t \leq t_2^*, \\ \bar{u}_x, & t_2^* < t \leq T_x^*, \end{cases} \quad (\text{C.24})$$

or vice versa. It is straightforward to verify that there exist closed-form solutions for the minimum time,  $T_x^*$ , as well as the switching times,  $t^*$ , in both cases (C.23) or (C.24).

Once the minimum time along each axis is computed, the maximum minimum time,  $T^* = \max(T_x^*, T_y^*, T_z^*)$ , is computed and motion primitives of the same form as in (C.23) or (C.24) are computed among the two faster axes but with the final time constrained to  $T^*$  such that trajectories along each axis end at the same time. In order for such a motion primitive to exist, a new parameter  $\alpha \in [0, 1]$  is introduced that scales the acceleration bounds, i.e., the applied control inputs are scaled to  $\alpha \underline{u}_x$  and  $\alpha \bar{u}_x$ , respectively. Fig. C.5 depicts the position and velocity of an example time-optimal motion primitive.

### C.6.2 Sampling-Based Receding Horizon Path Planning

The objective of the path planner is to find the time-optimal path from the drone’s current state to the final gate, passing through all the gates in the correct order. Since the previously introduced motion primitive allows the generation of time-optimal motions between any initial and any final state, the time-optimal path can be planned by concatenating a time-optimal motion primitive starting from the drone’s current (simplified) state to the first gate with time-optimal motion primitives that connect the gates in the correct order until the final gate. This reduces the path planning problem to finding the drone’s optimal state at each gate such that the total time is minimized. To find the optimal path, a sampling-based strategy is employed where states at each gate are randomly sampled and the total time is evaluated subsequently. In particular, the position of each sampled state at a specific gate is fixed to the center of the gate and the velocity is sampled



**Figure C.5** – Example time-optimal motion primitive starting from rest at the origin to a random final position with non-zero final velocity. The velocities are constrained to  $\pm 7.5\text{m/s}$  and the inputs to  $\pm 12\text{m/s}^2$ . The dotted lines denote the per-axis time-optimal maneuvers.

uniformly at random such the velocity lies within the constraints of the motion primitives and the angle between the velocity and the gate normal does not exceed a maximum angle,  $\varphi_{\max}$ . It is trivial to show that as the number of sampled states approaches infinity, the computed path converges to the time-optimal path.

In order to solve the problem efficiently, the path planning problem is interpreted as a shortest path problem. At each gate,  $M$  different velocities are sampled and the arc length from each sampled state at the previous gate is set to be equal to the duration,  $T^*$ , of the time-optimal motion primitive that guides the drone from one state to the other. Due to the existence of a closed-form expression for the minimum time,  $T^*$ , setting up and solving the shortest path problem can be done very efficiently using, e.g., Dijkstra’s algorithm [27] resulting in the optimal path  $\mathbf{p}^*(t)$ . In order to further reduce the computational cost, the path is planned in a receding horizon fashion, i.e., the path is only planned through the next  $N$  gates.

### C.6.3 Path Parameterization

Due to the simplifications of the dynamics that were made when computing the motion primitives, the resulting path is infeasible with respect to the quadrotor dynamics (C.1) and (C.2) and thus is impossible to be tracked accurately by the drone. To simplify the tracking of the time-optimal path, the path is approximated by fourth order polynomials in time. In particular, the path is divided into multiple segments of equal arc length. Let  $t \in [t_k, t_{k+1})$  be the time interval of the  $k$ -th segment. In order to fit the polynomials,  $\bar{\mathbf{p}}_k(t)$ , to the  $k$ -th segment of the time-optimal path, we require that the initial and final position and velocity are equal to those of the time-optimal path, i.e.,

$$\bar{\mathbf{p}}_k(t_k) = \mathbf{p}^*(t_k), \quad \bar{\mathbf{p}}_k(t_{k+1}) = \mathbf{p}^*(t_{k+1}), \quad (\text{C.25})$$

$$\dot{\bar{\mathbf{p}}}_k(t_k) = \dot{\mathbf{p}}^*(t_k), \quad \dot{\bar{\mathbf{p}}}_k(t_{k+1}) = \dot{\mathbf{p}}^*(t_{k+1}), \quad (\text{C.26})$$

## Appendix C. AlphaPilot: Autonomous Drone Racing

---

and that the positions at  $t = (t_{k+1} - t_k) / 2$  coincide as well:

$$\bar{\mathbf{p}}_k \left( \frac{t_{k+1} + t_k}{2} \right) = \mathbf{p}^* \left( \frac{t_{k+1} + t_k}{2} \right). \quad (\text{C.27})$$

The polynomial parameterization  $\bar{\mathbf{p}}_k(t)$  of the  $k$ -th segment is then given by

$$\bar{\mathbf{p}}_k(t) = \mathbf{a}_{4,k}s^4 + \mathbf{a}_{3,k}s^3 + \mathbf{a}_{2,k}s^2 + \mathbf{a}_{1,k}s + \mathbf{a}_{0,k}, \quad (\text{C.28})$$

with  $s = t - t_k$  being the relative time since the start of  $k$ -th segment. The velocity and acceleration required for the drone to track this polynomial path can be computed by taking the derivatives of (C.28), yielding

$$\dot{\bar{\mathbf{p}}}_k(t) = 4\mathbf{a}_{4,k}s^3 + 3\mathbf{a}_{3,k}s^2 + 2\mathbf{a}_{2,k}s + \mathbf{a}_{1,k}, \quad (\text{C.29})$$

$$\ddot{\bar{\mathbf{p}}}_k(t) = 12\mathbf{a}_{4,k}s^2 + 6\mathbf{a}_{3,k}s + 2\mathbf{a}_{2,k}. \quad (\text{C.30})$$

## C.7 Control

This section presents a control strategy to track the near time-optimal path from Section C.6. The control strategy is based on a cascaded control scheme with an outer position control loop and an inner attitude control loop, where the position control loop is designed under the assumption that the attitude control loop can track setpoint changes perfectly, i.e., without any dynamics or delay.

### C.7.1 Position Control

The position control loop along the inertial  $z$ -axis is designed such that it responds to position errors

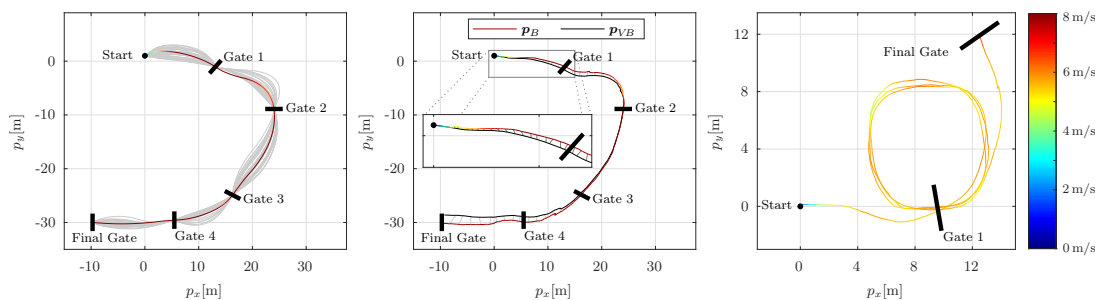
$$p_{\mathcal{B}_{\text{err}},z} = p_{\mathcal{B}_{\text{ref}},z} - p_{\mathcal{B},z}$$

in the fashion of a second-order system with time constant  $\tau_{\text{pos},z}$  and damping ratio  $\zeta_{\text{pos},z}$ ,

$$\ddot{p}_{\mathcal{B},z} = \frac{1}{\tau_{\text{pos},z}^2} p_{\mathcal{B}_{\text{err}},z} + \frac{2\zeta_{\text{pos},z}}{\tau_{\text{pos},z}} \dot{p}_{\mathcal{B}_{\text{err}},z} + \ddot{p}_{\mathcal{B}_{\text{ref}},z}. \quad (\text{C.31})$$

Similarly, two control loops along the inertial  $x$ - and  $y$ -axis are shaped to make the horizontal position errors behave like second-order systems with time constants  $\tau_{\text{pos},xy}$  and damping ratio  $\zeta_{\text{pos},xy}$ . Inserting (C.31) into the translational dynamics (C.1), the total thrust,  $f$ , is computed to be

$$f = \frac{[m(\ddot{\mathbf{p}}_{\mathcal{B}_{\text{ref}}} + \mathbf{g}) + \mathbf{R}_{IB} \mathbf{D} \mathbf{R}_{IB}^T \mathbf{v}_{\mathcal{B}}]_z}{[\mathbf{R}_{IB} \mathbf{e}_z^{\mathcal{B}}]_z}. \quad (\text{C.32})$$



**Figure C.6** – Top view of the planned (left) and executed (center) path at the championship race, and an executed multi-lap path at a testing facility (right). Left: Fastest planned path in color, sub-optimal sampled paths in gray. Center: VIO trajectory as  $p_{VB}$  and corrected estimate as  $p_B$ .

### C.7.2 Attitude Control

The required acceleration from the position controller determines the orientation of the drone’s  $z$ -axis and is used, in combination with a reference yaw angle,  $\varphi_{\text{ref}}$ , to compute the drone’s reference attitude. The reference yaw angle is chosen such that the drone’s  $x$ -axis points towards the reference position 5 m ahead of the current position, i.e., that the drone looks in the direction it flies. A non-linear attitude controller similar to [35] is applied that prioritizes the alignment of the drone’s  $z$ -axis, which is crucial for its translational dynamics, over the correction of the yaw orientation:

$$\boldsymbol{\omega} = \frac{2 \operatorname{sgn}(q_w)}{\sqrt{q_w^2 + q_z^2}} \mathbf{T}_{\text{att}}^{-1} \begin{bmatrix} q_w q_x - q_y q_z \\ q_w q_y + q_x q_z \\ q_z \end{bmatrix}, \quad (\text{C.33})$$

where  $q_w$ ,  $q_x$ ,  $q_y$  and  $q_z$  are the components of the attitude error,  $\mathbf{q}_{IB}^{-1} \otimes \mathbf{q}_{IB_{\text{ref}}}$ , and where  $\mathbf{T}_{\text{att}}$  is a diagonal matrix containing the per-axis first-order system time constants for small attitude errors.

## C.8 Results

The proposed system was used to race in the *2019 AlphaPilot* championship race. The course at the championship race consisted of five gates and had a total length of 74 m. A top view of the race course as well as the results of the path planning and the fastest actual flight are depicted in Fig. C.6 (left and center). With the motion primitive’s maximum velocity set to 8 m/s, the drone successfully completed the race course in a total time of 11.36 s, with only two other teams also completing the full race course. The drone flew at an average velocity of 6.5 m/s and reached the peak velocity of 8 m/s multiple times. Note that due to missing ground truth, Fig. C.6 only shows the estimated and corrected drone position.

The system was further evaluated at a testing facility where there was sufficient space for the drone to fly multiple laps (see Fig. C.6, right), albeit the course consisted of only two

## Appendix C. AlphaPilot: Autonomous Drone Racing

**Table C.2** – Comparison of different network architectures with respect to intersection over union (IoU), precision (Pre.) and recall (Rec.). The index in the architecture name denotes the number of levels in the U-Net. All networks contain one layer per level with kernel sizes of [3, 3, 5, 7, 7] and [12, 18, 24, 32, 32] filters per level. Architectures labelled with 'L' contain twice the amount of filters per level. Timings are measured for single input images of size 352x592 on a desktop computer equipped with an NVIDIA RTX 2080 Ti.

Arch.	IoU	Pre.	Rec.	#params	latency [s]
UNet-5L	0.966	0.997	0.967	613k	0.106
UNet-5	0.964	0.997	0.918	160k	0.006
UNet-4L	0.948	0.997	0.920	207k	0.085
UNet-4	0.941	0.989	0.862	58k	0.005
UNet-3L	0.913	0.991	0.634	82k	0.072
UNet-3	0.905	0.988	0.520	27k	0.005

gates. The drone was commanded to pass four times through *gate 1* before finishing in the *final gate*. Although the gates were not visible to the drone for most of the time, the drone successfully managed to fly multiple laps. Thanks to the global gate map and the VIO state estimate, the system was able to plan and execute paths to gates that are not directly visible. By repeatedly seeing either one of the two gates, the drone was able to compensate for the drift of the VIO state estimate, allowing the drone to pass the gates every time exactly through their center. Note that although seeing *gate 1* in Fig. C.6 (right) at least once was important in order to update the position of the gate in the global map, the VIO drift was also estimated by seeing the *final gate*.

The results of the system’s main components are discussed in detail in the following subsections, and a video of the results is attached to the paper.

### C.8.1 Gate Detection

**Architecture Search:** Due to the limited computational budget of the Jetson Xavier, the network architecture was designed to maximize detection accuracy while keeping a low inference time. To find such architecture, different variants of U-Net [287] are compared. Table C.2 summarizes the performance of different network architectures. Performance is evaluated quantitatively on a separate test set of 4k images with respect to intersection over union (IoU) and precision/recall for corner predictions. While the IoU score only takes full gate detections into account, the precision/recall scores are computed for each corner detection. Based on these results, architecture UNet-5 is selected for deployment on the real drone due to the low inference time and high performance. On the test set, this network achieves an IoU score with the human-annotated ground truth of 96.4%. When only analyzing the predicted corners, the network obtains a precision of 0.997 and a recall of 0.918.

**Deployment:** Even in instances of strong changes in illumination, the gate detector was able to accurately identify the gates in a range of 2 – 17 m. Fig. C.4 illustrates the quality of detections during the championship race (1st row) as well as for cases with



**Table C.3** – Total flight time vs. computation time averaged over 100 runs. The percentage in parenthesis is the computation time with respect to the computational time for the full track.

$N_{gates}$	flight time	computation time
1	9.593,5 s	1.66 ms 2.35%
2	9.291,3 s	18.81 ms 26.56%
3	9.270,9 s	35.74 ms 50.47%
4	9.266,7 s	53.00 ms 74.84%
5 (full track)	9.262,2 s	70.81 ms 100%
CPC [91] (full track)	6.520 s	$4.62 \cdot 10^5$ ms 6524%

multiple gates, represented in the test set (2nd/3rd row). With the network architecture explained in Section C.4, one simultaneous inference for the left- and right-facing camera requires computing 3.86 GFLOPS (40 kFLOPS per pixel). By implementing the network in TensorRT and performing inference in half-precision mode (FP16), this computation takes 10.5 ms on the Jetson Xavier and can therefore be performed at the camera update rate.

### C.8.2 State Estimation

Compared to a pure VIO-based solution, the EKF has proven to significantly improve the accuracy of the state estimation relative to the gates. As opposed to the works by [196, 156, 170], the proposed EKF is not constrained to only use the next gate, but can work with any gate detection and even profits from multiple detections in one image. Fig. C.6 (center) depicts the flown trajectory estimated by the VIO system as  $\mathbf{p}_{VB}$  and the EKF-corrected trajectory as  $\mathbf{p}_B$  (the estimated corrections are depicted in gray). Accumulated drift clearly leads to a large discrepancy between VIO estimate  $\mathbf{p}_{VB}$  and the corrected estimate  $\mathbf{p}_B$ . Towards the end of the track at the two last gates this discrepancy would be large enough to cause the drone to crash into the gate. However, the filter corrects this discrepancy accurately and provides a precise pose estimate relative to the gates. Additionally, the imperfect initial pose, in particular the yaw orientation, is corrected by the EKF while flying towards the first gate as visible in the zoomed section in Fig. C.6 (center).

### C.8.3 Planning and Control

Fig. C.6 (left) shows the nominally planned path for the *AlphaPilot* championship race, where the coloured line depicts the fastest path along all the sampled paths depicted in gray. In particular, a total of  $M = 150$  different states are sampled at each gate, with the velocity limited to 8 m/s and the angle between the velocity and the gate normal limited to  $\varphi_{\max} = 30^\circ$ . During flight, the path is re-planned in a receding horizon fashion through the next  $N = 3$  gates (see Fig. C.6, center). It was experimentally found that

choosing  $N \geq 3$  only has minimal impact of the flight time compared to planning over all gates, while greatly reducing the computational cost. Table C.3 presents the trade-offs between total flight time and computation cost for different horizon lengths  $N$  for the track shown in Fig. C.6 (left). In addition, Table C.3 shows the flight and computation time of the time-optimal trajectory generation from [91], which significantly outperforms our approach but is far away from real-time execution with a computation time of 462 s for a single solution. Online replanning would therefore not be possible, and any deviations from the nominal track layout could lead to a crash.

Please also note that the evaluation of our method is performed in Matlab on a laptop computer, while the final optimized implementation over  $N = 3$  gates achieved replanning times of less than 2 ms on the Jetson Xavier and can thus be done in every control update step. Fig. C.6 (right) shows resulting path and velocity of the drone in a multi-lap scenario, where the drone’s velocity was limited to 6 m/s. It can be seen that drone’s velocity is decreased when it has to fly a tight turn due to its limited thrust.

### C.9 Discussion and Conclusion

The proposed system managed to complete the course at a velocity of 5 m/s with a success rate of 100% and at 8 m/s with a success rate of 60%. At higher speeds, the combination of VIO tracking failures and no visible gates caused the drone to crash after passing the first few gates. This failure could be caught by integrating the gate measurements directly in a VIO pipeline, tightly coupling all sensor data. Another solution could be a perception-aware path planner trading off time-optimality against motion blur and maximum gate visibility.

The advantages of the proposed system are (i) a drift-free state estimate at high speeds, (ii) a global and consistent gate map, and (iii) a real-time capable near time-optimal path planner. However, these advantages could only partially be exploited as the races neither included multiple laps, nor had complex segments where the next gates were not directly visible. Nevertheless, the system has proven that it can handle these situations and is able to navigate through complex race courses reaching speeds up to 8 m/s and completing the championship race track of 74 m in 11.36 s.

While the *2019 AlphaPilot Challenge* pushed the field of autonomous drone racing, in particular in terms of speed, autonomous drones are still far away from beating human pilots. Moreover, the challenge also left open a number of problems, most importantly that the race environment was partially known and static without competing drones or moving gates. In order for autonomous drones to fly at high speeds outside of controlled or known environments and succeed in many more real-world applications, they must be able to handle unknown environments, perceive obstacles and react accordingly. These features are areas of active research and are intended to be included in future versions of the proposed drone racing system.

# D Deep Drone Racing: From Simulation to Reality with Domain Randomization

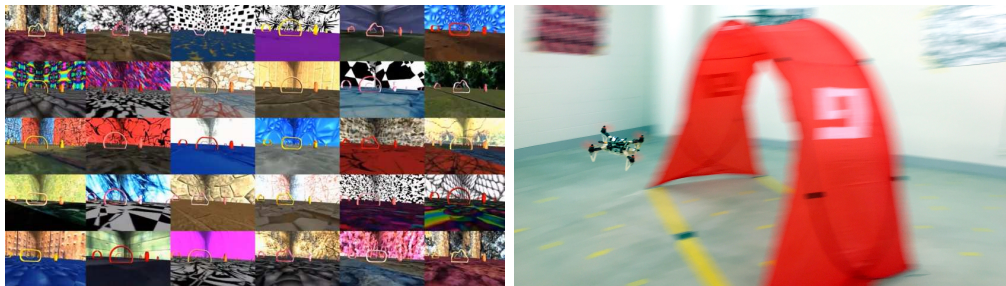
The version presented here is reprinted, with permission, from:

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. Robot.* 36.1 (2019), pp. 1–14. DOI: [10.1109/TRO.2019.2942989](https://doi.org/10.1109/TRO.2019.2942989)

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

Antonio Loquercio\*, Elia Kaufmann\*, René Ranftl, Matthias Müller,  
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.



**Figure D.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.

## D.1 Introduction

Drone racing is a popular sport in which professional pilots fly small quadrotors through complex tracks at high speeds (Fig. D.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 [366].

Developing a fully autonomous racing drone is difficult due to challenges that span dynamics modeling, onboard perception, localization and mapping, trajectory generation, and optimal control. For this reason, autonomous drone racing has attracted significant interest from the research community, giving rise to multiple autonomous drone racing competitions [236, 235].

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 [96, 278, 41]. Indeed, visual inertial odometry [96, 278] 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 [242] and tracker [78], which generate a short trajectory segment

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

---

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 [171] (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 [171], 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 [341]. 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 [171] when evaluated in environments or conditions not seen at training time. Specifically, we demonstrate performance increases of up to 300% in simulation (Fig. D.6) and up to 36% in real-world experiments (Fig. D.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.

### D.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 [96, 248, 216]. Other sensory modalities, e.g. LIDAR or event-based cameras, could partially alleviate these problems [37, 288]. 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 [227, 242, 239] and autonomous cars [184,

164, 174, 358]. 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.

### D.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 [71, 264, 283, 158]. These methods provide several interesting advantages, e.g. robustness against drifts in state estimation [71, 264] and the possibility to learn from failures [158]. 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 [298, 206, 106]. To overcome the problem of acquiring a large amount of annotated data to train a policy, Loquercio et al. [206] proposed to use data from ground vehicles, while Gandhi et al. [106] 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 [120, 66, 46, 52, 247]. This kind of modularity has proven to be particularly important for transferring sensorimotor systems across different tasks [52, 66] and application domains [46, 247].

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.

### D.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 [335, 83, 197]. 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 [156]. 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 [246]. Methods of this type usually predict low-level control

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

---

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.

### D.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 [118, 362, 34, 148, 294, 299]. In driving scenarios, synthetic data was mainly used to train perception systems for high-level tasks, such as semantic segmentation and object detection [284, 150]. One exception is the work of Müller et al. [247], 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 [298] learned a collision avoidance policy by using 3D simulation with extensive domain randomization.

Akin to many of the aforementioned methods, we use domain randomization [341] and modularization [247] 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.

## D.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 [242] that is tracked by a low-level controller [78]. 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  $300 \times 200$  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. [206]. In section D.4.3, we will present the details of our architecture, which was designed to optimize the trade-off between 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 D.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 D.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 D.2). Since we run all computations onboard, we use computationally efficient minimum-jerk trajectories [242] 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. [78].

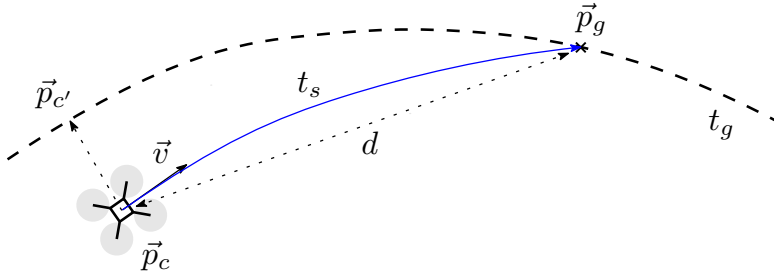
### D.3.1 Training Procedure

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 D.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 [227]. 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

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

---



**Figure D.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.

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 [290], 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 [290] to train our flying policy. On the data collected through the expert policy (Section D.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  $\epsilon$  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 [290] 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  $\epsilon$ , 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  $\epsilon$  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:

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

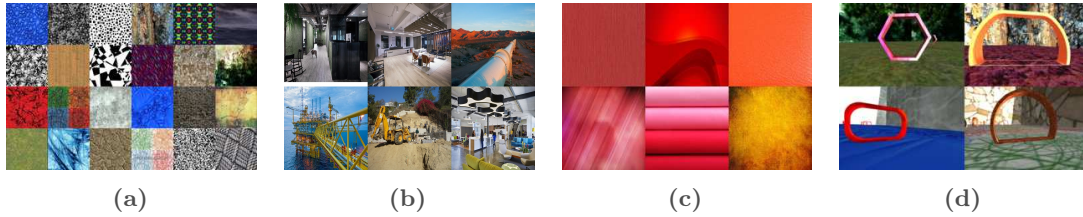
where  $\vec{x}_g$  denotes the groundtruth normalized image coordinates and  $v_g$  denotes the groundtruth normalized speed. By cross-validation, we found the optimal weight to be  $\gamma = 0.1$ , even though the performance was mostly insensitive to this parameter (see Appendix for details).

**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.

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

---



**Figure D.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.

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 D.3a). The gate textures are drawn from a pool of 10 mainly red/orange textures (Figure D.3c). For gate shape randomization (ii), we create 6 gate shapes of roughly the same size as the original gate. Figure D.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 and floor at test time represent common indoor and outdoor environments (Figure D.3b). For testing we use held-out configurations of gate shape and texture not seen during training.

### D.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 [227]. 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  $18 \text{ m s}^{-2}$  and the maximum roll and pitch rate to  $1.5 \text{ rad s}^{-1}$ . The maximum speed was chosen based on the dimensions of the track. For the large simulated track, a maximum speed of  $10 \text{ m s}^{-1}$  was chosen, while on

the smaller real-world track  $6 \text{ m s}^{-1}$ .

**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 D.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:

$$d_{train} = \max(d_{min}, \min(\|s_{last}\|, \|s_{next}\|)) , \quad (\text{D.2})$$

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  $d_{min} = 1.5 \text{ m}$  was used, while for the real track we used  $d_{min} = 1.0 \text{ m}$ .

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

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

---



**Figure D.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.

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:

$$d_{test} = \min [d_{max}, \max (d_{min}, m_d v_{out})] , \quad (\text{D.3})$$

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

## D.4 Experiments

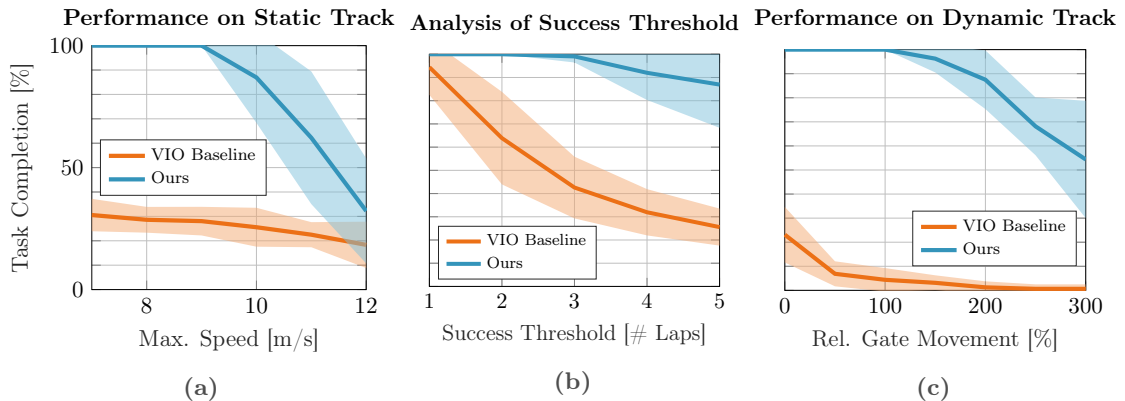
We extensively evaluate the presented approach in a wide range of simulated and real scenarios. We first use a controlled, simulated environment to test the main building blocks of our system, i.e. the convolutional architecture and the perception-control modularization. Then, to show the ability of 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.

### D.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 [102].

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

The platform is spawned in a flying space of cubical shape with side length of 70 meters,



**Figure D.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 D.5a, but the performance is reported for a fixed maximum speed of  $10 \text{ m s}^{-1}$  and different success thresholds. The *y*-axis is shared with Figure D.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  $8 \text{ m s}^{-1}$ . The *y*-axis is shared with Figure D.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.

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 D.4.5.

The large simulated race track (Figure D.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.

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.

## D.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

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

---

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 D.4a).

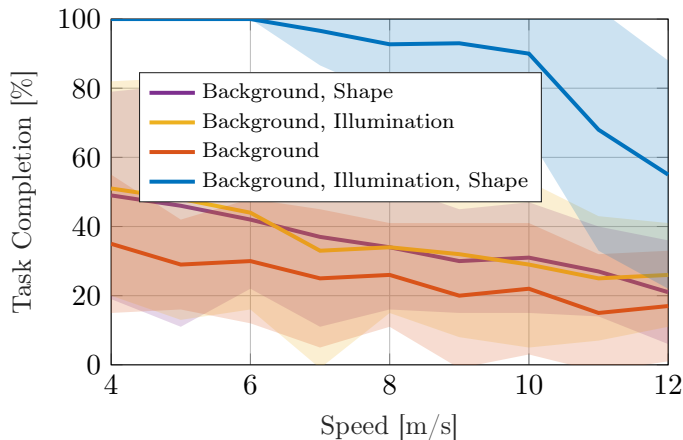
We use this track to compare the performance to a naive deep learning baseline that directly regresses body rates from raw images. Ground truth body rates for the baseline were provided by generating a minimum snap reference trajectory through all gates and then tracking it with a low-level controller [78]. 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 [246] is suboptimal for the task of drone navigation when operating in the real world. Since a quadrotor is an unstable platform [251], 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 [218]. 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 D.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 [96].

The results of this experiment are shown in Figure D.5a. We can observe that the VIO





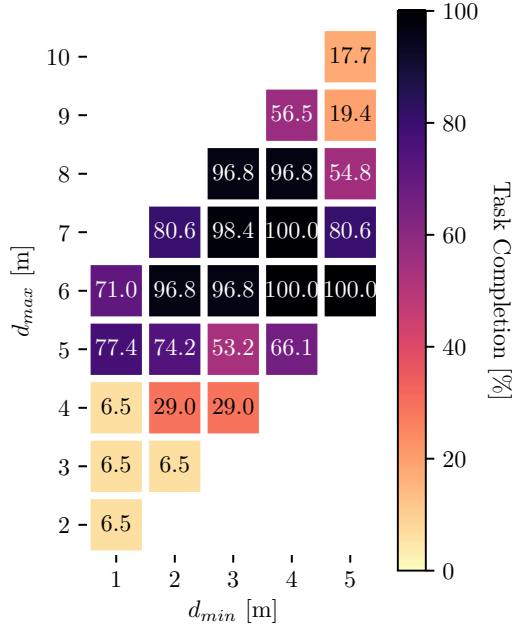
**Figure D.6** – Generalization tests on different backgrounds after domain randomization. More comprehensive randomization increases the robustness of the learned policy to unseen scenarios at different speeds. Lines denote mean performance, while the shaded areas indicate one standard deviation. Background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

baseline, due to accumulated drift, performs worse than our approach. Figure D.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 D.5a), a SLAM-based state estimator [248, 278] 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 [64].

Our approach works reliably up to a maximum speed of  $9 \text{ m s}^{-1}$  and performance degrades 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 D.4b), but dynamically move each gate according to a sinusoidal pattern in each dimension independently. Figure D.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 D.4.1. For this experiment, we kept the maximum platform velocity  $v_{max}$  constant at  $8 \text{ m s}^{-1}$ . 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

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

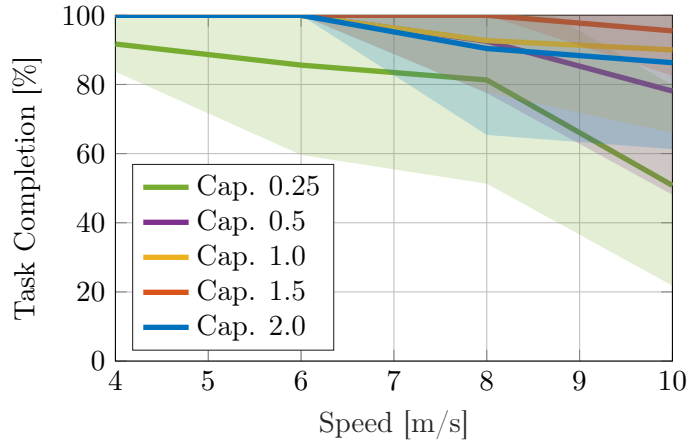


**Figure D.7** – Sensitivity analysis of planning length parameters  $d_{min}$ ,  $d_{max}$  on a simulated track. Maximum speed and (static) track layout are kept constant during the experiment.

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 D.3b) and use gate appearance and illumination conditions held out at training time.

Figure D.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 D.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.



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

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 D.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.

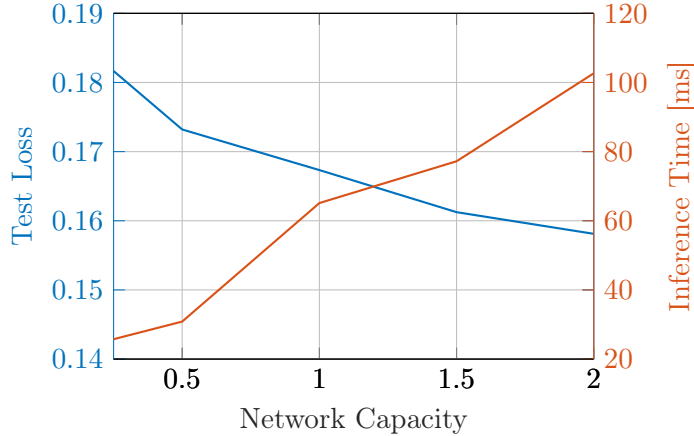
**Sensitivity to planning length.** We perform an ablation study of the *planning length* parameters  $d_{min}$ ,  $d_{max}$  on a simulated track. Both the track layout and the maximum speed ( $10.0 \text{ m s}^{-1}$ ) are kept constant in this experiment. We varied  $d_{min}$  between 1.0 m and 5.0 m and  $d_{max}$  between  $(d_{min} + 1.0)$ m and  $(d_{min} + 5.0)$ m. Figure D.7 shows the results of this evaluation. For each configuration the average *task completion rate* (Section D.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.

### D.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 [206].

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

---



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

Figure D.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 D.8, where the network with capacity 1.5 outperforms the one with capacity 2.0 at high speeds. Indeed, as shown in Figure D.9, larger networks entail smaller inference rates, which result in a decrease in agility.

In our previous conference paper [171], 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  $2 \text{ m s}^{-1}$ , where both 0.5 and 1.0 have equivalent closed-loop control performance (Figure D.8).

### D.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 D.10. We compare our approach to the handcrafted window detector of Falanga et al. [83] by replacing our perception system with the handcrafted detector and leaving the control system unchanged.

Table D.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



Figure D.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%

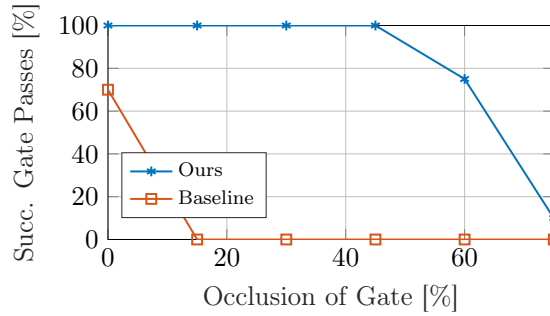
Table D.1 – Success rate for flying through a narrow gap from different initial angles. Each row reports the average of ten runs uniformly spanning the range. The gate was completely invisible at initialization in the experiments marked with \*.

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 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.

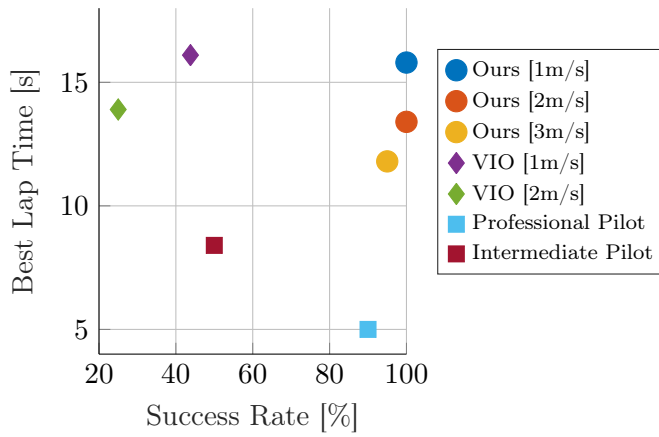
In order to further highlight the robustness and generalization abilities of the approach, we perform experiments with an increasing amount of clutter that occludes the gate. Note that the learning approach has not been trained on such occluded configurations. Figure D.11 shows that our approach is robust to occlusions of up to 50% of the total area of the gate (Figure D.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 D.13. It is composed of four gates and has a total length of 21 meters.

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization



**Figure D.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 D.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 D.4.1.



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

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 [203] and human pilots of 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 [203] for state estimation in order to track the global reference trajectory.

Figure D.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 D.2). It is evident from Figure D.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 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 D.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 [39, 101], 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 D.2 provides a comparison in term of task completion and lap time with respect to a professional

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 D.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 D.4.1.

## Appendix D. Deep Drone Racing: From Simulation to Reality with Domain Randomization

---

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.

### D.4.5 Simulation to Real World Transfer

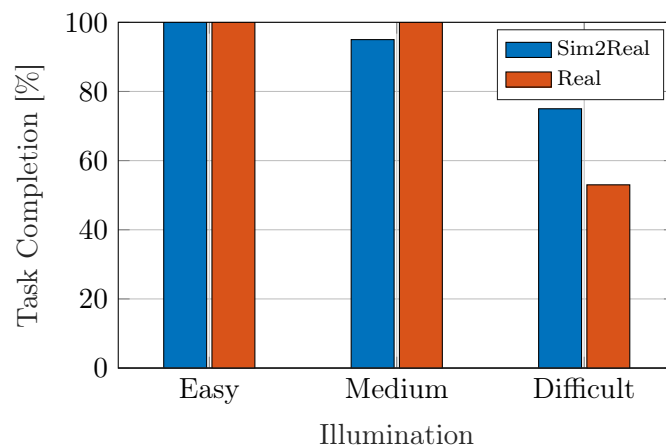
We now attempt direct simulation-to-real transfer of the navigation system. To train the policy in simulation, we use the same process to collect simulated data as in Section D.4.1, i.e. randomization of illumination conditions, gate appearance, and background. The resulting policy, evaluated in simulation in Figure D.6, is then used without any finetuning to fly a real quadrotor. Despite the large appearance differences between the simulated environment (Figure D.3d) and the real one (Figure D.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 supplementary 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 D.4.1 for this evaluation. All experiments are repeated 10 times and the results averaged. The results of this evaluation are shown in Figure D.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.

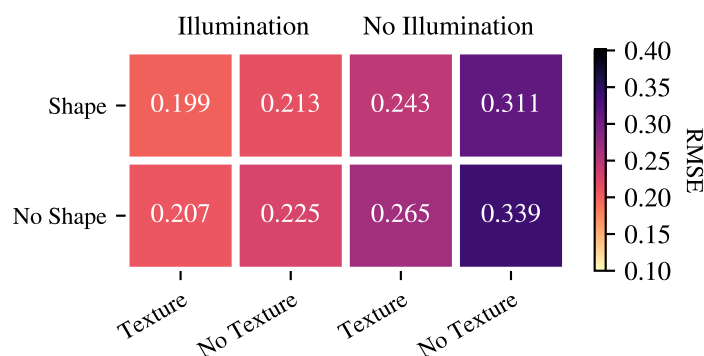
The policy trained in simulation performs on par with the one trained with real data in experiments that have the same illumination conditions as the training data of the real policy. However, when the environment conditions are drastically different (i.e. with very challenging illumination) the policy trained with real data is outperformed by the one trained in simulation. Indeed, as shown by previous work [148], 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 D.3. More specifically, the dataset is composed of





**Figure D.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.



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

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 [148, 341, 298]. We use as metric the Root Mean Square Error (RMSE) in prediction on our collected dataset. As shown in Figure D.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 D.6).

## D.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 [171] 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 [171], 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 [170].

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 [264].



# **E** Deep Drone Acrobatics

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

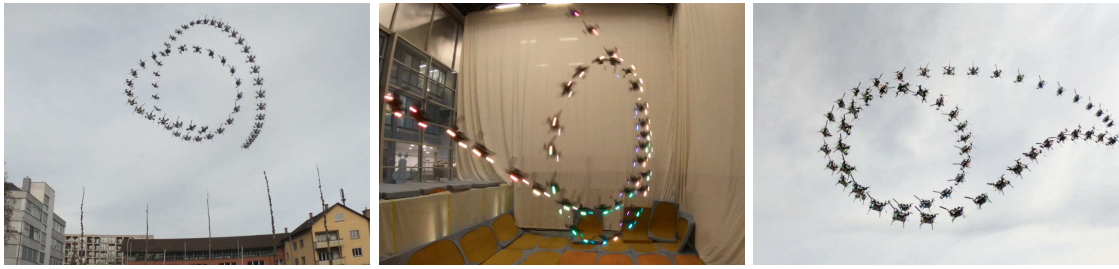
Elia Kaufmann\*, Antonio Loquercio\*, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Deep Drone Acrobatics”. In: *Robotics: Science and Systems (RSS)*. 2020

# Deep Drone Acrobatics

Elia Kaufmann\*, Antonio Loquercio\*, René Ranftl, Matthias Müller,

Vladlen Koltun, 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.



**Figure E.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.

## Supplementary Material

A video demonstrating acrobatic maneuvers is available at [https://youtu.be/2N\\_wKXQ6MXA](https://youtu.be/2N_wKXQ6MXA). Code can be found at [https://github.com/uzh-rpg/deep\\_drone\\_acrobatics](https://github.com/uzh-rpg/deep_drone_acrobatics).

### E.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<sup>1</sup>. Existing autonomous systems that perform agile maneuvers require external sensing and/or external computation [214, 2, 38]. 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 [41]. 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 [214, 2, 145, 38]. 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

<sup>1</sup><https://www.youtube.com/watch?v=T1vzjPa5260>

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 [79, 83, 311] or training end-to-end visuomotor agents [370]. However, acrobatic performance of high-acceleration maneuvers 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. E.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 sensorimotor 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.

### E.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. [214] proposed iterative learning of control strategies to enable platforms to perform multiple flips. Mellinger et al. [228] used a similar strategy to autonomously fly quadrotors through



a tilted window [228]. By switching between two controller settings, Chen et al. [48] also demonstrated multi-flip maneuvers. Abbeel et al. [2] 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. [311], 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. [203] and Falanga et al. [83] 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. [79] and Lee et al. [192] 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. [370] 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. [198] 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 [247] 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 [208, 298], 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.

### E.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  $\mathbf{o}[k] \in \mathbb{O}$  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

## Appendix E. Deep Drone Acrobatics

---

updated at different rates. The controller’s output is an action  $\mathbf{u}[k] = [c, \boldsymbol{\omega}^\top]^\top \in \mathbb{U}$  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* [47]. 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  $\mathbf{s}[k] \in \mathbb{S}$ . The privileged expert is based on a classic optimization-based planning and control pipeline that tracks a reference trajectory from the state  $\mathbf{s}[k]$  using MPC [79].

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 [247, 375].

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.

### E.4 Method

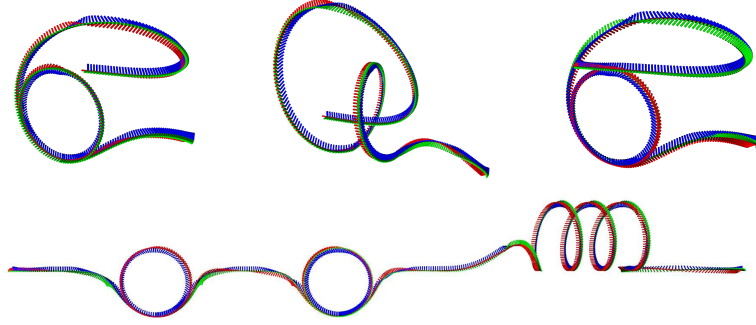
We define the task of flying acrobatic maneuvers with a quadrotor as a discrete-time, continuous-valued optimization problem. Our task is to find an end-to-end control policy  $\pi: \mathbb{O} \rightarrow \mathbb{U}$ , defined by a neural network, which minimizes the following finite-horizon objective:

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

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

We define the quadrotor state  $\mathbf{s}[k] = [\mathbf{p}, \mathbf{q}, \mathbf{v}, \boldsymbol{\omega}]$  as the platform position  $\mathbf{p}$ , its orientation quaternion  $\mathbf{q}$ , and their derivatives. Note that the agent  $\pi$  does not directly observe the state  $\mathbf{s}[k]$ . 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

$$\mathcal{C}(\boldsymbol{\tau}_r[k], \mathbf{s}[k]) = \mathbf{x}[k]^\top \mathcal{L} \mathbf{x}[k], \quad (\text{E.2})$$



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

where  $\mathbf{x}[k] = \boldsymbol{\tau}_r[k] - \mathbf{s}[k]$  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.

#### E.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

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

where  $\mathbf{p}_{\text{WB}}$ ,  $\mathbf{v}_{\text{WB}}$ ,  $\mathbf{q}_{\text{WB}}$  denote the position, linear velocity, and orientation of the platform body frame with respect to the world frame. The gravity vector  ${}_{\text{W}}\mathbf{g}$  is expressed in the world frame and  $\mathbf{q}_{\text{WB}} \odot \mathbf{c}_{\text{B}}$  denotes the rotation of the mass-normalized thrust vector  $\mathbf{c}_{\text{B}} = (0, 0, c)^\top$  by quaternion  $\mathbf{q}_{\text{WB}}$ . The time derivative of a quaternion  $\mathbf{q} = (q_w, q_x, q_y, q_z)^\top$  is given by  $\dot{\mathbf{q}} = \frac{1}{2} \Lambda(\boldsymbol{\omega}) \cdot \mathbf{q}$  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  $\mathbf{J} = \text{diag}(J_{xx}, J_{yy}, J_{zz})$  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*  $\mathbf{z} = [x, y, z, \psi]^\top$  proposed in [227], where  $x, y, z$  denote the position of the quadrotor and  $\psi$  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).

## Appendix E. Deep Drone Acrobatics

---

The core part of each acrobatic maneuver is a circular motion primitive with constant tangential velocity  $\mathbf{v}_l$ . 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  $\mathbf{v}_l$  to be larger by a margin  $\varepsilon$  than the critical tangential velocity that would lead to free fall at the top of the maneuver:

$$\|\mathbf{v}_l\| > \varepsilon\sqrt{rg}, \quad (\text{E.4})$$

where  $r$  denotes the radius of the loop,  $g = 9.81 \text{ m s}^{-2}$ , 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  $\mathbf{z}$  over time:

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

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.

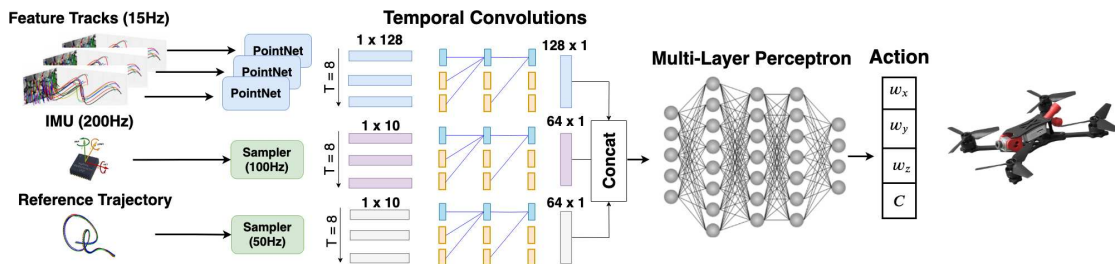
Finally, the trajectories are concatenated to the full reference trajectory, which is then converted back to the full state-space representation  $\boldsymbol{\tau}_r(t)$  [227]. Subsequent sampling with a frequency of 50 Hz results in the discrete-time representation  $\boldsymbol{\tau}_r[k]$  of the reference trajectory. Some example trajectories are illustrated in Figure E.2.

### E.4.2 Privileged Expert

Our privileged expert  $\pi^*$  consists of an MPC [79] 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 [242]:

$$\begin{aligned} \dot{\mathbf{p}}_{\text{WB}} &= \mathbf{v}_{\text{WB}} \\ \dot{\mathbf{v}}_{\text{WB}} &= {}_{\text{W}}\mathbf{g} + \mathbf{q}_{\text{WB}} \odot \mathbf{c}_{\text{B}} \\ \dot{\mathbf{q}}_{\text{WB}} &= \frac{1}{2}\Lambda(\boldsymbol{\omega}_{\text{B}}) \cdot \mathbf{q}_{\text{WB}} \end{aligned} \quad (\text{E.6})$$

In contrast to the model (E.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.



**Figure E.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.

Specifically, the action computed by the MPC is the first element of the solution to the following optimization problem:

$$\begin{aligned}
 \pi^* = \min_{\mathbf{u}} & \left\{ \mathbf{x}[N]^\top \mathcal{Q} \mathbf{x}[N] \right. \\
 & \left. + \sum_{k=1}^{N-1} \left( \mathbf{x}[k]^\top \mathcal{Q} \mathbf{x}[k] + \mathbf{u}[k]^\top \mathcal{R} \mathbf{u}[k] \right) \right\} \\
 \text{s.t. } & \mathbf{r}(\mathbf{x}, \mathbf{u}) = 0 \\
 & \mathbf{h}(\mathbf{x}, \mathbf{u}) \leq 0,
 \end{aligned} \tag{E.7}$$

where  $\mathbf{x}[k] = \boldsymbol{\tau}_r[k] - \mathbf{s}[k]$  denotes the difference between the state of the platform at time  $k$  and the corresponding reference  $\boldsymbol{\tau}_r[k]$ ,  $\mathbf{r}(\mathbf{x}, \mathbf{u})$  are equality constraints imposed by the system dynamics (E.6), and  $\mathbf{h}(\mathbf{x}, \mathbf{u})$  are optional bounds on inputs and states.  $\mathcal{Q}, \mathcal{R}$  are positive-semidefinite cost matrices.

### E.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 [47].

A lemma by Pan et al. [264] formally defines an upper bound between the expert and the student performance as

$$\begin{aligned}
 J(\pi) - J(\pi^*) & \leq C_{\pi^*} \mathbb{E}_{\rho(\pi)} \left[ DW(\pi, \pi^*) \right] \\
 & \leq C_{\pi^*} \mathbb{E}_{\rho(\pi)} \mathbb{E}_{\mathbf{u}^* \sim \pi^*} \mathbb{E}_{\mathbf{u} \sim \pi} [\|\mathbf{u}^* - \mathbf{u}\|],
 \end{aligned} \tag{E.8}$$

where  $DW(\cdot, \cdot)$  is the Wasserstein metric [108] and  $C_{\pi^*}$  is a constant depending on the smoothness of expert actions. Finding an agent  $\pi$  with the same performance as the

## Appendix E. Deep Drone Acrobatics

---

privileged controller  $\pi^*$  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 [289]. 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

$$\pi = \min_{\hat{\pi}} \mathbb{E}_{\mathbf{s}[k] \sim \rho(\pi)} [\|\mathbf{u}^*(\mathbf{s}[k]) - \hat{\pi}(\mathbf{o}[k])\|], \quad (\text{E.9})$$

where  $\mathbf{u}^*(\mathbf{s}[k])$  is the expert action and  $\mathbf{o}[k]$  is the observation vector in the state  $\mathbf{s}[k]$ . Running this process for  $O(N)$  iterations yields a policy  $\pi$  with performance  $J(\pi) \leq J(\pi^*) + O(N)$  [289].

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  $\mathbf{s}[k]$  and how can we protect the platform from damage when the partially trained student  $\pi$  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* [247]. Rather than operating on raw sensory input, our sensorimotor controller operates on an intermediate representation produced by a perception module [375]. 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  $M(\mathbf{z} | \mathbf{s}), L(\mathbf{z} | \mathbf{s}): \mathbb{S} \rightarrow \mathbb{O}$  denote the observation models in the real world and in simulation, respectively. Such models describe how an on-board sensor measurement  $\mathbf{z}$  senses a state  $\mathbf{s}$ . We further define  $\pi_r = \mathbb{E}_{\mathbf{o}_r \sim M(\mathbf{s})}[\pi(\mathbf{o}_r[k])]$  and  $\pi_s = \mathbb{E}_{\mathbf{o}_s \sim L(\mathbf{s})}[\pi(\mathbf{o}_s[k])]$  as the realizations of the policy  $\pi$  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 1.** *For a Lipschitz-continuous policy  $\pi$  the simulation-to-reality gap  $J(\pi_r) - J(\pi_s)$  is upper-bounded by*

$$J(\pi_r) - J(\pi_s) \leq C_{\pi_s} K \mathbb{E}_{\rho(\pi_r)} [DW(M, L)], \quad (\text{E.10})$$

where  $K$  denotes the Lipschitz constant.

*Proof.* The lemma follows directly from (E.8) and the fact that

$$\begin{aligned} DW(\pi_r, \pi_s) &= \inf_{\gamma \in \Pi(\mathbf{o}_r, \mathbf{o}_s)} \mathbb{E}_{(\mathbf{o}_r, \mathbf{o}_s)}[d_p(\pi_r, \pi_s)] \\ &\leq K \inf_{\gamma \in \Pi(\mathbf{o}_r, \mathbf{o}_s)} \mathbb{E}_{(\mathbf{o}_r, \mathbf{o}_s)}[d_o(\mathbf{o}_r, \mathbf{o}_s)] \\ &= K \cdot DW(M, L), \end{aligned}$$

where  $d_o$  and  $d_p$  are distances in observation and action space, respectively.  $\square$

We now consider the effect of abstraction of the input observations. Let  $f$  be a mapping of the observations such that

$$DW(f(M), f(L)) \leq DW(M, L). \quad (\text{E.11})$$

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. (E.11).

As the following lemma shows, training on such representations reduces the gap between task performance in simulation and the real world.

**Lemma 2.** *A policy that acts on an abstract representation of the observations  $\pi_f: f(\mathbb{O}) \rightarrow \mathbb{U}$  has a lower simulation-to-reality gap than a policy  $\pi_o: \mathbb{O} \rightarrow \mathbb{U}$  that acts on raw observations.*

*Proof.* The lemma follows directly from (E.10) and (E.11).  $\square$

#### E.4.4 Sensorimotor Controller

In contrast to the expert policy  $\pi^*$ , the student policy  $\pi$  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 [95, 75], (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 reference trajectory, followed by a multi-layer perceptron that produces actions. The architecture is illustrated in Fig. E.3. Similarly to visual-inertial odometry systems [51,

## Appendix E. Deep Drone Acrobatics

Maneuver	Input			Power Loop		Barrel Roll		Matty Flip		Combo	
	Ref	IMU	FT	Error (↓)	Success (↑)	Error (↓)	Success (↑)	Error (↓)	Success (↑)	Error (↓)	Success (↑)
VIO-MPC	✓	✓	✓	43 ± 14	<b>100%</b>	79 ± 43	<b>100%</b>	92 ± 41	<b>100%</b>	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	<b>100%</b>	64 ± 24	95%	67 ± 29	<b>100%</b>	134 ± 113	85%
Ours	✓	✓	✓	<b>24 ± 5</b>	<b>100%</b>	<b>58 ± 9</b>	<b>100%</b>	<b>53 ± 15</b>	<b>100%</b>	<b>128 ± 57</b>	<b>95%</b>

**Table E.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.

75, 95], 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.

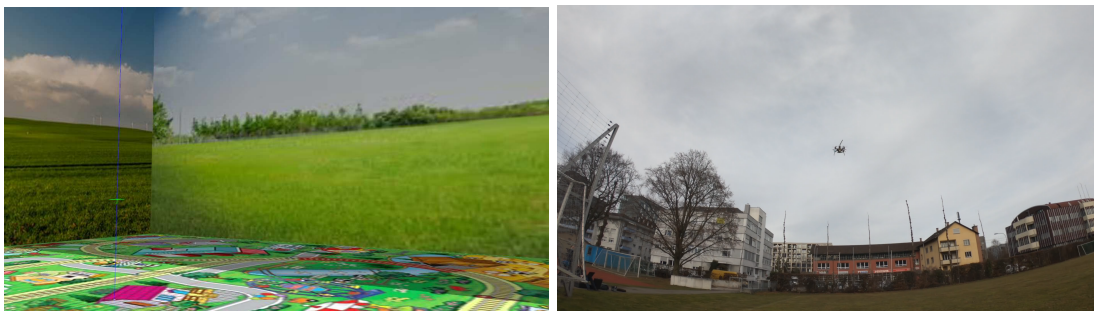
The distribution of raw IMU measurements depends on the exact sensor as well as environmental factors such as pressure and temperature. Instead of using the raw measurements as input to the policy, we preprocess the IMU signal by applying bias subtraction and gravity alignment. Modern visual-inertial odometry systems perform a similar pre-integration of the inertial data in their optimization backend [278]. 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 [15]. 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 [278] to generate feature tracks. The frontend extracts Harris corners [124] and tracks them using the Lucas-Kanade method [210]. We perform





**Figure E.4** – Example images from simulation (left) and the real test environment (right).

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 [280] 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.

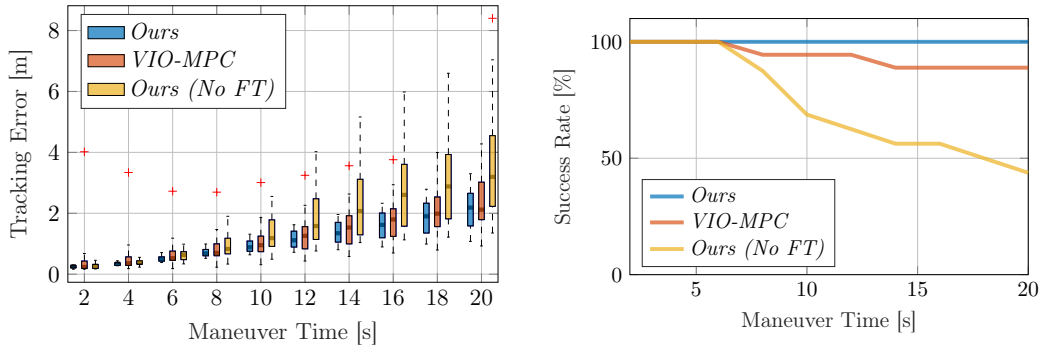
We account for the different input frequencies by allowing each of the input branches to operate asynchronously. Each branch operates independently from the others by generating an output only when a new input from the sensor arrives. The multi-layer perceptron uses the latest outputs from the asynchronous branches and operates at 100 Hz. It outputs control commands at approximately the same rate due to its minimal computational overhead.

#### E.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 [102]. 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. E.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

## Appendix E. Deep Drone Acrobatics



**Figure E.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.

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 [180] 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 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.

## E.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.

### E.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 m to a speed of  $4.5 \text{ ms}^{-1}$  and enter a

- loop maneuver with a radius of  $r = 1.5$  m.
2. Barrel Roll: Accelerate over a distance of 3 m to a speed of  $4.5 \text{ m s}^{-1}$  and enter a roll maneuver with a radius of  $r = 1.5$  m.
  3. Matty Flip: Accelerate over a distance of 4 m to a speed of  $4.5 \text{ m s}^{-1}$  while yawing  $180^\circ$  and enter a backwards loop maneuver with a radius of  $r = 1.5$  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. E.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 [278] and model predictive control [79]. 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.

### E.5.2 Experiments in Simulation

We first evaluate the performance for individual maneuvers in simulation. The results are summarized in Table E.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.

Table E.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.

## Appendix E. Deep Drone Acrobatics

Input	Train		Test 1		Test 2	
	Error ( $\downarrow$ )	Success ( $\uparrow$ )	Error ( $\downarrow$ )	Success ( $\uparrow$ )	Error ( $\downarrow$ )	Success ( $\uparrow$ )
Image	$90 \pm 32$	80%	$\infty$	0%	$\infty$	0%
Ours	<b><math>53 \pm 15</math></b>	<b>100%</b>	<b><math>58 \pm 18</math></b>	<b>100%</b>	<b><math>61 \pm 11</math></b>	<b>100%</b>

**Table E.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.

Maneuver	Power Loop	Barrel Roll	Matty Flip
Ours (No FT)	100%	90%	100%
Ours	100%	100%	100%

**Table E.3** – Success rate across 10 runs on the physical platform.

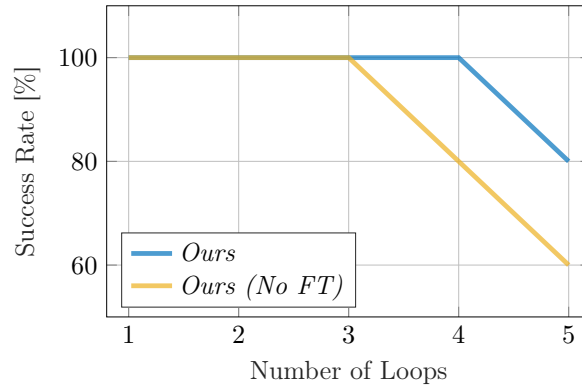
However, the longer the maneuver, the larger the drift accumulated by purely-inertial (*Ours (No FT)*) controllers. When both inertial and visual data is incorporated (*Ours*), drift is reduced and accuracy improves. For the longest sequence (*Combo*), the abstracted visual input raises the success rate by 10 percentage points.

Fig. E.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 E.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 [375]. 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 [200].) In contrast, our approach maintains a 100% success rate in these conditions.

### E.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 E.1 to fly a physical quadrotor, with no fine-tuning. Despite



**Figure E.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.

the differences in the appearance of simulation and reality (see Fig. E.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 E.3. Our controllers can fly all maneuvers with no intervention. An additional experiment is presented in Fig. E.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.

## E.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.



# F 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. “Learning High-Speed Flight in the Wild”. In: *Science Robotics*. 2021

# Learning High-Speed Flight in the Wild

Antonio Loquercio\*, Elia Kaufmann\*, René Ranftl, Matthias Müller, Vladlen

Koltun, 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>

### F.1 Introduction

Quadrotors are among the most agile and dynamic machines ever created<sup>1</sup> [351, 3]. 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 [131]. 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 [80].

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 [128, 103, 303, 76, 33], while others focus on planning without considering perception errors [39, 282, 6, 202]. Numerous systems that combine online mapping with traditional planning algorithms have been proposed to achieve autonomous flight in previously unknown environments [257, 261, 233, 20, 372, 295, 345, 50, 371]. A taxonomy of prior works is presented in Figure F.9 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 [371]. Their sequential nature also introduces additional latency, making high-speed and agile maneuvers difficult to impossible [80]. While these issues can be mitigated to some degree by careful

---

<sup>1</sup>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.

Appendix F. Learning High-Speed Flight in the Wild

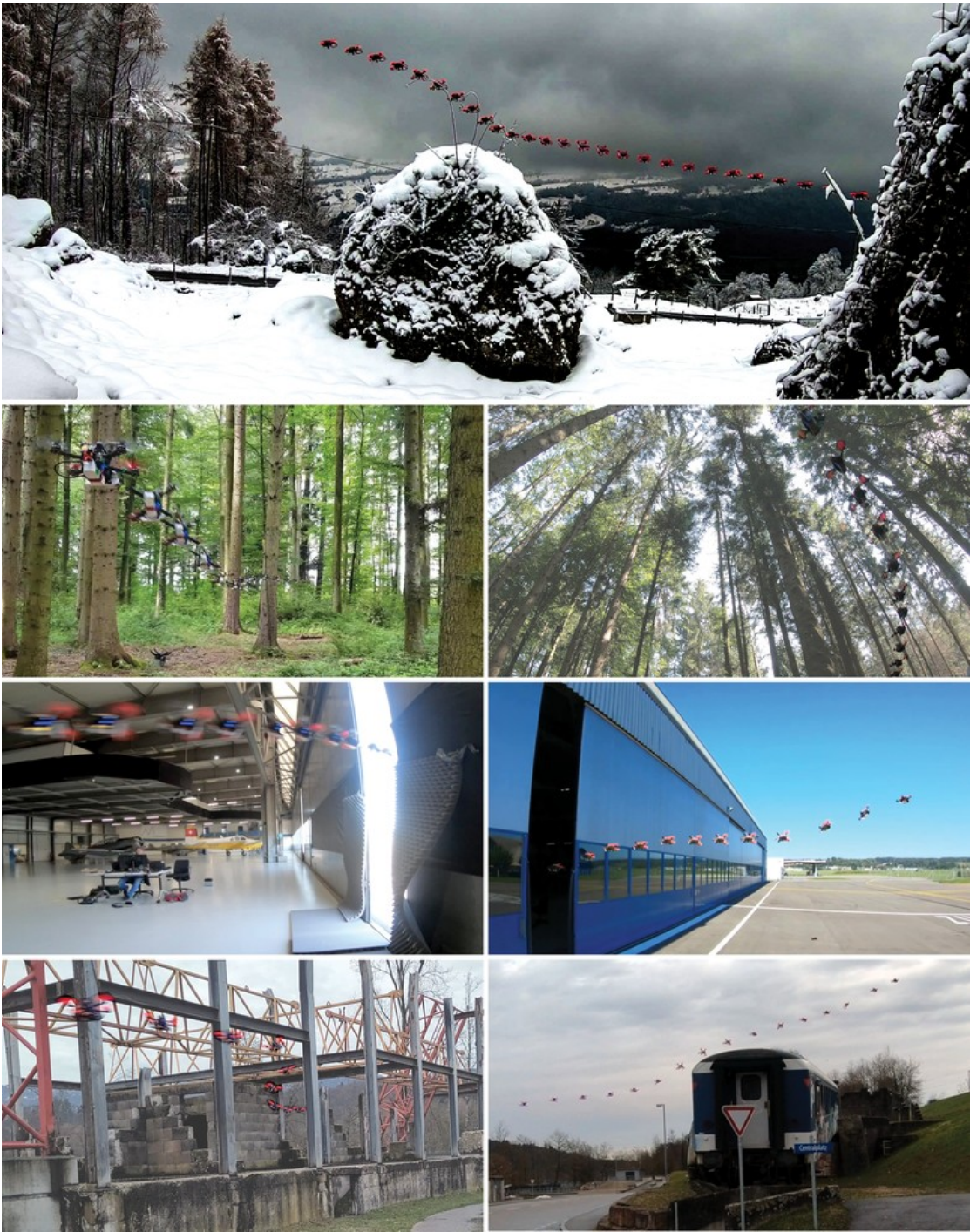


Figure F.1 – Deployment of the presented approach in challenging environments. To get a better sense of the speed and agility of the autonomous system, please watch <https://youtu.be/uTWcC6IBsE4>.

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 [204].

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 [290, 298, 106, 206]. These policies are trained by imitating a human [290, 206], from experience that was collected in simulation [298], or directly in the real world [106]. 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 [206, 106, 290] and/or discrete actions [298], at the cost of reduced maneuverability and agility. More recent work has demonstrated that very agile control policies can be trained in simulation [172]. 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 [247, 172]. 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 [47] 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 [39, 6, 202] 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 [202].

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

## Appendix F. Learning High-Speed Flight in the Wild

---

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 F.2). These results suggest that our methodology enables a multitude of applications that rely on agile autonomous drones with purely onboard sensing and computation.

### F.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 F.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 at <https://youtu.be/uTWC6IBsE4>.

#### F.2.1 High-Speed Flight in the Wild

We tested our approach in diverse real-world environments, as illustrated in Figures F.1 and F.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 F.3-C) or nearby obstacles (Figure F.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 tested in.



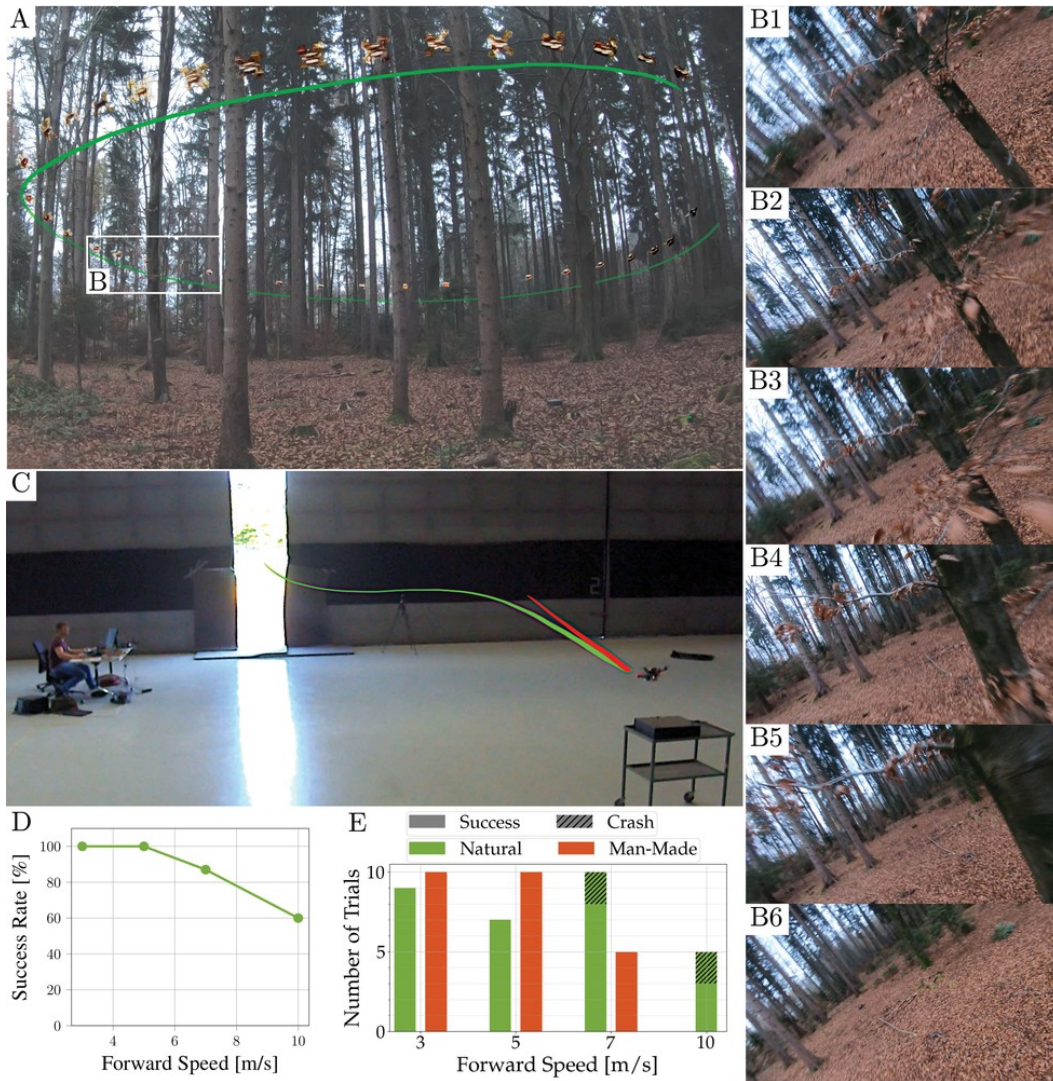
**Figure F.2** – Testing Environments. Zero-shot generalization of our approach in complex natural (A to F) and man-made (H to O) environments. The encountered obstacles can often be avoided in multiple directions and have very different size and structure.

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

## Appendix F. Learning High-Speed Flight in the Wild

of 56 experiments at different speeds. We report the cumulative and individual success rates at various speeds in Figures F.3-D and F.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 F.5. 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 F.3** – Evaluation in indoor and outdoor environments. (A) A circular path in the forest at an average speed of  $7 \text{ m s}^{-1}$ . (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 F.2-A

to F.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 F.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  $3 - 10\text{ m s}^{-1}$ . 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 F.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  $3\text{ m s}^{-1}$  and  $5\text{ m s}^{-1}$  our approach consistently completed the task without experiencing a single crash. For comparison, state-of-the-art methods with comparable actuation, sensing, and computation [372, 372] achieve in similar environments a maximum average speed of  $2.29\text{ m s}^{-1}$ . To study the limit of the system, we set the platform’s average speed to  $7\text{ m s}^{-1}$ . 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  $7\text{ m s}^{-1}$ , we push the average flight speed even higher to  $10\text{ m s}^{-1}$ . 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 F.2-G to F.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 F.2-M), and exit a building through a single narrow opening (c.f. Figure F.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  $3 - 7\text{ m s}^{-1}$ . Given the lower success rate experienced in the forest environment at  $10\text{ m s}^{-1}$ , we did not test at higher speeds to avoid fatal crashes. As shown in Figure F.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<sup>2</sup>. Specifically, the drones are required to exit a hangar by passing through a narrow gap of about 0.8 m in width

---

<sup>2</sup>The commercial drone (a Skydio R1) was tasked to follow a person running through the narrow gap. The speed of this drone cannot be enforced nor controlled, and was therefore estimated in post-processing.

(Figure F.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 F.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  $2.7 \text{ m s}^{-1}$  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  $3 \text{ m s}^{-1}$  and  $5 \text{ m s}^{-1}$  and never experienced a crash.

### F.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. [372] (*FastPlanner*) and the reactive planner of Florence et al. [89] (*Reactive*). The first approach [372] 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 [89] 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.

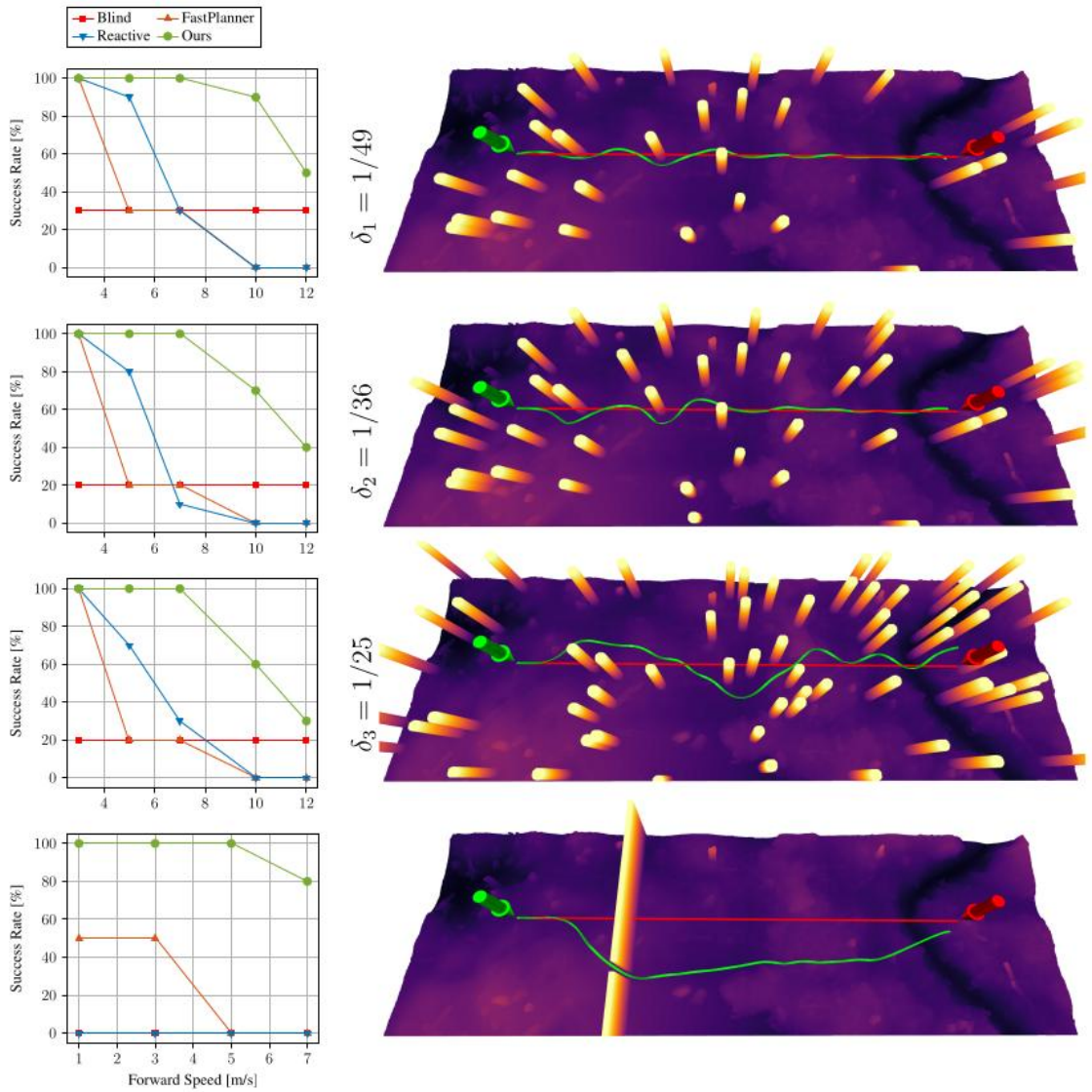
We perform all experiments in the Flightmare simulator [320] using the RotorS [102] Gazebo plugin for accurate physics modeling and Unity as a rendering engine [153]. The experiments are conducted in two types of environments that resemble the setup of our real-world experiments: a forest and a narrow gap. We build these environments by adding trees and walls to the uneven ground of an out-of-the-box Unity environment<sup>3</sup>.

**Forest environments.** We build a simulated forest [165] 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 randomly placed according to a homogeneous Poisson point process  $P$  with intensity  $\delta \text{ treem}^{-2}$  [165]. Note that Tomppo et al. [344] 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  $\delta$ . We set  $w = 30 \text{ m}$  and

---

<sup>3</sup><https://assetstore.unity.com/packages/3d/vegetation/forest-environment-dynamic-nature-150668>





**Figure F.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  $\delta$ . 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 at all speeds.

## Appendix F. Learning High-Speed Flight in the Wild

---

$l = 60$  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*)  $\text{treem}^{-2}$ . We vary the average forward speed of the drone between  $3 \text{ m s}^{-1}$  and  $12 \text{ m s}^{-1}$ . We repeat the experiments with 20 different random realizations of the forest for each difficulty, using the same random seed for all baselines.

The first three rows in Figure F.4 show the results of this experiment, together with one example environment for each difficulty. At a low speed of  $3 \text{ m s}^{-1}$ , all methods successfully complete every run even for high difficulties. As speed increases, the success rates of the baselines quickly degrade. At  $10 \text{ m s}^{-1}$ , 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  $5 \text{ m s}^{-1}$ . For speeds of  $10 \text{ m s}^{-1}$ , 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  $12 \text{ m s}^{-1}$  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 F.2.3 and Section F.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  $[-5, 5]$  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 F.4. The reactive and blind baselines consistently fail to solve this task, while FastPlanner has a success rate of up to 50% at  $5 \text{ m s}^{-1}$ , but still consistently fails at  $7 \text{ m s}^{-1}$ . 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  $5 \text{ m s}^{-1}$ . Even at a speed of  $7 \text{ m s}^{-1}$  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 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

Method		$\mu$ [ms]	$\sigma$ [ms]	Perc. [%]	Total Time [ms]
FastPlanner [372]	Sensing	14.6	2.3	22.3	65.2
	Mapping	49.2	8.7	75.5	
	Planning	1.4	1.6	2.2	
Reactive [89]	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 F.1** – Latency ( $\mu$ ) on a desktop computer equipped with a hexacore i7-8700 CPU and a GeForce RTX 2080 GPU. The standard deviation  $\sigma$  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 [372] and Reactive [89] 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 F.6. The proposed methodology is significantly faster than prior works.

our approach a significant performance advantage with respect to the baselines.

### F.2.3 Computational Cost

In this section, we compare the computational cost of our algorithm with the baselines. Table F.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.

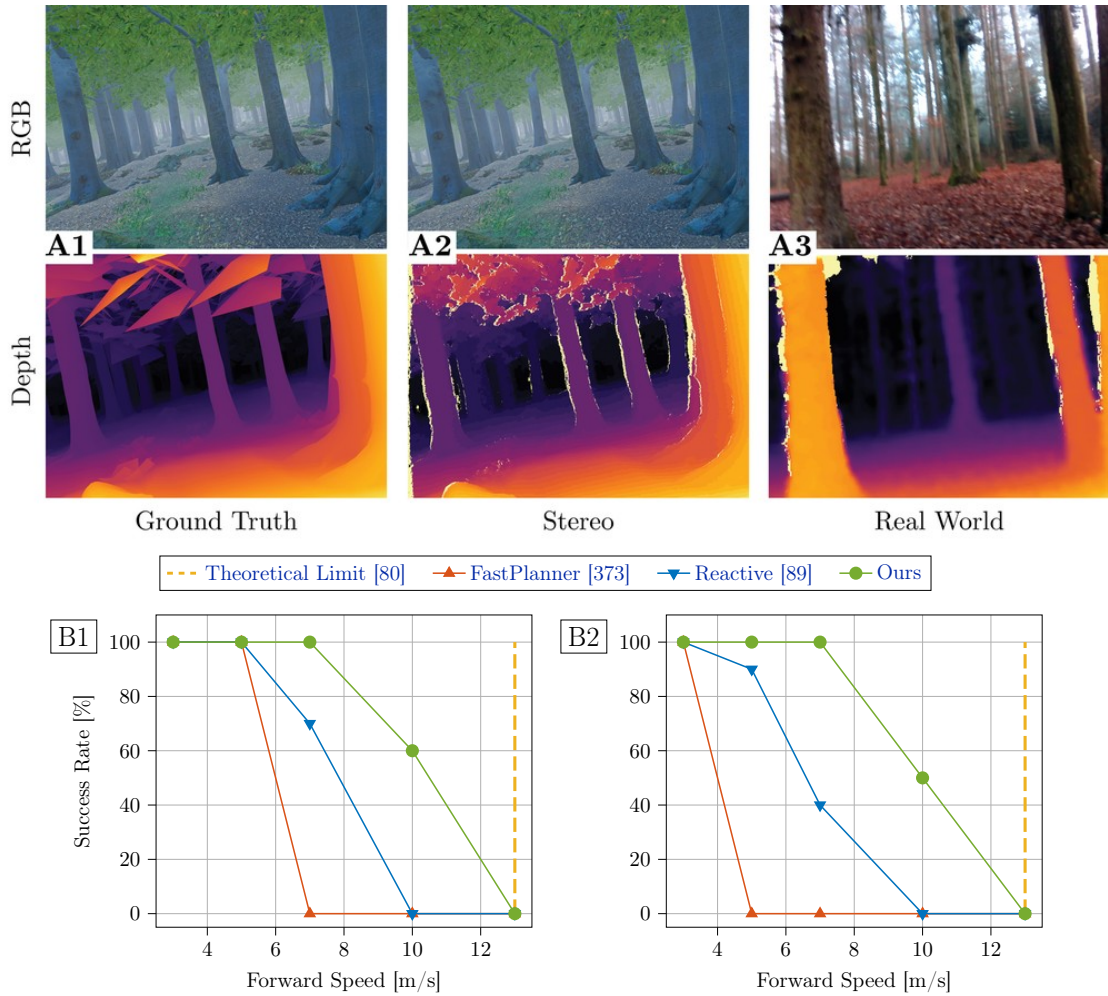
Our approach has significantly lower latency than both baselines; when network inference is performed on the GPU, our approach is 25.3 times faster than *FastPlanner* and 7.4 times faster than the *Reactive* baseline. When GPU inference is disabled, the network’s latency increases by only 8 ms, and our 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.

### F.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. [80] 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 F.7.

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  $v_{max} = 13 \text{ ms}^{-1}$ . We then perform the controlled experiment with varying forward speeds  $v$  in the range  $3 - 13 \text{ ms}^{-1}$ . 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 [132] to analyze the combined effect of latency and sensing errors.

**Ground-truth depth.** Figure F.5-B1 illustrates the results of this experiment when perfect depth perception (Figure F.5-A1) is available. All approaches can complete the task perfectly up to  $5 \text{ ms}^{-1}$ . However, even in these ideal conditions, the performance of the baselines drops for speeds beyond  $5 \text{ ms}^{-1}$ . This drop in performance for *Reactive* can be attributed to the 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



**Figure F.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.

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  $3\text{ m s}^{-1}$  in the original work [372], and thus was not designed to operate at high speeds. Our approach can successfully avoid the obstacle without a single failure up to  $7\text{ m s}^{-1}$ . For higher speeds, performance gracefully degrades to 60% at  $10\text{ m s}^{-1}$ . 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 F.5-A2). Figure F.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  $5\text{ m s}^{-1}$  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  $7\text{ m s}^{-1}$ . In contrast to the baselines, our approach is only marginally affected by the noisy depth readings, with only a 10% drop in performance at  $10\text{ m s}^{-1}$ , 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.

### F.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  $10\text{ m s}^{-1}$  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  $10\text{ m s}^{-1}$  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  $10\text{ m s}^{-1}$  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 [104], especially in presence of dynamic obstacles [82].

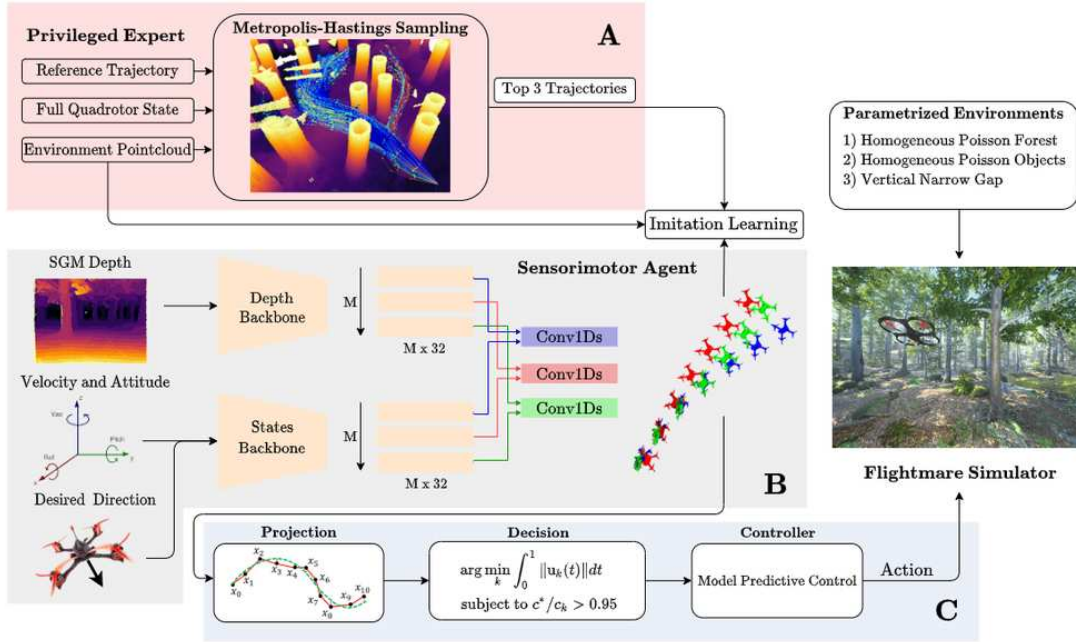
Overall, our approach is a stepping stone towards the development of autonomous systems that can navigate at high speeds through previously unseen environments with only on-board sensing and computation. Combining our short-horizon controller for local obstacle avoidance with a long-term planner is a major opportunity for many robotics applications, including autonomous exploration, delivery, and cinematography.

## F.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  $\mathbf{o}$  consists of a depth image  $\mathbf{d} \in \mathbb{R}^{640 \times 480}$ , an estimate of the platform velocity  $\mathbf{v} \in \mathbb{R}^3$  and attitude (expressed as a rotation matrix)  $\mathbf{q} \in \mathbb{R}^9$ , 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

## Appendix F. Learning High-Speed Flight in the Wild



**Figure F.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.

collision probability and input cost. The controller is trained using privileged learning [47]. 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 F.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 [320] with the RotorS [102] Gazebo plugin and Unity as a rendering engine [153]. Both the training and the testing environments are built by adding obstacles to the uneven ground of an out-of-the-box Unity environment<sup>4</sup>.

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

<sup>4</sup><https://assetstore.unity.com/packages/3d/vegetation/forest-environment-dynamic-nature-150668>



we use depth as an abstract input representation that shows only a negligible domain shift from simulation to the real-world (cf. Figure F.5-A). Specifically, the sensorimotor agent is trained with depth images that have been computed using Semi-Global Matching (SGM) [132] 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 [175], 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.

### F.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  $P(\boldsymbol{\tau} \mid \boldsymbol{\tau}_{ref}, \mathcal{C})$  is conditioned on the reference trajectory  $\boldsymbol{\tau}_{ref}$  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  $\boldsymbol{\tau}_{ref}$ . We define  $P$  as the following:

$$P(\boldsymbol{\tau} \mid \boldsymbol{\tau}_{ref}, \mathcal{C}) = \frac{1}{Z} \exp(-c(\boldsymbol{\tau}, \boldsymbol{\tau}_{ref}, \mathcal{C})) \quad (\text{F.1})$$

where  $Z = \int_{\boldsymbol{\tau}} P(\boldsymbol{\tau} \mid \boldsymbol{\tau}_{ref}, \mathcal{C})$  is the normalization factor and  $c(\boldsymbol{\tau}, \boldsymbol{\tau}_{ref}, \mathcal{C}) \in \mathbb{R}_+$  is a cost function indicating proximity to the reference and distance from obstacles. We define the trajectory cost function as

$$c(\boldsymbol{\tau}, \boldsymbol{\tau}_{ref}, \mathcal{C}) = \int_0^1 \lambda_c C_{collision}(\boldsymbol{\tau}(t)) + [\boldsymbol{\tau}(t) - \boldsymbol{\tau}_{ref}(t)]^\top \boldsymbol{Q} [\boldsymbol{\tau}(t) - \boldsymbol{\tau}_{ref}(t)] dt \quad (\text{F.2})$$

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  $r_q = 0.2$  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:

$$C_{collision}(\boldsymbol{\tau}(t)) = \begin{cases} 0 & \text{if } d_c > 2r_q \\ -d_c^2/r_q^2 + 4 & \text{otherwise.} \end{cases} \quad (\text{F.3})$$

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

## Appendix F. Learning High-Speed Flight in the Wild

---

with the Metropolis-Hastings (M-H) algorithm [125] 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 F.8, 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  $\boldsymbol{\tau}_{\text{bspline}} \in \mathbb{R}^{3 \times 3}$  curve with 3 control points and a uniform knot vector, enabling interpolation with high computational efficiency [105]. Cubic B-Splines are twice continuously differentiable and have a bounded derivative in a closed interval. Due to the differential flatness property of quadrotors [227], 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 [227, 79]. 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 0.1s and evaluate the discrete version of Equation (F.2). Specifically, we sample a total of 50K trajectories using a Gaussian with a variance of 2, 5, 10 that increases every 26K 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  $\boldsymbol{\tau}_{\text{ref}}$  in Equation (F.2) with a global collision-free trajectory  $\boldsymbol{\tau}_{\text{gbl}}$  from start to goal, that we compute using the approach of Liu et al. [202]. As illustrated in Figure F.10, conditioning sampling on  $\boldsymbol{\tau}_{\text{gbl}}$  practically increases the horizon of the expert and generates more conservative trajectories. An animation explaining our expert is available in <https://youtu.be/uTWcC6IBsE4>. 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.

### F.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) [132], 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 [139] 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  $\tau$  and its collision cost. In summary, our architecture receives as input a depth image  $\mathbf{d} \in \mathbb{R}^{640 \times 480}$ , the platform’s velocity  $\mathbf{v} \in \mathbb{R}^3$ , the drone’s attitude expressed as a rotation matrix  $\mathbf{q} \in \mathbb{R}^9$ , 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.*  $\mathcal{T}_n = \{(\tau_n^k, c_k) \mid k \in [0, 1, \dots, M - 1]\}$ , 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.*  $\tau_n^k \in \mathbb{R}^{10 \times 3}$ . Specifically, the network trajectory  $\tau_n^k$  is described by:

$$\tau_n^k = [\mathbf{p}(t_i)]_{i=1}^{10}, \quad t_i = \frac{i}{10}, \quad (\text{F.4})$$

where  $\mathbf{p}(t_i) \in \mathbb{R}^3$  is the drone’s position at time  $t = t_i$  relative to its current state  $\mathbf{x}$ . 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:

$$\text{R-WTA}(\mathcal{T}_e, \mathcal{T}_n) = \sum_{i=0}^{|\mathcal{T}_e|} \sum_{k=0}^{|\mathcal{T}_n|} \alpha(\tau_{e,p}^i, \tau_n^k) \|\tau_{e,p}^i - \tau_n^k\|^2, \quad (\text{F.5})$$

where  $\mathcal{T}_e$  and  $\mathcal{T}_n$  are the set of expert and network trajectories,  $\tau_{e,p}$  denotes the position component of  $\tau_e$ , and  $\alpha(\cdot)$  is defined as

$$\alpha(\tau_{e,p}^i, \tau_n^k) = \begin{cases} 1 - \epsilon & \text{if } \|\tau_{e,p}^i - \tau_n^k\|^2 \leq \|\tau_{e,p}^j - \tau_n^k\|^2 \quad \forall j \neq i \\ \frac{\epsilon}{M-1} & \text{otherwise.} \end{cases} \quad (\text{F.6})$$

Intuitively, an expert trajectory  $\tau_e^i \in \mathcal{T}_e$  is associated with the closest network trajectory  $\tau_n^k \in \mathcal{T}_n$  with a weight of  $2 - \epsilon = 0.95$  and with  $\epsilon/(M - 1) = 0.025$  to all remaining

## Appendix F. Learning High-Speed Flight in the Wild

---

hypotheses. This formulation, proposed by Rupprecht et al. [292], prevents mode collapse and was shown to outperform other popular approaches for multi-modal learning such as Mixture Density Networks [31]. 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 (F.3). In summary, the final training loss for each sample is equal to:

$$\mathcal{L}((\mathcal{T}_e, \mathcal{T}_n)) = \lambda_1 \text{R-WTA}(\mathcal{T}_e, \mathcal{T}_n) + \lambda_2 \sum_{k=0}^{|\mathcal{T}_n|} \|c_k - C_{collision}(\boldsymbol{\tau}_n^k)\|^2, \quad (\text{F.7})$$

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 [180] and a learning rate of  $2e-3$ .

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 [227]. Considering for example the  $x$  axis, we define the polynomial projection  $\mu_x(t) = \mathbf{a}_x^\top \cdot \mathbf{T}(t)$ , where  $\mathbf{a}_x^\top = [a_0, a_1, \dots, a_5]$  and  $\mathbf{T}(t)^\top = [1, t, \dots, t^5]$ . The projection term  $\mathbf{a}_x$  is found by solving the following optimization problem:

$$\begin{aligned} & \underset{\mathbf{a}_x}{\text{minimize}} && \sum_{i=1}^{10} \left( \boldsymbol{\tau}_{n,x}^{k,i} - \mathbf{a}_x^\top \cdot \mathbf{T}\left(\frac{i}{10}\right) \right)^2 \\ & \text{subject to} && s_x(0) - \mathbf{a}_x^\top \cdot \mathbf{T}(0) = 0 \\ & && \dot{s}_x(0) - \mathbf{a}_x^\top \cdot \dot{\mathbf{T}}(0) = 0 \\ & && \ddot{s}_x(0) - \mathbf{a}_x^\top \cdot \ddot{\mathbf{T}}(0) = 0 \end{aligned} \quad (\text{F.8})$$

where  $\boldsymbol{\tau}_{n,x}^{k,i}$  is the  $x$  component of the  $i^{\text{th}}$  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. [227]. The one with the lowest input costs is tracked by a model-predictive controller [79]. 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.

### F.4.3 Training Environments

We build custom environments in the Flightmare simulator [320] 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 F.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  $\delta$ .

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  $\tau_{gbl}$  from the starting location to a point 40 m in front of it. The trajectory  $\tau_{gbl}$  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.

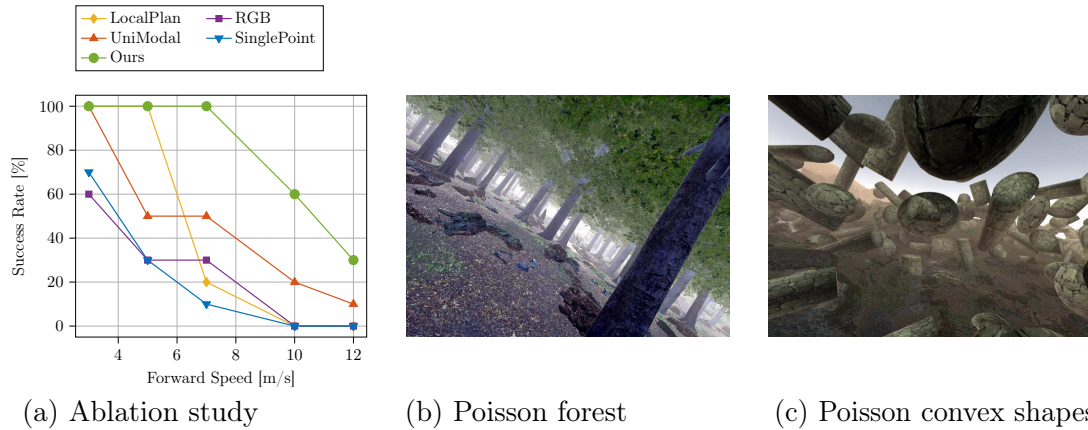
To assure sufficient coverage of the state space, we use the dataset aggregation strategy (Dagger) [289]. This process consists of rolling out the student policy and labeling the visited states with the expert policy. To avoid divergence from  $\tau_{gbl}$  and prevent crashes in the early stages of training, we track a trajectory predicted by the student only if the drone’s distance from the closest point on  $\tau_{gbl}$  is smaller than a threshold  $\xi$ , initialized to zero. Otherwise, we directly track  $\tau_{gbl}$  with a model-predictive controller. Every 30 environments the student is re-trained on all available data and the threshold  $\xi$  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 200 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 20K 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 [132] 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 F.5.

### F.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 F.7 show that all

## Appendix F. Learning High-Speed Flight in the Wild



**Figure F.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.

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 [375, 172] 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 [208], *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.

### F.5 Experimental Platform

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.



**Figure F.8** – 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.

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 (640x480px) at 30 Hz, which we use without further processing. The horizontal field of view of this observation is approximately  $90^\circ$ , which appeared to be sufficient for the task of obstacle avoidance. For experiments at speeds of  $7 \text{ m s}^{-1}$  and above, we tilt the depth sensor by  $30^\circ$  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<sup>5</sup> [129] 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.

<sup>5</sup><https://github.com/dhernandez0/sgm>

The platform only receives a start and stop command from the base computer, and is therefore completely autonomous during flight.

### F.6 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 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.

### F.7 Rotational Dynamics

Extending [80] to the quadrotor platform, we approximate the maximum speed  $v_{\max}$  that still allows successful avoidance of a vertical cylindrical obstacle of radius  $r_{\text{obs}}$ . 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

$$t_{\text{rot}} = \sqrt{\frac{2\phi J}{T_{\text{max}}}}, \quad (\text{F.9})$$

with  $J$  being the moment of inertia,  $T_{\text{max}}$  the maximum torque the quadrotor can produce, and  $\phi$  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  $\phi$  accelerates with full thrust, its lateral position  $p_{\text{lat}}$  can be described by

$$p_{\text{lat}}(t) = \frac{1}{2} \sin \phi \cdot c_{\text{max}} \cdot t^2, \quad (\text{F.10})$$

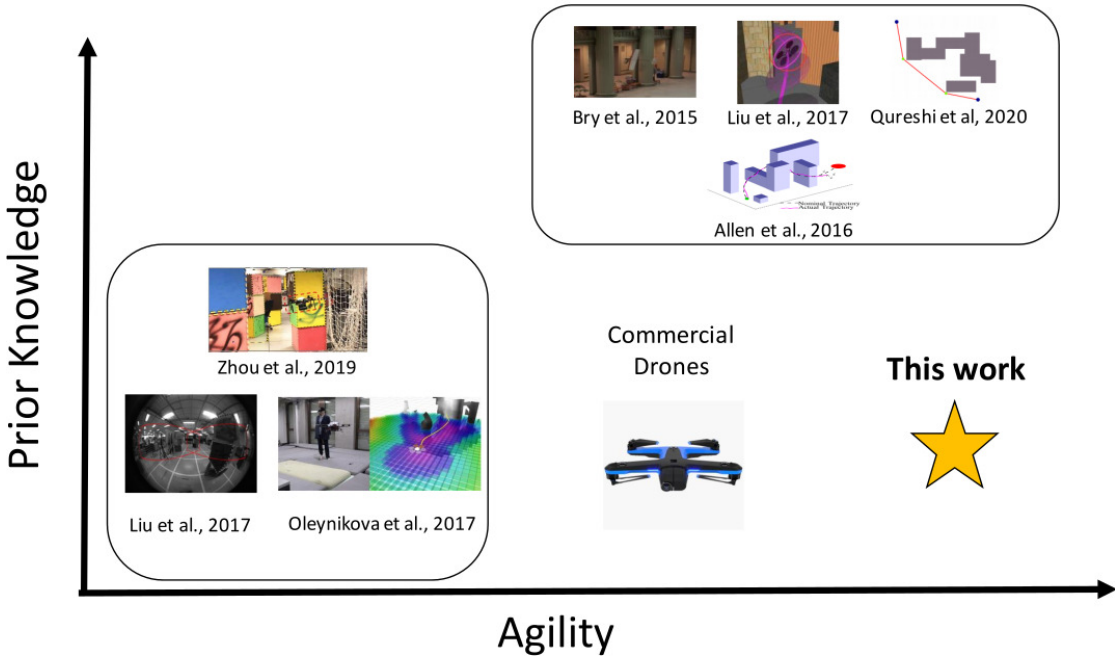
where  $c_{\text{max}}$  denotes the maximum mass-normalized thrust of the platform. Solving this equation for  $t$  and setting  $p_{\text{lat}} = r_{\text{obs}}$  allows to formulate a maximum linear speed that still allows successful avoidance when considering the sensing range  $s$ , the sensing latency  $t_{\text{s}}$ , and the latency introduced to reorient the platform  $t_{\text{rot}}$ :

$$v_{\text{max}} = \frac{s}{t_{\text{s}} + t_{\text{rot}} + \sqrt{\frac{2r_{\text{obs}}}{\sin \phi \cdot c_{\text{max}}}}}. \quad (\text{F.11})$$

Inserting (F.9) into (F.11) we can identify  $\phi$  that maximizes  $v_{\text{max}}$ . In our study case, we set  $r_{\text{obs}} = 0.5$  m, which is the sum of the radius of the pole to avoid and the size of the drone. In addition, we set  $c_{\text{max}} = 33 \text{ m s}^{-2}$ ,  $J = 0.007 \text{ kg m}^{-2}$ , and  $t_{\text{s}} = 66$  ms, since images are rendered at 15 Hz. Using these values, we derive a rotational latency  $t_{\text{rot}} = 119$  ms and a rotation angle  $\phi = 69.1^\circ$ . 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  $v_{\text{max}}$  according to Equation F.11. Note that this approximation does not account for the fact that the quadrotor platform already performs some lateral acceleration during the rotation phase.

## F.8 Metropolis-Hastings Sampling

In statistics, the Metropolis-Hastings (M-H) algorithm [125] 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 [10]. 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



**Figure F.9** – 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.

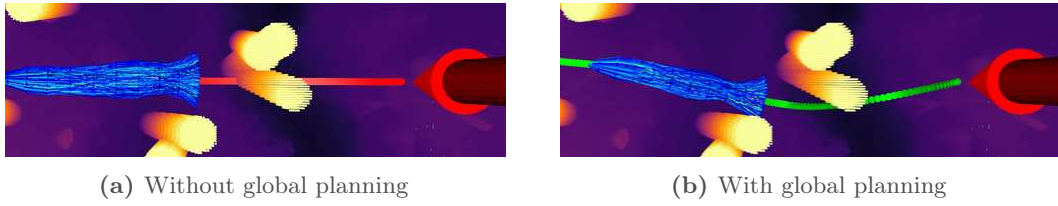
sample  $w_{t+1}$  is then accepted and used for the next iteration, or it is rejected, discarded, and the current sample  $w_t$  is re-used. Specifically, the sample is accepted with probability equal to

$$\alpha = \min \left( 1, \frac{d(w_{t+1})}{d(w_t)} \right) = \min \left( 1, \frac{P(w_{t+1})}{P(w_t)} \right). \tag{F.12}$$

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 [285] have shown that under the mild condition that

$$\alpha > 0 \quad \forall \quad w_t, w_{t+1} \in \mathcal{W}, \tag{F.13}$$

$\hat{P}(w)$  will asymptotically converge to the target distribution  $P(w)$ . According to Eq. (F.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 [285].



**Figure F.10** – 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.



# G A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

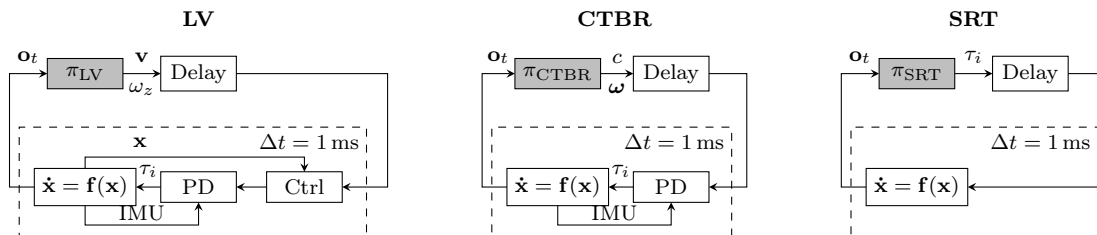
The version presented here is reprinted, with permission, from:

Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. “A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022

# A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

Elia Kaufmann, Leonard Bauersfeld, Davide Scaramuzza

**Abstract** — Quadrotors are highly nonlinear dynamical systems that require carefully tuned controllers to be pushed to their physical limits. Recently, learning-based control policies have been proposed for quadrotors, as they would potentially allow learning direct mappings from high-dimensional raw sensory observations to actions. Due to sample inefficiency, training such learned controllers on the real platform is impractical or even impossible. Training in simulation is attractive but requires to transfer policies between domains, which demands trained policies to be robust to such domain gap. In this work, we make two contributions: (i) we perform the first benchmark comparison of existing learned control policies for agile quadrotor flight and show that training a control policy that commands body-rates and thrust results in more robust sim-to-real transfer compared to a policy that directly specifies individual rotor thrusts, (ii) we demonstrate for the first time that such a control policy trained via deep reinforcement learning can control a quadrotor in real-world experiments at speeds over 45km/h.



**Figure G.1** – In this paper, we compare three different classes of control policies for the task of agile quadrotor flight. From left to right: policies specifying desired linear velocities (LV) (they rely on a control stack that maps the output velocities to individual rotor thrusts), policies commanding collective thrust and bodyrates (CTBR) (they rely on a low-level controller that maps the output bodyrates to individual rotor thrusts), policies directly outputting single-rotor thrust (SRT).

## Supplementary Material

A narrated video illustrating our findings is available at <https://youtu.be/zqdfVq2uWUA>

## G.1 Introduction

Agile quadrotor flight is a challenging problem that requires fast and accurate control strategies. In recent years, numerous learning-based controllers have been proposed for quadrotors. In contrast to their traditional counterparts, learned control policies have the potential to directly map sensory information to actions, alleviating the need for explicit state estimation [370, 172, 195, 209].

Prior work has proposed learned control policies that make use of various control input modalities to the underlying platform: while some directly specify motor commands [370, 145, 234], others, instead, output desired collective thrust and bodyrates [172, 246] (that are then executed by a low-level controller), or velocity commands [112, 206] (that are then executed by a control stack), or even a sequence of future waypoints [209]. Most published approaches do not justify their choice of control input. This renders performance comparisons among them and, thus, scientific progress difficult.

Due to the high sample complexity of learning-based policies, they are often trained in simulation, which then requires transferring the policy from simulation to the real world. This transfer between domains is known to be hard and is typically approached by increasing the simulation fidelity [337, 23], by randomization of dynamics [234, 12] or rendering properties [298, 341] at training time, or by abstraction of the policy inputs [172, 209]. Apart from simulation enhancements and input abstractions, also the choice of action space of the learned policy itself can facilitate transfer. Policies that generate high-level commands, such as desired linear velocity or future waypoints [209], have a reduced simulation to reality gap, as they abstract the task of flying by relying on an existing underlying control stack. However, while facilitating transfer, such abstractions also constrain the maneuverability of the platform. Approaches that do not rely on such abstractions (like those specifying collective thrust and body rates or even single-rotor-

## Appendix G. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

---

thrust commands) can potentially execute much more agile maneuvers, but have so far only been shown for near-hover trajectories [234] or require a dedicated policy for each maneuver [172].

In this paper, we compare and benchmark learned control policies with respect to their choice of action space. Specifically, we compare them in terms of peak performance in case of perfect model identification, as well as in terms of their transferability to a new platform with possibly different dynamics properties. We compare the learned policies with respect to their flight performance, which we characterize by the average tracking error on a set of predefined trajectories.

Our experiments, performed both in simulation and on a real quadrotor platform, show that control policies that command collective thrust and bodyrates are more robust to changes in the dynamics of the platform without compromising agility. Additionally, compared to high-level action parameterizations, specifying collective thrust and bodyrates allows performing significantly more agile maneuvers.

Finally, we demonstrate the first learning-based controller, trained via deep reinforcement learning, that is able to perform previously unseen agile maneuvers on a real quadrotor flying at speeds over  $45 \text{ km h}^{-1}$ . The policy is trained purely in simulation and transferred to the real platform without any fine-tuning.

### G.2 Related Work

In this section, we give an overview of the related work for learning-based quadrotor control while focusing on the choice of action space. While there exists a comparison of action spaces of learned policies for 2D locomotion [268], such analysis is still lacking in the aerial robotics community. In the following, we group learned control strategies according to their action space into a) Linear Velocity Commands (LV), b) Collective Thrust and Bodyrates (CTBR), and c) Single Rotor Thrusts (SRT).

**Linear Velocity.** Control policies specifying high-level commands, often in the form of receding-horizon waypoints or velocity commands, have been proposed for a variety of tasks, such as forest trail navigation [112], navigation in city streets [206] and indoor environments [298], or even drone racing [171]. Recently, [24] have used model-based meta reinforcement learning to generate velocity commands that adapt to unknown payloads. While these approaches have been successfully deployed in the real world, only [171] achieved flight speeds beyond  $3 \text{ m s}^{-1}$ , while the other policies result in near-hover flight. As the control policy does not take into account the dynamic constraints of the platform, it can be easily transferred, but does not exploit the platform’s full dynamic capabilities. Furthermore, such approaches rely on an existing underlying control stack, which itself is dependent on high-quality state estimation.

**Collective Thrust and Bodyrates.** Compared to specifying linear velocity commands, controlling collective thrust and bodyrates has been shown to allow performing significantly more aggressive maneuvers. In [246], a racing policy directly maps image observations to



collective thrust and bodyrate commands. Although the policy successfully races through challenging race tracks in simulation, it is not deployed on a real platform. In [312], the authors propose combining a classical controller with a learned residual command to correct for aerodynamic disturbances such as ground effect during near-hover flight. In [172], the authors use privileged learning to imitate a model predictive controller (MPC) to perform acrobatic maneuvers. While this approach successfully showed acrobatic flight on a real platform, it was constrained to a single maneuver and required a separate policy for each trajectory. In contrast to generating high-level commands, specifying collective thrust and bodyrates does not necessitate estimation of the full state of the platform, but only requires inertial measurements to perform feedback control on the bodyrates. This information is readily available at high frequency in today’s flight controllers, rendering collective thrust and bodyrates the preferred control input modality for professional human pilots.

**Single-Rotor Thrusts.** There are several works that propose to directly learn to control individual rotor thrusts [145, 234, 322, 370, 189, 271, 272]. As this control input does not require any additional control loop, it provides direct access to the actuators and as a result correctly represents the true actuation limits of the platform. It constitutes the most versatile control input investigated in this work. In [145, 234], the authors train a policy to map state observations directly to desired individual rotor thrusts. While [145] required a PID controller at data collection time to facilitate training, [234] demonstrated training of a stabilizing quadrotor control policy from scratch in simulation and deployment on multiple real platforms. [322] trains a policy for autonomous drone racing. Their approach demonstrates competitive racing performance in simulation, but is not deployed on a real quadrotor. In [370], the authors train a policy to perform obstacle avoidance using guided policy search by imitating an MPC controller that has access to privileged information about the environment. One of the few works that does not rely on simulated data for training is presented in [189], where the authors propose an approach based on deep model-based reinforcement learning to train a hovering policy for the *Crazyflie* quadrotor. The trained policy managed to control the real platform in hover for 6 s before crashing. A position controller is trained via reinforcement learning in [271] and extended in [272] to be robust against external disturbances such as wind. In [181], the authors train an attitude controller via deep reinforcement learning. They argue that their approach provides a better flight performance compared to a PID controller, while still being computationally lightweight. Although this method outputs individual rotor thrusts, it is still dependent on a higher-level controller that produces attitude setpoints to achieve stable flight.

While some of these works show successful deployment of their policies in the real world, none achieved agile flight, only reaching maximum speeds below  $4 \text{ m s}^{-1}$ .

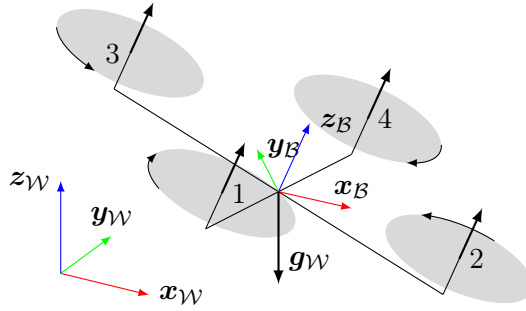


Figure G.2 – Diagram of the quadrotor depicting the world and body frames and propeller numbering.

### G.3 Quadrotor Dynamics

To train a control policy for agile flight, we implement the quadrotor dynamics as an environment in TensorFlow Agents<sup>1</sup>. The following section gives a brief overview of the dynamics implemented in the simulator.

#### G.3.1 Notation

Scalars are denoted in non-bold  $[s, S]$ , vectors in lowercase bold  $\mathbf{v}$ , and matrices in uppercase bold  $\mathbf{M}$ . World  $\mathcal{W}$  and Body  $\mathcal{B}$  frames are defined with orthonormal basis i.e.  $\{\mathbf{x}_{\mathcal{W}}, \mathbf{y}_{\mathcal{W}}, \mathbf{z}_{\mathcal{W}}\}$ . The frame  $\mathcal{B}$  is located at the center of mass of the quadrotor. A vector from coordinate  $\mathbf{p}_1$  to  $\mathbf{p}_2$  expressed in the  $\mathcal{W}$  frame is written as:  ${}_{\mathcal{W}}\mathbf{v}_{12}$ . If the vector's origin coincides with the frame it is described in, the frame index is dropped, e.g. the quadrotor position is denoted as  $\mathbf{p}_{\mathcal{W}\mathcal{B}}$ . Unit quaternions  $\mathbf{q} = (q_w, q_x, q_y, q_z)$  with  $\|\mathbf{q}\| = 1$  are used to represent orientations, such as the attitude state of the quadrotor body  $\mathbf{q}_{\mathcal{W}\mathcal{B}}$ .

Finally, full SE3 transformations, such as changing the frame of reference from body to world for a point  $\mathbf{p}_{\mathcal{B}1}$ , can be described by  ${}_{\mathcal{W}}\mathbf{p}_{\mathcal{B}1} = {}_{\mathcal{W}}\mathbf{t}_{\mathcal{W}\mathcal{B}} + \mathbf{q}_{\mathcal{W}\mathcal{B}} \odot \mathbf{p}_{\mathcal{B}1}$ . Note the quaternion-vector product is denoted by  $\odot$  representing a rotation of the vector by the quaternion as in  $\mathbf{q} \odot \mathbf{v} = \mathbf{q}\mathbf{v}\bar{\mathbf{q}}$ , where  $\bar{\mathbf{q}}$  is the quaternion's conjugate.

#### G.3.2 Quadrotor Dynamics

The quadrotor is assumed to be a 6 degree-of-freedom rigid body of mass  $m$  and diagonal moment of inertia matrix  $\mathbf{J} = \text{diag}(J_x, J_y, J_z)$ . Furthermore, the rotational speeds of the four propellers  $\Omega_i$  are modeled as first-order system with time constant  $k_{\text{mot}}$  where the commanded motor speeds  $\mathbf{\Omega}_{\text{cmd}}$  are the input.

---

<sup>1</sup><https://github.com/tensorflow/agents>

The state space is thus 17-dimensional and its dynamics can be written as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}}_{\mathcal{WB}} \\ \dot{\mathbf{q}}_{\mathcal{WB}} \\ \dot{\mathbf{v}}_{\mathcal{WB}} \\ \dot{\boldsymbol{\omega}}_{\mathcal{B}} \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} = \begin{bmatrix} \mathbf{v}_{\mathcal{W}} \\ \mathbf{q}_{\mathcal{WB}} \cdot \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{\mathcal{B}}/2 \end{bmatrix} \\ \frac{1}{m} (\mathbf{q}_{\mathcal{WB}} \odot (\mathbf{f}_{\text{prop}} + \mathbf{f}_{\text{drag}})) + \mathbf{g}_{\mathcal{W}} \\ \mathbf{J}^{-1} (\boldsymbol{\tau}_{\text{prop}} - \boldsymbol{\omega}_{\mathcal{B}} \times \mathbf{J} \boldsymbol{\omega}_{\mathcal{B}}) \\ \frac{1}{k_{\text{mot}}} (\boldsymbol{\Omega}_{\text{cmd}} - \boldsymbol{\Omega}) \end{bmatrix}, \quad (\text{G.1})$$

where  $\mathbf{g}_{\mathcal{W}} = [0, 0, -9.81 \text{ m/s}^2]^\top$  denotes earth's gravity,  $\mathbf{f}_{\text{prop}}$ ,  $\boldsymbol{\tau}_{\text{prop}}$  are the collective force and the torque produced by the propellers, and  $\mathbf{f}_{\text{drag}}$  is a linear drag term. The quantities are calculated as follows:

$$\mathbf{f}_{\text{prop}} = \sum_i \mathbf{f}_i, \quad \boldsymbol{\tau}_{\text{prop}} = \sum_i \boldsymbol{\tau}_i + \mathbf{r}_{\text{P},i} \times \mathbf{f}_i, \quad (\text{G.2})$$

$$\mathbf{f}_{\text{drag}} = - [k_{vx} v_{\mathcal{B},x} \quad k_{vy} v_{\mathcal{B},y} \quad k_{vz} v_{\mathcal{B},z}]^\top, \quad (\text{G.3})$$

where  $\mathbf{r}_{\text{P},i}$  is the location of propeller  $i$  expressed in the body frame,  $\mathbf{f}_i$ ,  $\boldsymbol{\tau}_i$  are the forces and torques generated by the  $i$ -th propeller, and  $(k_{vx}, k_{vy}, k_{vz})$  [102, 78] are linear drag coefficients. A commonly used [102, 309] model for the forces and torques exerted by a single propeller is presented in the following: the thrust and drag torque are assumed to be proportional to the square of the propellers' rotational speed. The corresponding thrust and drag coefficients  $c_t$  and  $c_d$  can be readily identified on a static propeller test stand. By also measuring the rotational speed of the propeller during those tests, the motor time constant  $k_{\text{mot}}$  can be estimated. Overall, the force and torque produced by a single propeller are modeled as follows:

$$\mathbf{f}_i(\boldsymbol{\Omega}) = [0 \quad 0 \quad c_t \cdot \boldsymbol{\Omega}^2]^\top, \quad \boldsymbol{\tau}_i(\boldsymbol{\Omega}) = [0 \quad 0 \quad c_d \cdot \boldsymbol{\Omega}^2]^\top \quad (\text{G.4})$$

The dynamics are integrated using a symplectic Euler scheme with step size 1 ms. For numerical values of the identified mass, inertia, and thrust and drag constants, we refer the reader to Section G.4.3.

## G.4 Methodology

We address the challenge of robust and agile quadrotor flight using learned control policies by identifying the best choice of action space for the task. We train deep neural control policies that directly map observations  $\mathbf{o}_t$  in the form of platform state and a reference trajectory to control actions  $\mathbf{u}_t$ . The control policies are trained using model-free reinforcement learning (PPO [308]) purely in simulation on a set of over 600 reference trajectories that cover the full performance envelope of the quadrotor. We train policies of three different types that only differ in their choice of action space  $\mathbf{u}_t$ , as illustrated in Figure G.1:

## Appendix G. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

---

1. **Linear Velocity & Yaw Rate (LV):** Each action specifies a desired linear velocity and yaw rate, which are then tracked by a full control stack with access to accurate state estimation.  $\pi_{LV}(\mathbf{o}_t) \Rightarrow \mathbf{u}_t = \{v_x, v_y, v_z, \omega_z\}$
2. **Collective Thrust & Bodyrate (CTBR):** Each action represents desired collective thrust and bodyrates, which are tracked by a low-level controller using measurements from an inertial sensor.  $\pi_{CTBR}(\mathbf{o}_t) \Rightarrow \mathbf{u}_t = \{c, \omega_x, \omega_y, \omega_z\}$
3. **Single-Rotor Thrust (SRT):** Each action directly specifies desired individual rotor thrusts, which are then applied for the duration of a control step.  $\pi_{SRT}(\mathbf{o}_t) \Rightarrow \mathbf{u}_t = \{f_1, f_2, f_3, f_4\}$

All policy types feature a 4-dimensional action space, are fed the same observations  $\mathbf{o}_t$ , and are represented by the same network architecture.

### G.4.1 Observations, Actions, and Rewards

An observation  $\mathbf{o}_t$  obtained from the environment at time  $t$  consist of (i) a history of previous states and applied actions and (ii) the future reference along the trajectory. Specifically, the state information contains a history of length  $H = 10$  of the  $z$ -position of the platform, its velocity, attitude represented as rotation matrix, and bodyrates. Even though the simulator internally uses quaternions, we pass attitude as rotation matrix to the networks to avoid discontinuities [376]. The reference information consists of a sequence of length  $R = 10$  of future relative position, velocity, and bodyrates as well as the full rotation matrix of the reference. The position and velocity components of the reference states are expressed as the residual from the current state of the quadrotor. All observations are normalized before passing them to the networks.

Since the value network is only used during training time, it can access privileged information about the environment that is not accessible to the policy network. Specifically, this

**Table G.1** – Input features to the policy and value networks. The state is represented by a sliding window of length  $H$  of current and previous states, the reference is represented by a receding-horizon window of length  $R$  of current and future reference states. Both networks observe the same state and reference, but only the value network observes privileged information, such as biases in mass, inertia, drag and gravity applied during training with domain randomization.

Input	Components	Dimensions	Policy NW	Value NW
State	z-Position	$H \times 1$	✓	✓
	Velocity	$H \times 3$	✓	✓
	Attitude	$H \times 9$	✓	✓
	Bodyrates	$H \times 3$	✓	✓
	Privileged Info.	$H \times 7$	✗	✓
Reference	Position	$R \times 3$	✓	✓
	Velocity	$R \times 3$	✓	✓
	Attitude	$R \times 9$	✓	✓
	Bodyrates	$R \times 3$	✓	✓

**Table G.2** – Physical parameters of the simulation. At the start of each rollout, the parameters are sampled from a uniform distribution around the nominal values with the randomization specified above.

Parameter	Nominal Value	Randomization
Mass [kg]	0.768	$\pm 30\%$
Inertia [kg m <sup>2</sup> ]	[2.5e-3, 2.1e-3, 4.3e-3]	$\pm 30\%$
Gravity [m s <sup>-2</sup> ]	[0.0, 0.0, -9.81]	$\pm 0.4$
$k_{vx}$ [N s m <sup>-1</sup> ]	0.3	$\pm 0.3$
$k_{vy}$ [N s m <sup>-1</sup> ]	0.3	$\pm 0.3$
$k_{vz}$ [N s m <sup>-1</sup> ]	0.15	$\pm 0.15$
$c_l$ [N rad <sup>-1</sup> s <sup>-1</sup> ]	1.563e-6	$\pm 0.0$
$c_d$ [N m rad <sup>-1</sup> s <sup>-1</sup> ]	1.909e-8	$\pm 0.0$

privileged information contains the mass and inertia biases applied during randomization, as well as the sampled drag coefficients and the additive gravity bias. An overview of the observation provided to the policy and value network is given in Table G.1. The value network and the policy network share the same architecture but have different parameters. The state and reference information are encoded by two separate fully-connected neural networks with 3 hidden layers with 64 neurons each. The encodings are then concatenated and fed to a final multilayer perceptron with two layers of 128 neurons each.

We use a dense shaped reward formulation to learn the task of agile trajectory tracking. The reward  $r_t$  at timestep  $t$  is given by

$$r_t = - (\mathbf{x}_t - \mathbf{x}_{\text{ref},t})^\top \mathbf{Q} (\mathbf{x}_t - \mathbf{x}_{\text{ref},t}) - (\mathbf{u}_t - \mathbf{u}_{\text{ref},t})^\top \mathbf{R} (\mathbf{u}_t - \mathbf{u}_{\text{ref},t}) - r_{\text{crash}}, \quad (\text{G.5})$$

where  $\mathbf{Q}$  and  $\mathbf{R}$  are diagonal matrices,  $\mathbf{x}_t$  the full state of the quadrotor,  $\mathbf{u}_t$  the applied action,  $\mathbf{x}_{\text{ref},t}$  and  $\mathbf{u}_{\text{ref},t}$  their respective references, and  $r_{\text{crash}}$  is a binary penalty that is only active when the altitude of the platform is negative, which also ends the episode.

### G.4.2 Policy Learning

All control policies are trained using Proximal Policy Optimization (PPO) [308]. PPO has been shown to achieve state-of-the-art performance on a set of continuous control tasks and is well suited for learning problems where interaction with the environment is fast. Data collection is performed by simulating 50 agents in parallel. At each environment reset, every agent samples a new trajectory from the set of training trajectories and is initialized with bounded perturbation at the start of the trajectory.

Inspired by prior work on simulation to reality transfer, we perform randomization of the dynamics of the platform during training time and apply Gaussian noise to the policy observations. Specifically, we randomize mass, inertia, aerodynamic drag, and thrust variations of the quadrotor.

## Appendix G. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

Table G.3 – Training hyperparameters.

Hyperparameter	Value
$\gamma$ (discount factor)	0.98
Actor learning rate	3e-4
Critic learning rate	3e-4
Entropy regularization	1e-2
$\varepsilon$ (importance ratio clipping)	0.2

### G.4.3 Training Details

The policies are trained in a simulated quadrotor environment implemented using TensorFlow Agents. The nominal quadrotor parameters such as mass and inertia are identified from the real platform and are summarized in Table G.2 together with the amount of randomization applied at training time. Training hyperparameters specified in Table G.3.

During trajectory tracking, the agent receives at each timestep a reward that penalizes tracking error and deviation from the reference action as laid out in Eq. (G.5). The matrices  $\mathbf{Q}$  and  $\mathbf{R}$  have nonzero elements only on the diagonal. Specifically, we use  $\mathbf{Q} = \text{diag}\{0.1 \cdot \mathbf{1}_{3 \times 1}, 0.02 \cdot \mathbf{1}_{9 \times 1}, 0.002 \cdot \mathbf{1}_{3 \times 1}, 0.01 \cdot \mathbf{1}_{3 \times 1}\}$  and  $\mathbf{R} = \text{diag}\{0.001 \cdot \mathbf{1}_{4 \times 1}\}$ . The episode is terminated when the quadrotor crashes (i.e.  $p_z \leq 0.0$ ) with a reward of  $r_{\text{crash}} = -500$ .

## G.5 Experiments

We design our experimental setup to investigate the influence of the choice of action space on flight performance. Specifically, we design our experiments to answer the following research questions: (i) How is the peak control performance in situation of perfect model identification affected by the actuation model? (ii) How does the choice of action space affect the robustness against model mismatch? (iii) What is the impact of the choice of action space on training data requirement?

We evaluate the performance of all policies on a set of test trajectories of varying agility, spanning from a hover trajectory up to a racing trajectory [91] that requires to perform accelerations beyond 3g to track. All test trajectories are within the distribution of training trajectories and are feasible, i.e. they do not exceed the platform limits. Table G.4 shows the key metrics of all test trajectories.

### G.5.1 Simulation Experiments

In a set of controlled experiments in simulation, the tracking performance of each policy is investigated. We compare performance with respect to average positional tracking error. Experiments are performed on the test trajectories in two settings: (i) in the *Nominal* setting, the test environment perfectly matches the training environment; (ii) in

the *Model Mismatch* setting, the environment at test time is different from the training environment. Specifically, we use in setting (ii) a quadrotor simulation that was identified from real flight data and uses blade-element momentum theory to accurately model the aerodynamic forces acting on the platform [23]. We also apply a control delay of 20 ms to simulate wireless communication latency. Note that we can only use this simulation at test time, since it is computationally too expensive to run it at training time.

While setting (i) is focused on the maximum possible performance achievable by a method and its training data requirement, setting (ii) investigates the robustness of policies against model mismatch. All policies tested in setting (i) have been trained specifically for the nominal environment without any randomization, while the policies tested in setting (ii) have been trained on a distribution of environments as explained in Section G.4.2. We also compare against two state-of-the-art classical control approaches: MPC-SRT represents an optimization-based controller [346] that directly controls at individual rotor thrust level, while MPC-CTBR makes use of a low-level controller. All learned policies are run at a constant frequency of 50 Hz, while the traditional controllers are executed at 100 Hz.

**Nominal Model.** Table G.5 shows the results of the experiments in the *Nominal* setting (i). SRT and CTBR policies perform comparable in this setting, with CTBR marginally outperforming on slower trajectories, while SRT performs slightly better on the more aggressive maneuvers. These results confirm previous findings from experiments in the domain of 2D locomotion [268]: policies that operate in concert with an underlying low-level controller outperform end-to-end policies. The policies that produce linear velocity commands (LV) perform inferior especially for agile maneuvers. This can be explained by the fact that the action space of linear velocity commands does not correctly represent the dynamic constraints of a quadrotor platform, which leads to a reduced maneuverability. This result extends the findings of [268] and shows that more abstraction does not necessarily lead to better performance. Compared to the learned policies, the traditional control approaches (MPC-SRT, MPC-CTBR) perform significantly better in the *Nominal* setting. This results is expected, as the system dynamics implemented in the MPC exactly match the simulated dynamics of the platform. We still provide these results to allow a comparison with traditional control approaches.

**Table G.4** – Maxima of velocity, mass-normalized collective thrust and bodyrates of the test trajectories.

Trajectory	$\ \mathbf{v}\ _{\max}[\text{m s}^{-1}]$	$c_{\max}[\text{m s}^{-2}]$	$\ \boldsymbol{\omega}\ _{\max}[\text{rad s}^{-1}]$
Hover	0.0	9.81	0.0
RandA	3.87	12.68	1.27
RandB	6.36	13.54	1.52
RandC	8.92	14.52	1.93
RaceA	10.48	16.18	5.74
RaceB	11.97	24.94	8.37
Split-S	12.40	26.35	6.11
RaceC	14.22	33.04	11.56

## Appendix G. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

**Table G.5** – Average positional tracking error in centimeter on each test trajectory in case of no model mismatch. The table reports results for learned policies (SRT, CTBR, LV), and traditional approaches (MPC-SRT, MPC-CTBR). Results report mean and standard deviation for 10 trained policies.

	SRT	CTBR	LV	MPC-SRT	MPC-CTBR
Hover	1.0±0.2	<b>0.6±0.2</b>	7.0±1.6	<b>0.1</b>	0.2
RandA	1.5±0.2	<b>0.9±0.1</b>	15.4±3.0	<b>0.2</b>	0.3
RandB	2.4±0.2	<b>1.6±0.1</b>	61.5±21.0	<b>0.2</b>	0.4
RandC	3.0±0.3	<b>2.0±0.2</b>	85.7±11.5	<b>0.2</b>	0.4
RaceA	<b>5.0±1.2</b>	<b>5.0±1.0</b>	121.1±25.8	<b>0.3</b>	1.3
RaceB	7.1±1.8	<b>6.9±1.5</b>	170.2±16.3	<b>0.7</b>	3.0
Split-S	<b>3.5±0.4</b>	6.6±1.1	92.1±20.8	<b>1.0</b>	2.1
RaceC	<b>9.2±3.2</b>	12.3±2.2	197.9±38.1	<b>1.2</b>	3.9

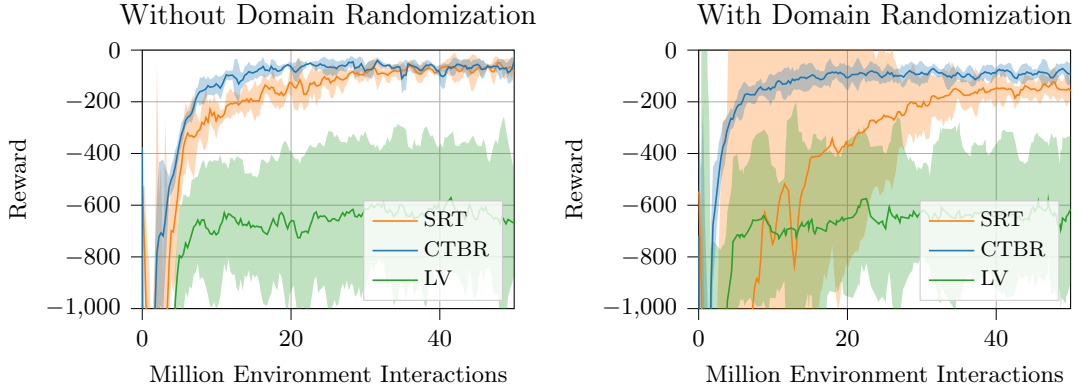
**Table G.6** – Average positional tracking error in centimeter on each test trajectory obtained in a quadrotor simulator based on blade-element momentum theory with a control delay of 20 ms. Results report mean and standard deviation for 10 trained policies.

	SRT	CTBR	LV	MPC-SRT	MPC-CTBR
Hover	11.3±4.5	<b>0.6±0.5</b>	6.7±2.0	1.0	<b>0.5</b>
RandA	12.0±4.0	<b>1.2±0.5</b>	17.8±1.4	3.3	<b>1.1</b>
RandB	14.4±2.4	<b>2.2±0.8</b>	57.0±12.0	7.0	<b>2.0</b>
RandC	17.6±5.9	<b>2.6±0.8</b>	78.9±13.4	8.5	<b>2.6</b>
RaceA	crash	<b>5.6±1.7</b>	144.0±20.1	12.6	<b>4.8</b>
RaceB	crash	<b>10.0±4.0</b>	171.4±17.0	crash	<b>6.3</b>
Split-S	crash	<b>6.9±2.6</b>	83.8±9.7	<b>11.3</b>	11.4
RaceC	crash	<b>14.9±5.5</b>	176.7±22.0	crash	<b>7.5</b>

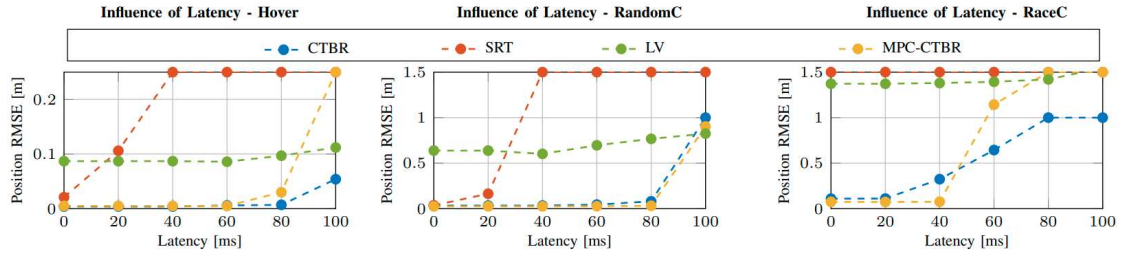
**Model Mismatch.** Table G.6 shows the results of the *Model Mismatch* scenario. Controllers that directly specify single rotor thrusts exhibit a significant reduction in performance, especially for agile trajectories: SRT has a significantly higher tracking error for slow trajectories and often crashes on the faster maneuvers; MPC-SRT also has higher tracking error and even crashes on RaceB and RaceC. We report *crash*, as soon as one policy crashes on the maneuver. The CTBR policies (as well as MPC-CTBR) are less affected by the model mismatch and can still execute all maneuvers with a modest increase in tracking errors. The LV policies show a smaller sensitivity to the model mismatch, but are still consistently outperformed by the CTBR policies on all trajectories.

**Training Data Requirement.** Figure G.3 depicts the learning curves of all policies in case of no domain randomization (left) and with domain randomization (right). All policies have been trained for a total of 50M environment interactions. The learning curves also show the robustness of CTBR and LV to changes in the platform dynamics, we even observed that training with a randomized platform accelerates learning in the early stages of training. In contrast, the learning curves of SRT in case of domain randomization initially exhibit a high variance, train slower, and converge to a final performance substantially lower than in case of no domain randomization.





**Figure G.3** – Learning curves of policies trained without (left) and with (right) domain randomization. All policies are trained for a total of 50M environment interactions. Learning curves show mean performance and standard deviation computed over all trained policies.



**Figure G.4** – Sensitivity to control delay on three trajectories of increasing agility. The results show that policies that operate on single rotor thrust (SRT) are less robust against control delay.

**Influence of Delay.** Our experiments show that the performance of the tested control policies varied significantly in case of unknown control delay. Figure G.4 shows that policies that operate at higher abstraction levels such as LV or CTBR are less sensitive to such delay. Furthermore, accurate identification of control delay is more important for agile trajectories; while hover is possible for CTBR without a noticeable decrease in performance for latencies up to 60 ms, the same latency leads to a crash on the racing trajectory.

**Table G.7** – Positional tracking error in centimeter on a set of test trajectories executed in the real world. Results report mean and standard deviation for 5 trained policies.

	CTBR	LV	MPC-CTBR
Hover	4.4±1.4	6.2±2.0	3.0
RandA	8.1±1.0	60.0±16.8	8.0
RandB	8.6±0.8	87.0±30.3	8.0
RandC	47.8±9.9	134.8±19.6	14.0
Circle	31.8±4.4	170.7±11.6	25.0
Lemniscate	26.8±4.4	189.5±13.7	16.0
Racing	53.0±9.2	200.8±14.5	20.0

### G.5.2 Real World Experiments

We assess the performance of different control policies when deployed on a real quadrotor platform. As in the simulation experiments, we execute a set of trajectories and compare tracking performance between the methods presented in Section G.4. We encourage the reader to watch the supplementary video to understand the dynamic nature of these experiments.

The results of the real world experiments are shown in Table G.7. Due to its significant sensitivity to control delays and a communication latency of 60 ms imposed by the real system, the SRT policies could not be deployed. The CTBR policies instead manage to fly unseen maneuvers on the real platform despite the control delay. The LV policies transfer to the real platform as well, but CTBR significantly outperforms on agile trajectories. Compared to the results in the BEM simulator, tracking errors are higher in the real world mainly due to unmodelled effects such as varying battery voltage, imperfect motor thrust mappings, and torque imbalances due to imperfect mass distribution. Throughout the tested trajectories, the CTBR policies reach accelerations of up to 3g and speeds beyond  $45 \text{ km h}^{-1}$ , which outperforms the previous state of the art in learning-based quadrotor control by a factor of 3 in terms of speed.

## G.6 Conclusion

We presented a comparison of learning-based controllers for agile quadrotor flight. We compared policies that specify individual rotor thrusts, collective thrust and bodyrates, and linear velocity commands. While all tested policy types were able to learn a universal flight controller, they differed strongly in terms of peak performance and robustness against dynamics mismatch. We identified that policies producing collective thrust and bodyrates exhibit strong resilience against dynamics mismatch and transfer well between domains while retaining high agility. This work can serve as guideline for future work on learning-based quadrotor control by identifying the control input modality that is best suited for agile flight.

## G.7 Supplementary Material

In the supplementary material we provide implementation details of our quadrotor simulation, training hyperparameters, reference trajectories and the MPC baselines. Additionally, we show a set of additional ablation studies regarding the choice of low level controller, the history length  $H$ , and the reference length  $R$ .

### G.7.1 MPC Baselines

This section gives a brief overview of the MPC baselines implemented in this work. MPC stabilizes a system subject to its dynamics  $\dot{\mathbf{x}} = \mathbf{f}_{dyn}(\mathbf{x}, \mathbf{u})$  along a reference  $\mathbf{x}^*(t), \mathbf{u}^*(t)$ , where  $\mathbf{f}_{dyn}$  is represented by Eq. (G.1), omitting the dynamics of the motors. In each

control update, MPC minimizes a cost  $\mathcal{L}(\mathbf{x}, \mathbf{u})$  as in:

$$\begin{aligned} & \min_{\mathbf{u}} \int \mathcal{L}(\mathbf{x}, \mathbf{u}) & (G.6) \\ \text{subject to} & \quad \dot{\mathbf{x}} = \mathbf{f}_{dyn}(\mathbf{x}, \mathbf{u}) \\ & \quad \mathbf{x}(t_0) = \mathbf{x}_{init} \\ & \quad \mathbf{h}(\mathbf{x}, \mathbf{u}) \leq 0 \end{aligned}$$

where  $\mathbf{x}_0$  denotes the initial condition,  $\mathbf{f}_{dyn}$  implements the system dynamics as equality constraints, and  $\mathbf{h}$  represents inequality constraints, such as input limitations.

For our application, and as most commonly done, we specify the cost to be of quadratic form  $\mathcal{L}(\mathbf{x}, \mathbf{u}) = \|\mathbf{x} - \mathbf{x}^*\|_Q^2 + \|\mathbf{u} - \mathbf{u}^*\|_R^2$  and discretize the system into  $N$  steps over time horizon  $T$  of size  $dt = T/N$ . We account for input limitations by constraining  $0 \leq \mathbf{u} \leq u_{max}$ .

$$\begin{aligned} & \min_{\mathbf{u}} \mathbf{x}_N^T Q \mathbf{x}_N + \sum_{k=0}^N \mathbf{x}_k^T Q \mathbf{x}_k + \mathbf{u}_k^T R \mathbf{u}_k & (G.7) \\ \text{subject to} & \quad \mathbf{x}_{k+1} = \mathbf{f}_{RK4}(\mathbf{x}_k, \mathbf{u}_k, \delta t) \\ & \quad \mathbf{x}_0 = \mathbf{x}_{init} \\ & \quad u_{min} \leq \mathbf{u}_k \leq u_{max} \end{aligned}$$

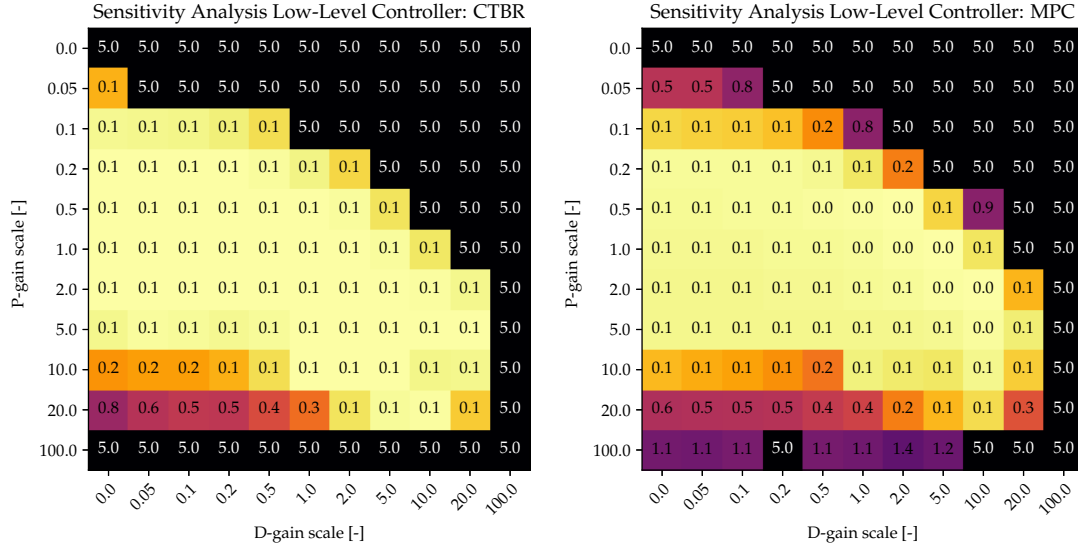
where  $\mathbf{f}_{RK4}$  represents the discretized dynamics  $\mathbf{f}_{dyn}$  using a 4th-order Runge-Kutta scheme. To solve this quadratic optimization problem, we construct it using a multiple shooting scheme and solve it through a sequential quadratic program (SQP) executed in a real-time iteration scheme. All implementations are done using ACADO [138]. Both MPC baselines in this work (MPC-SRT and MPC-CTBR) use the same formulation. While MPC-SRT forwards the predicted single rotor thrusts to the motors, in case of MPC-CTBR the bodyrates and collective thrust from the first predicted state are used as setpoints for the low level controller.

### G.7.2 Ablation Studies

We ablate the performance of our trained policies to investigate the impact of the history length  $H$  and the reference length  $R$ , and analyze the sensitivity to the tuning of the low-level controller.

**Sensitivity to Low-Level Controller.** We investigate the sensitivity of the CTBR control policies to the tuning of the underlying low-level controller. The low-level controller consists of a PD-controller that tracks desired angular rates. The sensitivity analysis is performed by individually scaling P- and D-gains and analyzing the tracking performance. The analysis is performed on the *RaceA* maneuver in the nominal simulation setting. For reference, the sensitivity analysis is performed also for the MPC-CTBR baseline. Figure G.5 shows the results of this sensitivity analysis. Starting from the nominal PD-tuning with a scaling factor of (1.0, 1.0), 121 controller tunings have been tested with scaling factors in the range [0.0, 100.0]. The learned CTBR policies show comparable

## Appendix G. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight



**Figure G.5** – Positional tracking error in meters for varying low-level controllers on *RaceA* for the MPC-CTBR baseline (left) and the learned CTBR policies (right). The low-level PD controller is randomized by individually scaling its P- and D-gains. Tracking errors are clipped at 5m. The learned CTBR policies exhibit a significant robustness against changes in the underlying low-level controller that is comparable to the MPC-CTBR baseline.

robustness against changes in the low-level controller as the MPC-CTBR baseline.

**Influence of Observation History.** We investigate the impact of the history length  $H$  on the tracking performance of our trained policies. All policies are trained using domain randomization and only differ in the length of the history of prior states and actions observed. We train policies for  $H = \{1, 5, 10\}$ . The reference length  $R = 10$  is kept constant.

The results of this ablation study are shown in Table G.8. Policies with access to a history of observations strictly outperform reactive policies for all control input modalities on all tested trajectories. Transitioning from no history (H1) to a medium-sized history (H5) leads to a larger improvement in performance compared to the difference between a long history (H10) and a medium-sized history (H5). Policies operating at a lower abstraction level (SRT, CTBR) show a larger sensitivity to history length than policies operating at a higher abstraction level (LV).

**Influence of Reference Length.** We investigate the impact of the reference length  $R$  on the tracking performance of our trained policies. All policies are trained using domain randomization and only differ in the length of the receding-horizon reference. We train policies for  $R = \{1, 5, 10\}$ . The history length  $H = 10$  is kept constant.

The results of this ablation study are shown in Table G.9. Policies with a longer reference length of  $R = 10$  perform superior compared to policies with only access to a single reference  $R = 1$  or a short reference  $R = 5$ . This trend is consistent across action spaces and is more pronounced for more aggressive maneuvers.

## G.7. Supplementary Material

**Table G.8** – Ablation of the impact of the history length  $H$  on the tracking performance, evaluated in a quadrotor simulator based on blade-element momentum theory. Results report mean and standard deviation for 10 trained policies.

	Hover	RandA	RandB	RandC	RaceA	RaceB	Split-S	RaceC
SRT-H1	crash	crash	crash	crash	crash	crash	crash	crash
CTBR-H1	1.7±0.5	5.4±2.9	5.0±1.2	6.2±1.4	16.4±3.0	46.5±18.6	12.3±3.6	67.0±16.0
LV-H1	7.6±2.0	21.5±6.7	79.2±7.6	133±30	168±55	217±19	138±48	210±35
SRT-H5	0.106	crash	crash	crash	crash	crash	crash	crash
CTBR-H5	1.2±0.4	2.3±0.9	4.4±1.7	4.0±1.5	7.7±2.9	11.5±4.2	7.7±3.2	28.5±17.6
LV-H5	7.2±0.9	19.2±1.7	67.7±9.0	98.3±24.6	155.3±9.8	182.0±24.9	103.3±24.6	196.0±10.7
SRT-H10	11.3±4.5	12.0±4.0	14.4±2.4	17.6±5.9	crash	crash	crash	crash
CTBR-H10	0.6±0.5	1.2±0.5	2.2±0.8	2.6±0.8	5.6±1.7	10.0±4.0	6.9±2.6	14.9±5.5
LV-H10	6.7±2.0	17.8±1.4	57.0±12.0	78.9±13.4	144.0±20.1	161.4±17.0	83.8±9.7	161.7±22.0

**Table G.9** – Ablation of the impact of the reference length  $R$  on the tracking performance, evaluated in a quadrotor simulator based on blade-element momentum theory. Results report mean and standard deviation for 10 trained policies.

	Hover	RandA	RandB	RandC	RaceA	RaceB	Split-S	RaceC
SRT-R1	14.0±2.4	17.7±2.1	crash	crash	crash	crash	crash	crash
CTBR-R1	5.6±2.1	7.3±2.4	7.6±2.0	6.1±2.4	11.8±5.7	33.0±8.1	22.1±6.4	43.3±14.6
LV-R1	10.6±3.4	27.9±9.4	73.3±13.5	101±17	174±22	191±21	97.9±10.6	218±41
SRT-R5	12.3±2.4	15.0±2.9	crash	crash	crash	crash	crash	crash
CTBR-R5	0.9±0.2	1.6±0.3	4.5±1.7	5.0±2.4	10.1±5.2	18.0±7.0	11.7±5.3	23.4±6.0
LV-R5	7.8±1.6	20.3±3.6	62.3±7.5	87.7±12.4	162.0±25.5	178.3±23.9	92.3±12.1	188.9±38.5
SRT-R10	11.3±4.5	12.0±4.0	14.4±2.4	17.6±5.9	crash	crash	crash	crash
CTBR-R10	0.6±0.5	1.2±0.5	2.2±0.8	2.6±0.8	5.6±1.7	10.0±4.0	6.9±2.6	14.9±5.5
LV-R10	6.7±2.0	17.8±1.4	57.0±12.0	78.9±13.4	144.0±20.1	161.4±17.0	83.8±9.7	161.7±22.0

### G.7.3 Tracking Performance

In addition to the tracking errors reported in Section G.5, we provide trajectory plots for the test trajectory *RaceA*. We show plots for both simulation settings evaluated in the experimental section: the nominal simulation and the simulation based on blade-element momentum theory with 20ms delay. Each plot illustrates the reference position and the actual position of the platform. In case of a crash, the rollout is terminated (Figure G.9).

**Nominal Model.** Figures G.6, G.7, G.8 illustrate the positional tracking performance of each policy type on the maneuver *RaceA* in the nominal simulation setting. The corresponding numerical tracking errors can be found in Table G.5. While SRT (Figure G.6) and CTBR (Figure G.7) achieve near-perfect performance, LV (Figure G.8) exhibits significant tracking error, especially in high-acceleration regimes of the trajectory.

**Model Mismatch.** Figures G.9, G.10, G.11 illustrate the performance of each policy type on the maneuver *RaceA* in the model mismatch setting, which uses blade-element momentum theory to model the aerodynamic forces and torques acting on the platform. The corresponding numerical tracking errors can be found in Table G.6. SRT policies are very sensitive to changes in the quadrotor model, leading to a crash already after 5.5s as shown in Figure G.9. In contrast, CTBR policies manage to complete the entire trajectory with similar performance as in the nominal setting (Figure G.10). LV policies are also robust to changes in platform dynamics, with tracking performance comparable to the nominal case. Also in the model mismatch setting, LV policies show large tracking errors, especially in high-acceleration regimes of the trajectory (Figure G.11).

### G.7.4 Reference Trajectories

The control policies are trained on a set of feasible reference trajectories. We generate smooth trajectories in position using two approaches: (i) we generate random circular trajectories of different inclination angles, radii, and speeds; (ii) we generate random position trajectories by combining periodic exponential-sine-squared kernels of different magnitudes and frequencies. Both type of trajectories are extended to full-state quadrotor trajectories by exploiting the differential flatness property of the quadrotor dynamics [78]. In total we generate over 600 trajectories covering speeds from  $0 \text{ m s}^{-1}$  up to  $20 \text{ m s}^{-1}$  and accelerations up to  $35 \text{ m s}^{-2}$ . An illustration of one sample of both trajectory types is provided in Figure G.12 and Figure G.13.

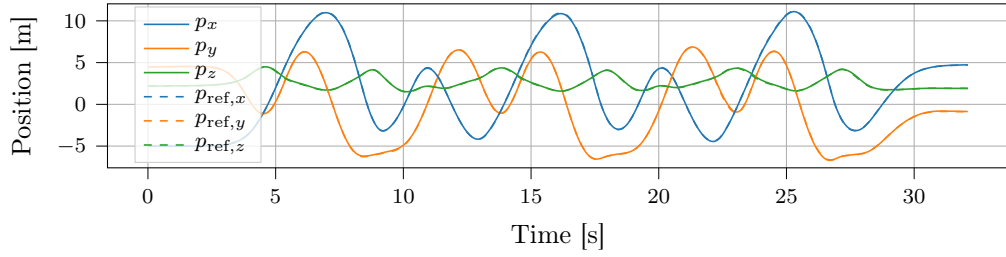


Figure G.6 – Tracking performance of SRT on test trajectory RaceA evaluated with the nominal quadrotor model.

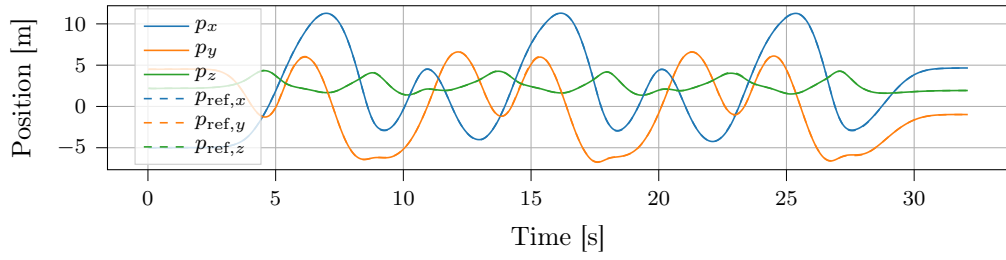


Figure G.7 – Tracking performance of CTBR on test trajectory RaceA evaluated with the nominal quadrotor model.

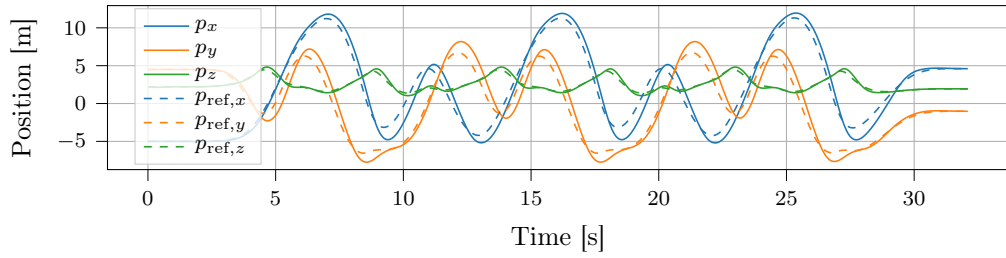


Figure G.8 – Tracking performance of LV on test trajectory RaceA evaluated with the nominal quadrotor model.

# Appendix G. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight

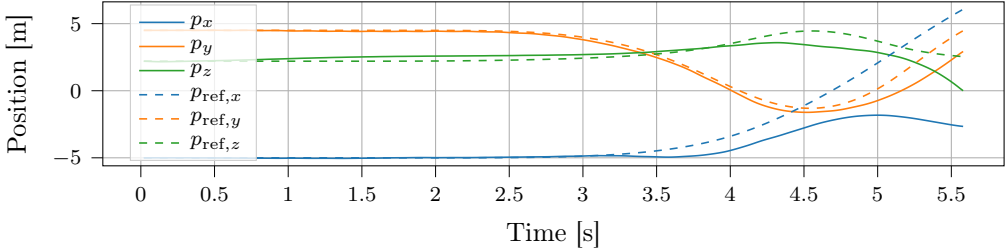


Figure G.9 – Tracking performance of SRT on test trajectory RaceA evaluated in the BEM simulation.

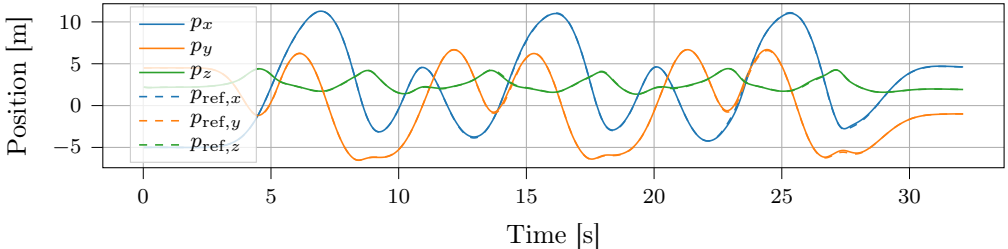


Figure G.10 – Tracking performance of CTBR on trajectory RaceA evaluated in the BEM simulation.

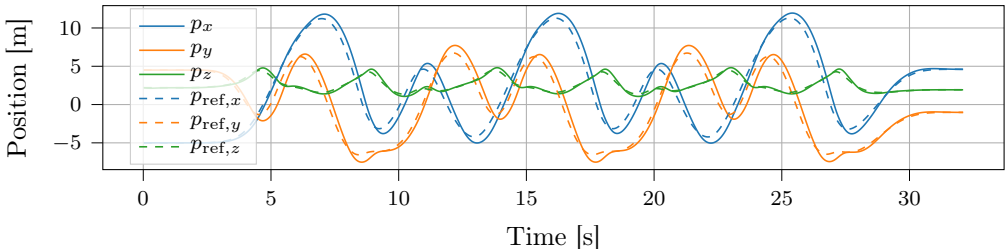
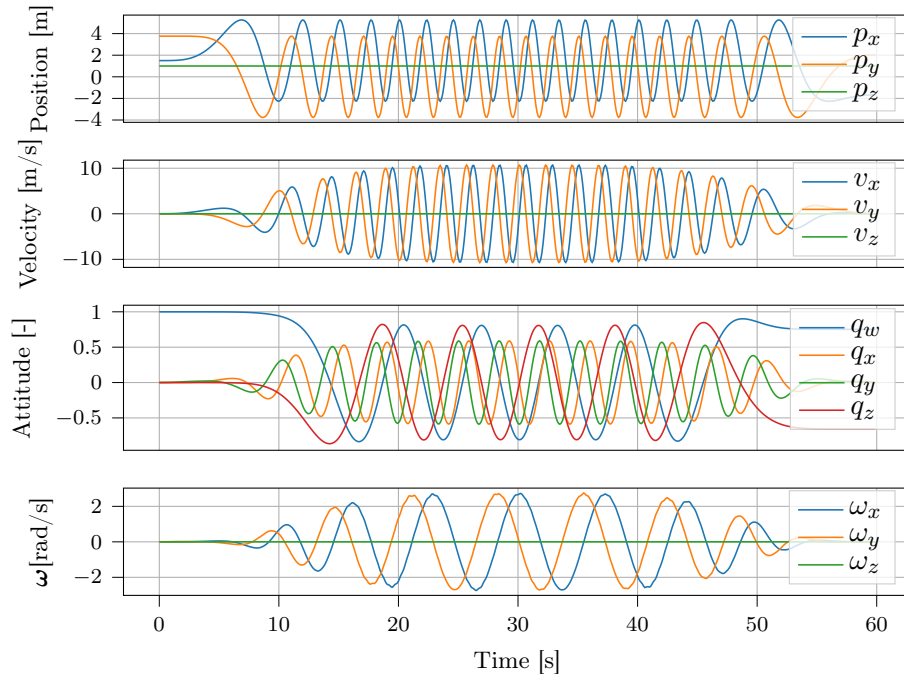
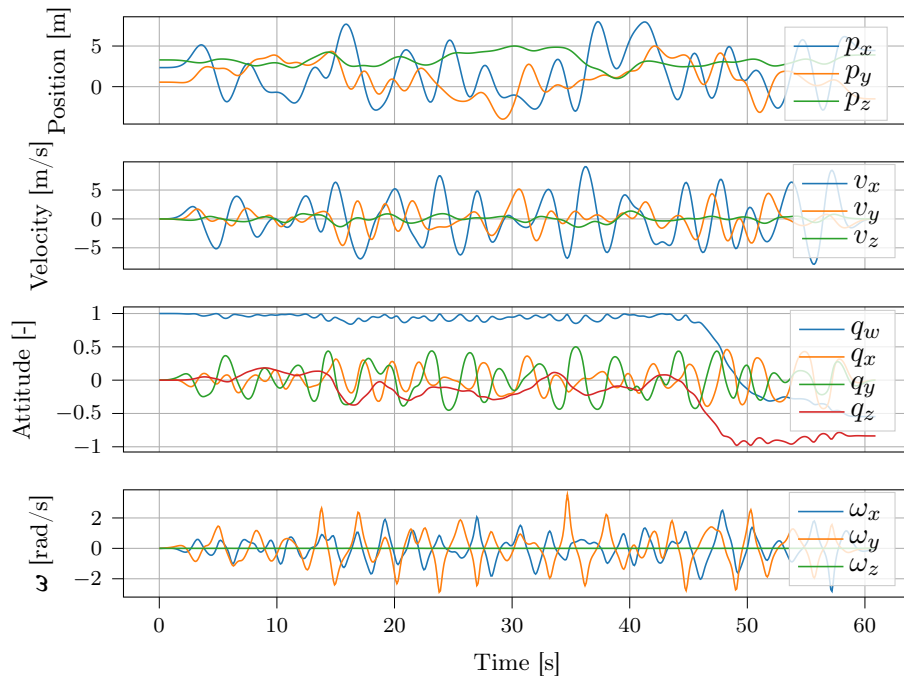


Figure G.11 – Tracking performance of LV on test trajectory RaceA evaluated in the BEM simulation.





**Figure G.12** – Illustration of a sample circular reference trajectory in the training set. Circular trajectories are generated with random inclination angles, radii and speeds.



**Figure G.13** – Illustration of a sample random reference trajectory in the training set. Random trajectories are generated by combining periodic exponential-sine-squared kernels of different magnitudes and frequencies, resulting in a smooth position trajectory. Reference attitude and angular rate is then computed using the differential flatness property of the quadrotor platform.



# H Champion-Level Drone Racing using Deep Reinforcement Learning

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

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Champion-Level Drone Racing using Deep Reinforcement Learning”. In: *Nature* (2023)

# Champion-Level Drone Racing using Deep Reinforcement Learning

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, Davide Scaramuzza

**Abstract** — First-person view (FPV) drone racing is a televised sport in which professional competitors pilot high-speed aircraft through a 3D circuit. Each pilot sees the environment from the perspective of their drone by means of video streamed from an onboard camera. Reaching the level of professional pilots with an autonomous drone is challenging since the robot needs to fly at its physical limits while estimating its speed and location in the circuit exclusively from onboard sensors. Here we introduce Swift, an autonomous system that can race physical vehicles at the level of the human world champions. The system combines deep reinforcement learning in simulation with data collected in the physical world. Swift competed against three human champions, including the world champions of two international leagues, in real-world head-to-head races. Swift won multiple races against each of the human champions and demonstrated the fastest recorded race time. This work represents a milestone for mobile robotics and machine intelligence, which may inspire the deployment of hybrid learning-based solutions in other physical systems.

## H.1 Introduction

Deep Reinforcement Learning (RL) [332] has enabled a number of recent advances in artificial intelligence. Policies trained with deep RL have outperformed humans in complex competitive games including Atari [230, 307, 74], Go [314, 316, 315, 307], chess [315, 307], StarCraft [353], Dota 2 [26], and Gran Turismo [99, 363]. These impressive demonstrations of the capabilities of machine intelligence have primarily been limited to simulation and board game environments, which support policy search in an exact replica of the testing conditions. Overcoming this limitation and demonstrating champion-level performance in physical competitions is a long-standing problem in autonomous mobile robotics and artificial intelligence [100, 326, 361].

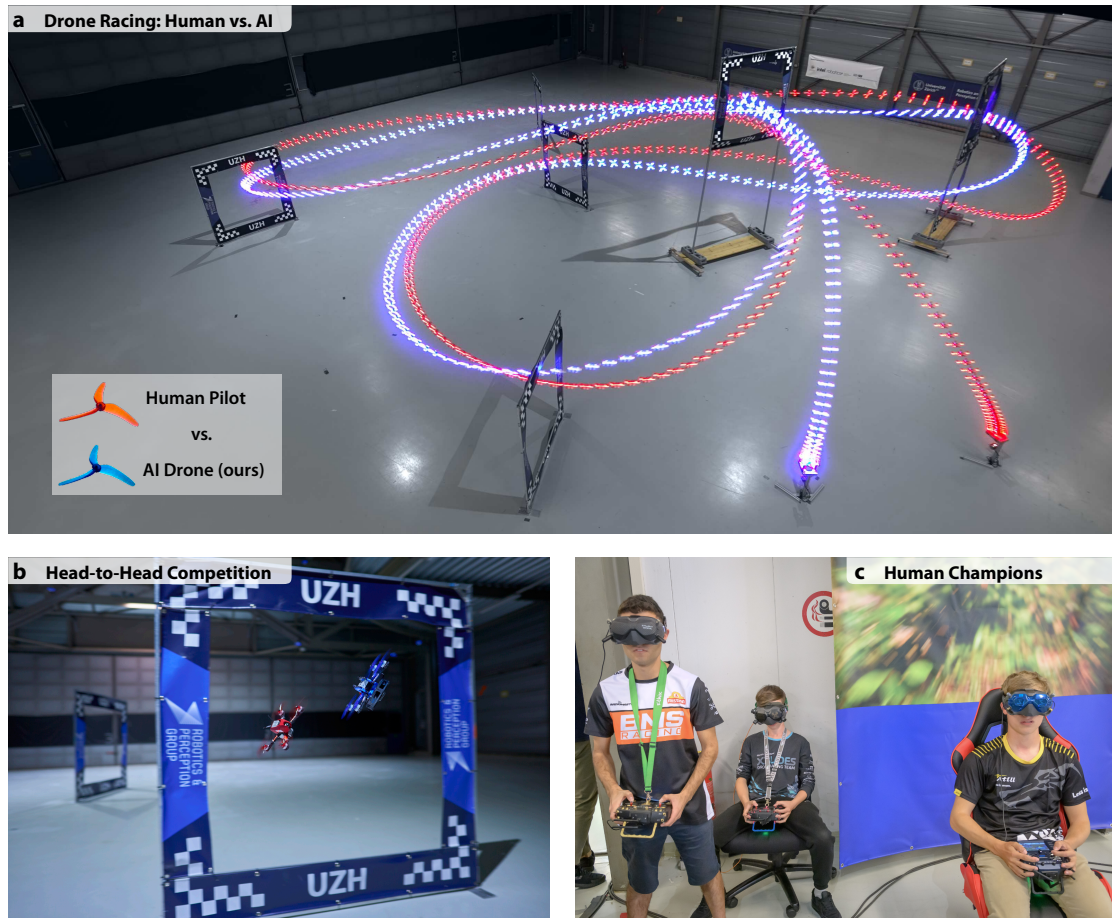
First-person view (FPV) drone racing is a televised sport in which highly trained human pilots push aerial vehicles to their physical limits in high-speed agile maneuvers H.1a. The vehicles used in FPV racing are quadcopters, which are among the most agile machines ever built H.1b. During a race, the vehicles exert forces that surpass their own weight by a factor of five or more, reaching speeds over  $100 \text{ km h}^{-1}$  and accelerations multiple times that of gravity, even in confined spaces. Each vehicle is remotely controlled by a human pilot who wears a headset showing a video stream from an onboard camera, creating an immersive “first-person view” experience H.1c.

Attempts to create autonomous systems that reach the performance of human pilots date back to the first autonomous drone racing competition in 2016 [237]. A series of innovations followed, including the use of deep networks to identify the next gate location [156, 171, 369], transfer of racing policies from simulation to reality [208, 209], and accounting for uncertainty in perception [170, 196]. The 2019 AlphaPilot autonomous drone racing competition showcased some of the best research in the field [7]. However, the first two teams still took twice as long as a professional human pilot to complete the track [93, 355]. More recently, autonomous systems have begun to reach expert human performance [91, 286, 329]. However, these works rely on near-perfect state estimation provided by an external motion capture system. This makes the comparison with human pilots unfair, since humans only have access to onboard observations from the drone.

In this article, we describe Swift, the first autonomous system that can race a quadrotor at the level of human world champions while using only onboard sensors and computation. Swift consists of two key modules: (i) a perception system that translates high-dimensional visual and inertial information into a low-dimensional representation and (ii) a control policy that ingests the low-dimensional representation produced by the perception system and produces control commands.

The control policy is represented by a feedforward neural network and is trained in simulation using model-free on-policy deep reinforcement learning [308]. To bridge discrepancies in sensing and dynamics between simulation and the physical world, we leverage non-parametric empirical noise models estimated from data collected on the physical system. These empirical noise models have proven to be instrumental for successful transfer of the control policy from simulation to reality.

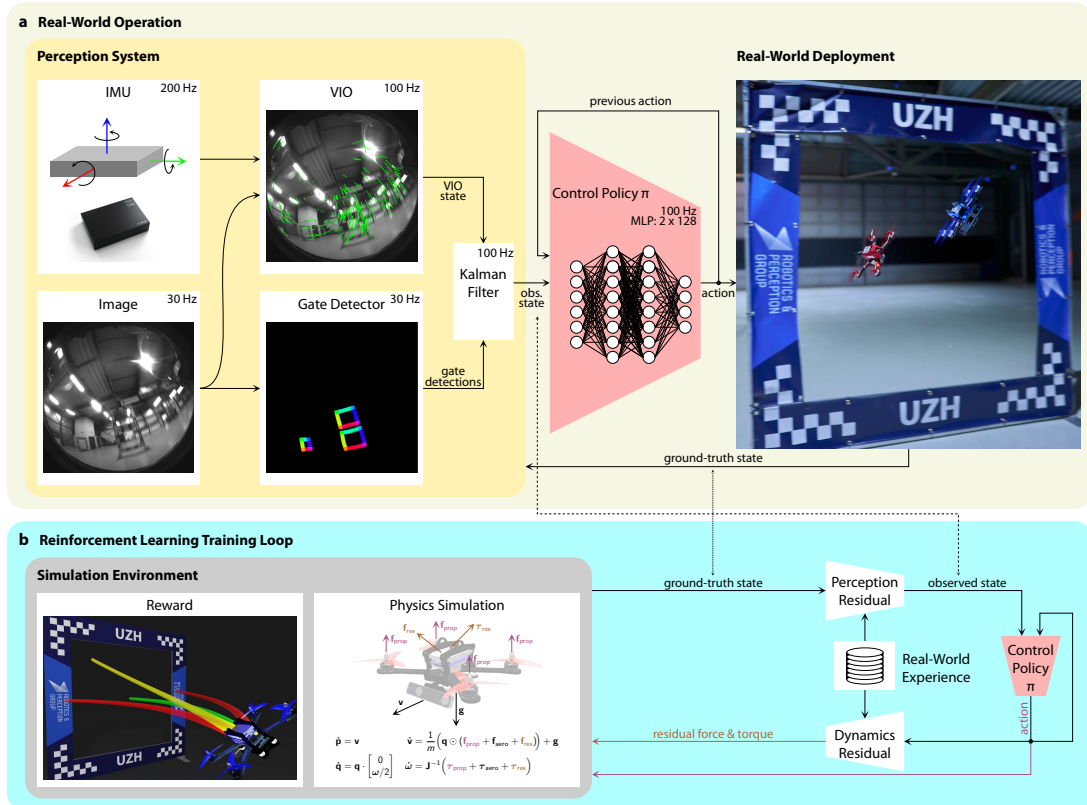
## Appendix H. Champion-Level Drone Racing using Deep Reinforcement Learning



**Figure H.1 – Drone Racing.** **a**, Swift (blue) races head-to-head against Alex Vanover, the 2019 Drone Racing League world champion (red). The track contains seven square gates that must be passed in order in each lap. To win a race, a competitor has to complete three consecutive laps before its opponent. **b**, A close-up view of Swift, illuminated with blue LEDs, and a human-piloted drone, illuminated with red LEDs, during one of the races. The autonomous drones used in this work rely only on onboard sensory measurements, with no support from external infrastructure such as motion capture systems. **c**, From left to right: Thomas Bitmatta, Marvin Schaepper, and Alex Vanover racing their drones through the track. Each pilot wears a headset that displays a video stream transmitted in real time from a camera onboard their aircraft. The headsets provide an immersive “first-person view” experience.

We evaluate Swift on a physical track designed by a professional drone-racing pilot H.1a. The track contains seven square gates arranged in a volume of  $30 \times 30 \times 8$  meters, forming a lap of 75 meters in length. Swift raced this track against three human champions: Alex Vanover, the 2019 Drone Racing League world champion, Thomas Bitmatta, two-time MultiGP International Open World Cup champion, and Marvin Schaepper, three-time Swiss National champion. The quadrotors used by Swift and by the human pilots have the same weight, shape, and propulsion. They are similar to drones used in international competitions.

The human pilots were given one week of practice on the race track. After this week of



**Figure H.2 – The Swift system.** Swift consists of two key modules: a perception system that translates visual and inertial information into a low-dimensional state observation, and a control policy that maps this state observation to control commands. **a**, The perception system consists of a visual-inertial odometry (VIO) module that computes a metric estimate of the drone state from camera images and high-frequency measurements obtained by an inertial measurement unit (IMU). The VIO estimate is coupled with a neural network that detects the corners of racing gates in the image stream. The corner detections are mapped to a 3D pose and fused with the VIO estimate using a Kalman filter. **b**, We leverage model-free on-policy deep reinforcement learning to train the control policy in simulation. During training, the policy maximizes a reward that combines progress towards the center of the next racing gate with a perception objective to keep the next gate in the camera’s field of view. To transfer the racing policy from simulation to the physical world, we augment the simulation with data-driven residual models of the vehicle’s perception and dynamics. These residual models are identified from real-world experience collected on the race track.

practice, each pilot competed against Swift in multiple head-to-head races H.1a,b. In each head-to-head race, two drones (one controlled by a human pilot and one controlled by Swift) start from a podium. The race is set off by an acoustic signal. The first vehicle that completes three full laps through the track, passing all gates in the correct order in each lap, wins the race.

Swift won multiple races against each of the human pilots and achieved the fastest race time recorded during the events. Our work marks the first time that a machine achieved world-champion-level performance in a real-world competitive sport.

### The Swift System

Swift uses a combination of learning-based and traditional algorithms to map onboard sensory readings to control commands. This mapping comprises two parts: (i) an observation policy, which distills high-dimensional visual and inertial information into a task-specific low-dimensional encoding, and (ii) a control policy that transforms the encoding into commands for the drone. A schematic overview of the system is provided in [H.2](#).

The observation policy consists of a visual-inertial estimator [[304](#), [140](#)] that operates in concert with a gate detector [[93](#)], which is a convolutional neural network that detects the racing gates in the onboard images. Detected gates are then used to estimate the drone’s global position and orientation along the race track. This is done by a camera resectioning algorithm [[54](#)] in combination with a map of the track. The estimate of the global pose obtained from the gate detector is then fused with the estimate from the visual-inertial estimator via a Kalman filter, resulting in a more accurate representation of the robot’s state. The control policy, represented by a two-layer perceptron, maps the output of the Kalman filter to control commands for the aircraft. The policy is trained using on-policy model-free deep reinforcement learning [[308](#)] in simulation. During training, the policy maximizes a reward that combines progress towards the next racing gate [[322](#)] with a perception objective that rewards keeping the next gate in the field of view of the camera. Seeing the next gate is rewarded because it increases the accuracy of pose estimation.

Optimizing a policy purely in simulation yields poor performance on physical hardware if the discrepancies between simulation and reality are not mitigated. The discrepancies are caused primarily by two factors: (i) the difference between simulated and real dynamics and (ii) the noisy estimation of the robot’s state by the observation policy when provided with real sensory data. We mitigate these discrepancies by collecting a small amount of data in the real world and using this data to increase the realism of the simulator.

Specifically, we record onboard sensory observations from the robot together with highly accurate pose estimates from a motion capture system while the drone is racing through the track. The recorded data allows to identify the characteristic failure modes of perception and dynamics observed through the race track. These intricacies of failing perception and unmodeled dynamics are environment-, platform-, track-, and sensor-dependent. The perception and dynamics residuals are modeled using Gaussian processes [[357](#)] and k-nearest neighbor regression, respectively. These residual models are integrated into the simulation and the racing policy is fine-tuned in this augmented simulation. This approach is related to the empirical actuator models used for simulation-to-reality transfer by Hwangbo et al. [[143](#)], but further incorporates empirical modeling of the perception system and also accounts for the stochasticity in the estimate of the platform state.

We ablate each component of Swift in controlled experiments reported in the supplementary material. In addition, we compare against recent work that tackles the task of autonomous drone racing with traditional methods including trajectory planning and model predictive control. While such approaches achieve comparable or even superior performance to our approach in idealized conditions, such as simplified dynamics and



perfect knowledge of the robot’s state, their performance collapses when their assumptions are violated. We find that approaches that rely on precomputed paths [91, 286] are particularly sensitive to noisy perception and dynamics. No traditional method has achieved competitive lap times to Swift or human world champions, even when provided with highly accurate state estimation from a motion capture system. Detailed analysis is provided in the supplementary material.

## H.2 Results

The drone races take place on a track designed by an external world-class FPV pilot. The track features characteristic and challenging maneuvers such as a *Split-S*<sup>1</sup> (Fig. H.1a (top-right corner) and H.4d). Pilots are allowed to continue racing even after a crash, provided that their vehicle is still able to fly. If both drones crash and cannot complete the track, the drone that proceeded farther along the track wins.

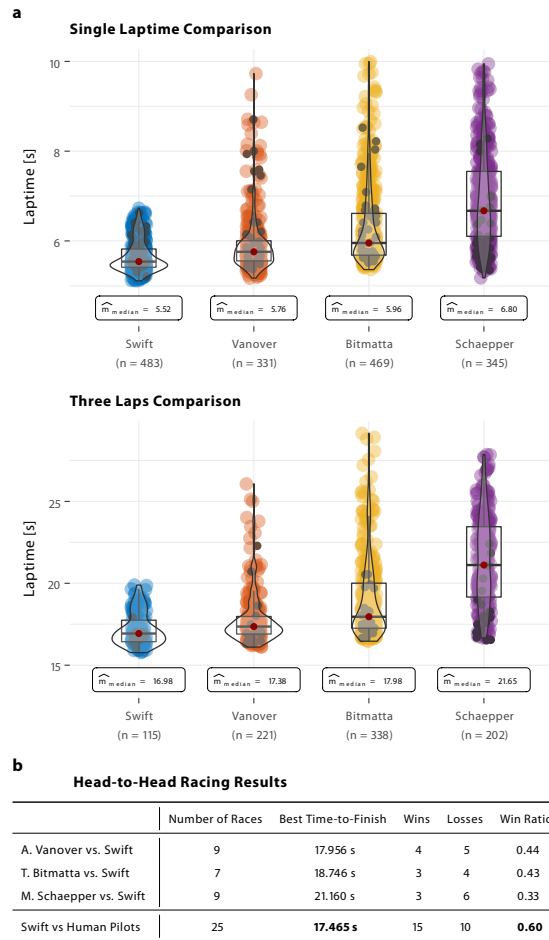
As shown in Fig. H.3b, Swift wins 5 out of 9 races against A. Vanover, 4 out of 7 races against T. Bitmatta, and 6 out of 9 races against M. Schaepper. Overall, Swift wins the majority of races against each human pilot. Swift also achieves the fastest race time recorded, with a lead of half a second over the best time clocked by a human pilot (A. Vanover).

Figure H.4 provides an analysis of the fastest lap flown by Swift and each human pilot. While Swift is globally faster than all human pilots, it is not faster on all individual segments of the track (Fig. H.4f). Swift is consistently faster at the start and in tight turns such as the Split-S. At the start, Swift has a lower reaction time, taking off from the podium on average 120ms before human pilots. In addition, it accelerates faster and reaches higher speeds going into the first gate (Fig. H.4f, segment 1). In sharp turns, as shown in Fig. H.4c,d, Swift finds tighter maneuvers. One hypothesis is that Swift optimizes trajectories on a longer time scale than human pilots. It is known that model-free RL can optimize long-term rewards via a value function [142]. Conversely, human pilots plan their motion on a shorter time scale, up to one gate into the future [270]. This is apparent for example in the Split-S (Fig. H.4b,d), where human pilots are faster in the beginning and at the end of the maneuver, but slower overall (Fig. H.4f, segment 3). In addition, human pilots orient the aircraft to face the next gate earlier than Swift does (Fig. H.4c,d). We hypothesize that human pilots are accustomed to keeping the upcoming gate in view, whereas Swift has learned to execute some maneuvers while relying on other cues, such as inertial data and visual odometry against features in the surrounding environments. Overall, averaged over the entire track, the autonomous drone achieves the highest average speed, finds the shortest racing line, and manages to maintain the aircraft closer to its actuation limits throughout the race as indicated by the average thrust and power drawn (Fig. H.4f).

We also compare the performance of Swift and the human champions in time trials (Fig. H.3a). In a time trial, a single pilot races the track, with the number of laps left to

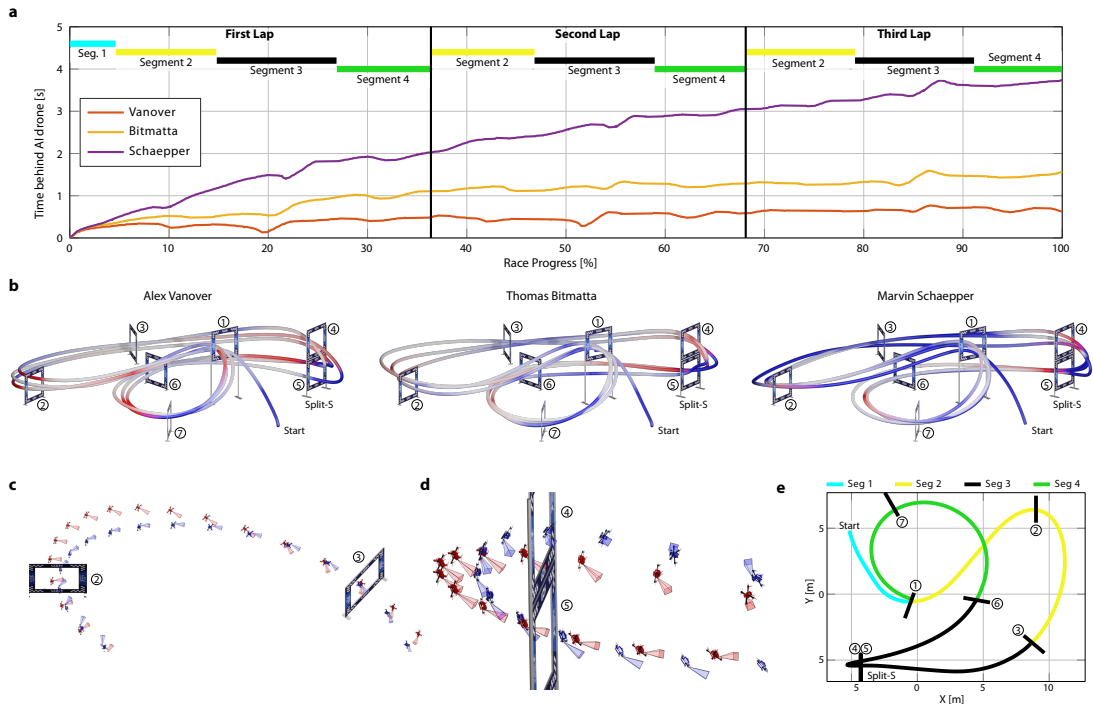
<sup>1</sup>To execute a split S, the pilot half-rolls their aircraft inverted and executes a descending half-loop, resulting in level flight in the opposite direction at a lower altitude.

## Appendix H. Champion-Level Drone Racing using Deep Reinforcement Learning



**Figure H.3 – Results.** **a**, Laptime results. We compare Swift against the human pilots in time-trial races. Lap times indicate best single-lap times and best average times achieved in a heat of 3 consecutive laps. The reported statistics are computed over a dataset recorded during one week on the race track, which corresponds to 483 (115) datapoints for Swift, 331 (221) for A. Vanover, 469 (338) for T. Bitmatta, and 345 (202) for M. Schaepper. The first number is the number of single laps, the second is the number of three consecutive laps. The dark points in each distribution correspond to laps flown in race conditions. **b**, Head-to-head results. We report the number of head-to-head races flown by each pilot, the number of wins and losses, as well as the win ratio.

the pilot’s discretion. We accumulate time trial data from the practice week and the races, including training runs (Fig. H.3a, colored) and laps flown in race conditions (Fig. H.3a, black). For each contestant, we use more than 300 laps for computing statistics. The autonomous drone more consistently pushes for fast lap times, exhibiting lower mean and variance. Conversely, human pilots decide whether to push for speed on a lap-by-lap basis, yielding higher mean and variance in lap times, both during training and in the races. The ability to adapt the flight strategy allows human pilots to maintain a slower pace if they identify that they have a clear lead, so as to reduce the risk of a crash. The autonomous drone is unaware of its opponent and pushes for fastest expected completion time no matter what, potentially risking too much when in the lead and too little when



**Figure H.4 – Analysis.** **a**, Comparison of the fastest race of each pilot, illustrated by the time behind Swift. The time difference from the autonomous drone is computed as the time since it passed the same position on the track. While Swift is globally faster than all human pilots, it is not necessarily faster on all individual segments of the track. **b**, Visualization of where the human pilots are faster (red) and slower (blue) compared to the autonomous drone. Swift is consistently faster at the start and in tight turns such as the Split-S. **c**, Analysis of the maneuver after gate 2. (Swift in blue, Vanover in red.) Swift gains time against human pilots in this segment as it executes a tighter turn while maintaining comparable speed. **d**, Analysis of the Split-S maneuver. (Swift in blue, Vanover in red.) The Split-S is the most challenging segment in the race track, requiring a carefully coordinated roll and pitch motion that yields a descending half-loop through the two gates. Swift gains time against human pilots on this segment as it executes a tighter turn with less overshoot. **e**, Illustration of track segments used for analysis. Segment 1 is traversed once at the start, while segments 2-4 are traversed in each lap (three times over the course of a race). **f**, Comparison of the average speed, power, thrust, time, and distance traveled for each pilot during the fastest flown race. Best numbers are indicated in bold.

trailing behind [324].

## Discussion

First-person view drone racing requires real-time decision making based on noisy and incomplete sensory input from the physical environment. We have presented an autonomous physical system that achieves champion-level performance in this sport, reaching and at times exceeding the performance of human world champions. Our system has certain structural advantages over the human pilots. First, it leverages inertial data from an onboard inertial measurement unit [304]. This is akin to the human vestibular system [60],

## Appendix H. Champion-Level Drone Racing using Deep Reinforcement Learning

---

which is not leveraged by the human pilots because they are not physically in the aircraft and do not feel the accelerations acting on it. Second, our system benefits from lower sensorimotor latency (40 ms for Swift versus an average of 220 ms for expert human pilots [270]). On the other hand, the limited refresh rate of the camera used by Swift (30Hz) can be considered a structural advantage for human pilots, whose cameras' refresh rate is four times as fast (120Hz), improving their reaction time [179].

Human pilots are impressively robust: they can crash at full speed, and, if the hardware still functions, carry on flying and complete the track. Swift was not trained to recover after a crash. Human pilots are also robust to changes in environmental conditions, such as illumination, which can dramatically alter the appearance of the track. In contrast, Swift's perception system assumes that the appearance of the environment is consistent with what was observed during training. If this assumption fails, the system can fail. Robustness to appearance changes can be provided by training the gate detector and the residual observation model in a diverse set of conditions.

Notwithstanding the remaining limitations and the work ahead, the attainment by an autonomous mobile robot of world-champion-level performance in a popular physical sport is a milestone for robotics and machine intelligence. This work may inspire the deployment of hybrid learning-based solutions in other physical systems, such as autonomous ground vehicles, aircraft, and personal robots across a broad range of applications.

### H.3 Methods

#### H.3.1 Quadrotor Simulation

**Quadrotor dynamics.** To enable large-scale training, we use high-fidelity simulation of the quadrotor dynamics. This section briefly explains the simulation. The dynamics of the vehicle can be written as

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}}_{\mathcal{WB}} \\ \dot{\mathbf{q}}_{\mathcal{WB}} \\ \dot{\mathbf{v}}_{\mathcal{W}} \\ \dot{\boldsymbol{\omega}}_{\mathcal{B}} \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} = \begin{bmatrix} \mathbf{v}_{\mathcal{W}} \\ \mathbf{q}_{\mathcal{WB}} \cdot \begin{bmatrix} 0 \\ \boldsymbol{\omega}_{\mathcal{B}}/2 \end{bmatrix} \\ \frac{1}{m} \left( \mathbf{q}_{\mathcal{WB}} \odot (\mathbf{f}_{\text{prop}} + \mathbf{f}_{\text{aero}}) \right) + \mathbf{g}_{\mathcal{W}} \\ \mathbf{J}^{-1} \left( \boldsymbol{\tau}_{\text{prop}} + \boldsymbol{\tau}_{\text{mot}} + \boldsymbol{\tau}_{\text{aero}} + \boldsymbol{\tau}_{\text{iner}} \right) \\ \frac{1}{k_{\text{mot}}} \left( \boldsymbol{\Omega}_{\text{des}} - \boldsymbol{\Omega} \right) \end{bmatrix}, \quad (\text{H.1})$$

where  $\odot$  represents quaternion rotation,  $\mathbf{p}_{\mathcal{WB}}$ ,  $\mathbf{q}_{\mathcal{WB}}$ ,  $\mathbf{v}_{\mathcal{W}}$ , and  $\boldsymbol{\omega}_{\mathcal{B}}$  denote the position, attitude quaternion, inertial velocity, and bodyrates of the quadcopter, respectively. The motor time constant is  $k_{\text{mot}}$  and the motor speeds  $\boldsymbol{\Omega}$  and  $\boldsymbol{\Omega}_{\text{des}}$  are the actual and commanded motor speeds, respectively. The matrix  $\mathbf{J}$  is the quadcopter's inertia and  $\mathbf{g}_{\mathcal{W}}$  denotes the gravity vector. Two forces act on the quadrotor: the lift force  $\mathbf{f}_{\text{prop}}$  generated by the propellers and an aerodynamic force  $\mathbf{f}_{\text{aero}}$  that aggregates all other forces, such as aerodynamic drag, dynamic lift, and induced drag. The torque is modeled as a sum

of four components: the torque generated by the individual propeller thrusts  $\boldsymbol{\tau}_{\text{prop}}$ , the yaw-torque  $\boldsymbol{\tau}_{\text{mot}}$  generated by a change in motor speed, an aerodynamic torque that accounts for various aerodynamic effects such as blade flapping, and an inertial term  $\boldsymbol{\tau}_{\text{iner}}$ . The individual components are given as

$$\boldsymbol{f}_{\text{prop}} = \sum_i \boldsymbol{f}_i, \quad \boldsymbol{\tau}_{\text{prop}} = \sum_i \boldsymbol{\tau}_i + \boldsymbol{r}_{\text{P},i} \times \boldsymbol{f}_i, \quad (\text{H.2})$$

$$\boldsymbol{\tau}_{\text{mot}} = J_{\text{m+p}} \sum_i \boldsymbol{\zeta}_i \dot{\Omega}_i, \quad \boldsymbol{\tau}_{\text{iner}} = -\boldsymbol{\omega}_{\text{B}} \times \boldsymbol{J} \boldsymbol{\omega}_{\text{B}} \quad (\text{H.3})$$

where  $\boldsymbol{r}_{\text{P},i}$  is the location of propeller  $i$ , expressed in the body frame, and  $\boldsymbol{f}_i, \boldsymbol{\tau}_i$  are the forces and torques generated by the  $i$ -th propeller. The axis of rotation of the  $i$ -th motor is denoted by  $\boldsymbol{\zeta}_i$ , the combined inertia of motor and propeller is  $J_{\text{m+p}}$ , and the derivative of the  $i$ -th motor speed is  $\dot{\Omega}_i$ . The individual propellers are modelled using a commonly used quadratic model, which assumes that the lift force and drag torque are proportional to the square of the propeller speed  $\Omega_i$ :

$$\boldsymbol{f}_i(\Omega) = [0 \quad 0 \quad c_l \cdot \Omega^2]^\top, \quad \boldsymbol{\tau}_i(\Omega) = [0 \quad 0 \quad c_d \cdot \Omega^2]^\top \quad (\text{H.4})$$

where  $c_l$  and  $c_d$  denote the propeller lift and drag coefficients, respectively.

**Aerodynamic forces and torques.** The aerodynamic forces and torques are hard to model with a first-principles approach. We thus use a data-driven model [23]. To maintain the low computational complexity required for large-scale RL training, a graybox polynomial model is used rather than a neural network. The aerodynamic effects are assumed to primarily depend on the velocity  $\boldsymbol{v}_{\text{B}}$  (in body frame) and the average squared motor speed  $\overline{\Omega^2}$ . Based on insights from the underlying physical processes, linear and quadratic combinations of the individual terms are selected. For readability, the coefficients multiplying each summand have been omitted:

$$\begin{aligned} f_x &\sim v_x + v_x |v_x| + \overline{\Omega^2} + v_x \overline{\Omega^2} \\ f_y &\sim v_y + v_y |v_y| + \overline{\Omega^2} + v_y \overline{\Omega^2} \\ f_z &\sim v_z + v_z |v_z| + v_{xy} + v_{xy}^2 + v_{xy} \overline{\Omega^2} + v_z \overline{\Omega^2} + v_{xy} v_z \overline{\Omega^2} \\ \tau_x &\sim v_y + v_y |v_y| + \overline{\Omega^2} + v_y \overline{\Omega^2} + v_y |v_y| \overline{\Omega^2} \\ \tau_y &\sim v_x + v_x |v_x| + \overline{\Omega^2} + v_x \overline{\Omega^2} + v_x |v_x| \overline{\Omega^2} \\ \tau_z &\sim v_x + v_y \end{aligned}$$

The respective coefficients are then identified from real-world flight data, where motion capture is used to provide ground-truth forces and torque measurements. We use data from the race track, allowing the dynamics model to fit the track. This is akin to the human pilots' training for days or weeks prior to the race on the specific track that they will be racing. In our case, the human pilots are given a week of practice on the same track ahead of the competition.

## Appendix H. Champion-Level Drone Racing using Deep Reinforcement Learning

---

**Betaflight low-level controller.** To control the quadrotor, the neural network outputs collective thrust and bodyrates. This control signal is known to combine high agility with good robustness to simulation-to-reality transfer [169]. The predicted collective thrust and bodyrates are then processed by an onboard low-level controller that computes individual motor commands. On the physical vehicle, this low-level controller is implemented using the open-source Betaflight firmware [340]. In simulation, we use an accurate model of this low-level controller.

**Battery model.** The action produced by the neural network policy specifies an acceleration in the body-z direction. The low-level controller, however, does not perform closed-loop control on the body-z acceleration, but instead converts the thrust command into a (pulse-width modulation) PWM signal for the motors. For a given PWM command, the motor speed is a function of the battery voltage. Therefore, the motor thrust depends on the battery voltage. Our simulation thus models the battery voltage as well. We use a graybox battery model [22] that simulates the voltage based on instantaneous power consumption  $P_{\text{mot}}$ :

$$P_{\text{mot}} = \frac{c_d \Omega^3}{\eta} \tag{H.5}$$

The battery model then simulates the battery voltage based on this power demand. We use a polynomial mapping between the commanded collective thrust, the battery voltage, and the steady-state motor speed  $\Omega_{\text{des}}$  that has been identified from experimental data. Together with the model of the low-level controller, this enables the simulator to correctly translate an action in the form of collective thrust and bodyrates to desired motor speeds  $\Omega_{\text{des}}$  in eq. (H.1).

### Policy Training

We train deep neural control policies that directly map observations  $\mathbf{o}_t$  in the form of platform state and next gate observation to control actions  $\mathbf{u}_t$  in the form of mass-normalized collective thrust and bodyrates [169]. The control policies are trained using model-free reinforcement learning in simulation.

**Training algorithm.** Training is performed using proximal policy optimization [308]. This actor-critic approach requires jointly optimizing two neural networks during training: the policy network, which maps observations to actions, and the value network, which serves as the “critic” and evaluates actions taken by the policy. After training, only the policy network is deployed on the robot.

**Observations, actions, and rewards.** An observation  $\mathbf{o}_t \in \mathbb{R}^{31}$  obtained from the environment at time  $t$  consists of (i) an estimate of the current robot state, (ii) the relative pose of the next gate to be passed on the track layout, and (iii) the action applied in the

previous step. Specifically, the estimate of the robot state contains the position of the platform, its velocity, and attitude represented by a rotation matrix. Even though the simulation uses quaternions internally, we use a rotation matrix to represent attitude to avoid ambiguities [376]. The relative pose of the next gate is encoded by providing the relative position of the four gate corners with respect to the vehicle. All observations are normalized before being passed to the network. Since the value network is only used during training time, it can access privileged information about the environment that is not accessible to the policy [273]. This privileged information is concatenated with other inputs to the policy network and contains the exact position, orientation, and velocity of the robot.

For each observation  $\mathbf{o}_t$ , the policy network produces an action  $\mathbf{a}_t \in \mathbb{R}^4$  in the form of desired mass-normalized collective thrust and bodyrates.

We use a dense shaped reward formulation to learn the task of perception-aware autonomous drone racing. The reward  $r_t$  at timestep  $t$  is given by

$$r_t = r_t^{\text{prog}} + r_t^{\text{perc}} + r_t^{\text{cmd}} + r_t^{\text{crash}} \quad (\text{H.6})$$

where  $r^{\text{prog}}$  rewards progress towards the next gate [322],  $r^{\text{perc}}$  encodes perception awareness by adjusting the vehicle’s attitude such that the optical axis of the camera points towards the next gate’s center,  $r^{\text{cmd}}$  rewards smooth actions, and  $r^{\text{crash}}$  is a binary penalty that is only active when colliding with a gate or when the platform leaves a pre-defined bounding box. If  $r_{\text{crash}}$  is triggered, the training episode ends.

Specifically, the reward terms are

$$r_t^{\text{prog}} = \lambda_1 [d_{t-1}^{\text{Gate}} - d_t^{\text{Gate}}] \quad (\text{H.7})$$

$$r_t^{\text{perc}} = \lambda_2 \exp[\lambda_3 \cdot \delta_{\text{cam}}^4] \quad (\text{H.7})$$

$$r_t^{\text{cmd}} = \lambda_4 \mathbf{a}_t^\omega + \lambda_5 \|\mathbf{a}_t - \mathbf{a}_{t-1}\|^2 \quad (\text{H.8})$$

$$r_t^{\text{crash}} = \begin{cases} -5.0, & \text{if } p_z < 0 \text{ or in collision with gate.} \\ 0, & \text{otherwise} \end{cases}$$

where  $d_t^{\text{Gate}}$  denotes the distance from the vehicle’s center of mass to the center of the next gate at timestep  $t$ ,  $\delta_{\text{cam}}$  represents the angle between the optical axis of the camera and the center of the next gate, and  $\mathbf{a}_t^\omega$  are the commanded bodyrates. The hyperparameters  $\lambda_1, \dots, \lambda_5$  balance different terms (Table H.1).

**Training details.** Data collection is performed by simulating 100 agents in parallel that interact with the environment in episodes of 1500 steps. At each environment reset, every agent is initialized at a random gate on the track, with bounded perturbation around a state previously observed when passing this gate. In contrast to prior work [234, 12, 169], we do not perform randomization of the platform dynamics at training time. Instead we perform fine-tuning based on real-world data. The training environment is implemented using TensorFlow Agents [116]. The policy network and the value network

## Appendix H. Champion-Level Drone Racing using Deep Reinforcement Learning

---

are both represented by two-layer perceptrons with 128 nodes in each layer. Network parameters are optimized using the Adam optimizer with learning rate  $3e-4$  for both the policy network and the value network.

Policies are trained for a total of  $1e8$  environment interactions, which takes 50 minutes on a workstation (i9 12900K, RTX 3090, 32RAM DDR5). Fine-tuning is performed for  $2e7$  environment interactions.

Table H.1 – Training hyperparameters.

Hyperparameter	Value
$\gamma$ (discount factor)	0.99
$\varepsilon$ (importance ratio clipping)	0.2
$\lambda_1$	1.0
$\lambda_2$	0.02
$\lambda_3$	-10.0
$\lambda_4$	-2e-4
$\lambda_5$	-1e-4

### Residual Model Identification

We perform fine-tuning of the original policy based on a small amount of data collected in the real world. Specifically, we collect three full rollouts in the real world, corresponding to approximately 50s of flight time. We fine-tune the policy by identifying *residual observations* and *residual dynamics*, which are then used for training in simulation.

**Residual observation model.** Navigating at high speeds results in substantial motion blur, which can lead to a loss of tracked visual features and severe drift in linear odometry estimates. We fine-tune policies with an odometry model that is identified from only a handful of trials recorded in the real world. To model the drift in odometry, we use Gaussian processes [357], as they allow fitting a posterior distribution of odometry perturbations, from which we can sample temporally consistent realizations.

Specifically, the Gaussian process model fits residual position, velocity, and attitude as a function of the ground-truth robot state. The observation residuals are identified by comparing the observed VIO estimates during a real-world rollout with the ground-truth platform states, which are obtained from an external motion tracking system.

We treat each dimension of the observation separately, effectively fitting a set of 9 one-dimensional Gaussian processes to the observation residuals. We use a mixture of Radial Basis Function (RBF) kernels

$$\kappa(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{z}_i - \mathbf{z}_j)^\top \mathbf{L}^{-2}(\mathbf{z}_i - \mathbf{z}_j)\right) + \sigma_n^2 \quad (\text{H.9})$$

where  $\mathbf{L}$  is the diagonal length scale matrix and  $\sigma_f, \sigma_n$  represent the data and prior noise



variance, respectively, and  $\mathbf{z}_i, \mathbf{z}_j$  represent data features. The kernel hyperparameters are optimized by maximizing the log-marginal-likelihood. After kernel hyperparameter optimization, we sample new realizations from the posterior distribution that are then used during fine-tuning of the policy. Figure H.5 illustrates the residual observations in position, velocity and attitude in real-world rollouts, as well as 100 sampled realizations from the Gaussian process model.

**Residual dynamics model.** We use a residual model to complement the simulated robot dynamics [346]. Specifically, we identify residual accelerations as a function of the platform state  $\mathbf{s}$  and the commanded mass-normalized collective thrust  $c$ :

$$\mathbf{a}_{\text{res}} = KNN(\mathbf{s}, c) \tag{H.10}$$

We use k-nearest neighbor regression with  $k = 5$ . The size of the dataset used for residual dynamics model identification depends on the track layout, and ranges between 800 and 1000 samples for the track layout used in this work.

### VIO Drift Estimation

The odometry estimates from the VIO pipeline [147] exhibit substantial drift during high-speed flight. We use gate detection to stabilize the pose estimates produced by VIO. The gate detector outputs the coordinates of all visible gates' corners. A relative pose is first estimated for all predicted gates using infinitesimal plane-based pose estimation (IPPE) [54]. Given this relative pose estimate, each gate observation is assigned to the closest gate in the known track layout, thus yielding a pose estimate for the drone.

Due to the low frequency of the gate detections and the high quality of the VIO orientation estimate, we only refine the translational components of the VIO measurements. We estimate the drift of the VIO pipeline using a Kalman filter that estimates the static drift  $\mathbf{p}_d$  (position offset) and its derivative, the drift velocity  $\mathbf{v}_d$ . The filter state  $\mathbf{x}$  is given by  $\mathbf{x} = [\mathbf{p}_d^\top, \mathbf{v}_d^\top]^\top \in \mathbb{R}^6$ .

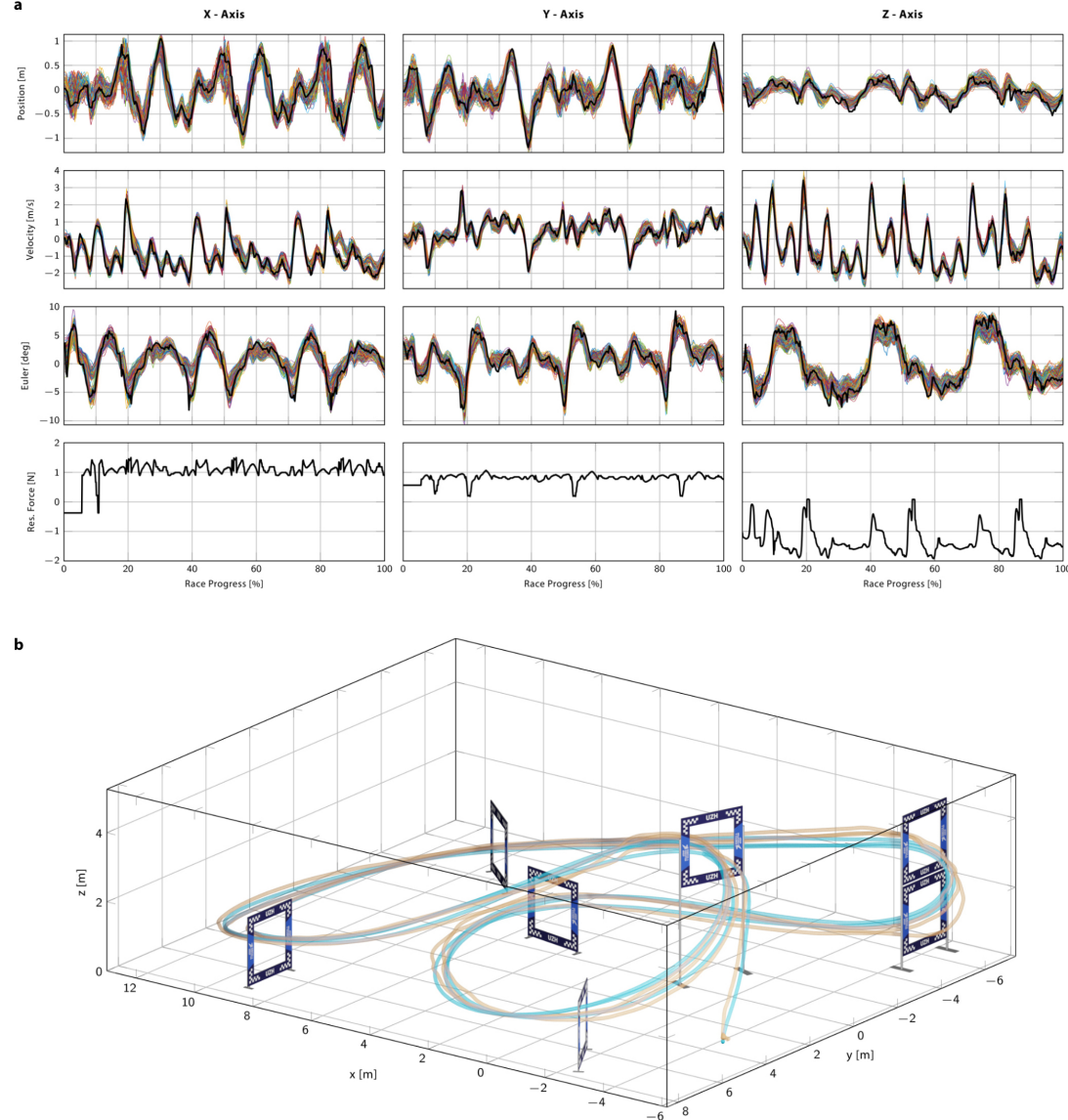
The state  $\mathbf{x}$  and covariance  $\mathbf{P}$  updates are given by:

$$\mathbf{x}_{k+1} = \mathbf{F}\mathbf{x}_k, \quad \mathbf{P}_{k+1} = \mathbf{F}\mathbf{P}_k\mathbf{F}^\top + \mathbf{Q}, \tag{H.11}$$

$$\mathbf{F} = \begin{bmatrix} \mathbb{I}^{3 \times 3} & dt \mathbb{I}^{3 \times 3} \\ \mathbf{0}^{3 \times 3} & \mathbb{I}^{3 \times 3} \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} \sigma_{\text{pos}} \mathbb{I}^{3 \times 3} & \mathbf{0}^{3 \times 3} \\ \mathbf{0}^{3 \times 3} & \sigma_{\text{vel}} \mathbb{I}^{3 \times 3} \end{bmatrix}. \tag{H.12}$$

Based on measurements, the process noise is set to  $\sigma_{\text{pos}}=0.05$  and  $\sigma_{\text{vel}}=0.1$ . The filter state and covariance are initialized to zero. For each measurement  $\mathbf{z}_k$  (pose estimate from a gate detection), the predicted VIO drift  $\mathbf{x}_k^-$  is corrected to the estimate  $\mathbf{x}_k^+$  according

# Appendix H. Champion-Level Drone Racing using Deep Reinforcement Learning



**Figure H.5** – **a**, Visualization of the residual observation model and the residual dynamics model identified from real-world data. Black curves depict the residual observed in the real world, colored lines show 100 sampled realizations of the residual observation model. **b**, Predicted residual observation for a simulated rollout.

to the Kalman filter equations:

$$\begin{aligned}
\mathbf{K}_k &= \mathbf{P}_k^- \mathbf{H}_k^\top (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^\top + \mathbf{R})^{-1}, \\
\mathbf{x}_k^+ &= \mathbf{x}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}(\mathbf{x}_k^-)), \\
\mathbf{P}_k^+ &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^-,
\end{aligned} \tag{H.13}$$

where  $\mathbf{K}_k$  is the Kalman gain,  $\mathbf{R}$  is the measurement covariance, and  $\mathbf{H}_k$  is the measurement matrix. The main source of measurement error is the uncertainty in the gate-corner detection of the network. This error in the image plane results in a pose error when IPPE is applied. We opted for a sampling-based approach to estimate the pose error from the known average gate-corner detection uncertainty. For each gate, the IPPE algorithm is applied to the nominal gate observation as well as to 20 perturbed gate-corner estimates. The resulting distribution of pose estimates is then used to approximate the measurement covariance  $\mathbf{R}$  of the gate observation.

### Simulation Results

Reaching champion-level performance in autonomous drone racing requires overcoming two challenges: imperfect perception and incomplete models of the system’s dynamics. In controlled experiments in simulation, we assess the robustness of our approach to both of these challenges. To this end, we evaluate performance in a racing task when deployed in four different settings: In setting (i), we simulate a simplistic quadrotor model with access to ground-truth state observations. In setting (ii), we replace the ground-truth state observations with noisy observations identified from real-world flights. Settings (iii) and (iv) share the observation models with the previous two settings, respectively, but replace the simplistic dynamics model with more accurate aerodynamical simulation [23]. These four settings allow controlled assessment of the sensitivity of the approach to changes in the dynamics and the observation fidelity.

In all four settings, we benchmark our approach against the following baselines: *Zero-Shot*, *Domain Randomization*, and *Time-Optimal*. The *Zero-Shot* baseline represents a learning-based racing policy [322] trained using model-free RL that is deployed zero-shot from the training domain to the test domain. The training domain of the policy is equal to experimental setting (i), i.e. idealized dynamics and ground-truth observations. *Domain Randomization* extends the learning strategy from the *Zero-Shot* baseline by randomizing observations and dynamics properties to increase robustness. The *Time-Optimal* baseline uses a precomputed time-optimal trajectory [91] that is tracked using an MPC controller. The dynamics model used by the trajectory generation and the MPC controller matches the simulated dynamics of experimental setting (i).

Performance is assessed by evaluating the fastest lap time, the average and minimum observed gate margin of successfully passed gates, and percentage of track successfully completed. The gate margin metric measures the distance between the drone and the closest point on the gate when crossing the gate plane. A high gate margin indicates that the quadrotor passed close to the center of the gate. Leaving a smaller gate margin can increase speed but can also increase risk of collision or missing the gate. Any lap that

## Appendix H. Champion-Level Drone Racing using Deep Reinforcement Learning

results in a crash is not considered valid.

The results are summarized in Table H.2. All approaches manage to successfully complete the task when deployed in idealized dynamics and ground-truth observations, with the *Time-Optimal* baseline yielding the lowest lap time. When deployed in settings that feature domain shift, either in the dynamics or the observations, the performance of all baselines collapses and none of the three baselines are able to complete even a single lap. This performance drop is exhibited by both learning-based and traditional approaches. In contrast, our approach, which features empirical models of dynamics and observation noise, succeeds in all deployment settings, with small increases in lap time.

**Table H.2** – Evaluation in simulation, with idealized dynamics (top) versus realistic dynamics (bottom) and ground-truth observations (left) versus noisy observations (right). We report the fastest achieved collision-free lap time in seconds, the average and smallest gate margin of successfully passed gates, and percentage of track completed. We compare our approach with a learning-based approach that performs zero-shot transfer, with and without domain randomization during training, as well as a traditional planning and control approach [91].

Approach		Ground-truth observations			Noisy observations		
		Lap time (↓) in seconds	Gate margin (↑) in meters	Completion (↑) in %	Lap time (↓) in seconds	Gate margin (↑) in meters	Completion (↑) in %
Ideal. dyn.	Zero-Shot Transfer [322]	4.88	0.63   0.46	100	∞	n/a   n/a	0
	Domain Randomization	5.06	0.60   0.47	100	∞	0.43   0.30	9
	Time-Opt. Traj. + MPC [91]	4.60	0.50   0.25	100	∞	0.48   0.29	9
	Ours	4.88	0.63   0.46	100	5.26	0.56   0.44	100
Real. dyn.	Zero-Shot Transfer [322]	∞	0.62   0.62	4	∞	0.41   0.21	9
	Domain Randomization	∞	0.28   0.28	4	∞	0.47   0.45	9
	Time-Opt. Traj. + MPC [91]	∞	n/a   n/a	0	∞	0.23   0.23	4
	Ours	5.20	0.51   0.30	100	5.42	0.48   0.23	100

The key feature that enables our approach to succeed across deployment regimes is the use of an empirical model of dynamics and observation noise, estimated from real-world data. A comparison between an approach that has access to such data and approaches that do not is not entirely fair. For that reason, we also benchmark the performance of all baseline approaches when having access to the same real-world data used by our approach. Specifically, we compare the performance in experimental setting (ii), which features the idealized dynamics model but noisy perception. All baseline approaches are provided with the predictions of the same Gaussian process model we use to characterize observation noise. The results are summarized in Table H.3. All baselines benefit from the more realistic observations, yielding higher completion rates. Nevertheless, our approach is the only one that reliably completes the entire track. In addition to the predictions of the observation noise model, our approach also takes into account the model’s uncertainty.

### Drone Hardware Configuration

The quadrotors used by the human pilots and Swift have the same weight, shape, and propulsion. The platform design is based on the Agilicious framework [94]. Each vehicle has a weight of 870 g and can produce a maximum static thrust of approximately 35 N,

**Table H.3** – Comparison to baselines that are provided with the same observation noise model used by our approach.

Approach	Lap time	Gate margin	Comp- letion
Zero-Shot Transfer [322]	4.92	0.57   0.41	42
Domain Randomization	$\infty$	0.34   0.23	19
Time-Opt. Traj. + MPC [91]	$\infty$	0.51   0.41	19
Ours	5.26	0.56   0.44	100

which results in a static thrust-to-weight ratio of 4.1. The base of each platform consists of an Armattan Chameleon 6-inch main frame that is equipped with T-Motor Velox 2306 motors and 5-inch, three-bladed propellers. An NVIDIA Jetson TX2 accompanied by a ConnectTech Quasar carrier board provides the main compute resource for the autonomous drones. The human-piloted drones do not carry a Jetson computer and are instead equipped with a corresponding ballast weight. Control commands in the form of collective thrust and bodyrates produced by the human pilots or Swift are sent to a commercial flight controller running BetaFlight, an open-source flight control software [340].

### Human Pilot Impressions

The following quotes convey the impressions of the three human champions that raced against Swift.

#### Alex Vanover:

- *These races will be decided at the Split-S, it is the most challenging part of the track.*
- *This was the best race! I was so close to the autonomous drone, I could really feel the turbulence when trying to keep up with it.*

#### Thomas Bitmatta:

- *The possibilities are endless, this is the start of something that could change the whole world. On the flip side, I'm a racer, I don't want anything to be faster than me.*
- *As you fly faster, you trade off precision for speed.*
- *It's inspiring to see the potential of what drones are actually capable of. Soon, the AI drone could even be used as a training tool to understand what would be possible.*

## Appendix H. Champion-Level Drone Racing using Deep Reinforcement Learning

---

**Marvin Schaepper:**

- *It feels different racing against a machine, because you know that the machine doesn't get tired.*

# I Data-Driven MPC for Quadrotors

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

Guillem Torrente\*, Elia Kaufmann\*, Philipp Foehn, and Davide Scaramuzza. “Data-driven mpc for quadrotors”. In: *IEEE Robot. Autom. Lett.* 6.2 (2021), pp. 3769–3776

## Data-Driven MPC for Quadrotors

Guillem Torrente\*, Elia Kaufmann\*, Philipp Foehn, Davide Scaramuzza

**Abstract** — Aerodynamic forces render accurate high-speed trajectory tracking with quadrotors extremely challenging. These complex aerodynamic effects become a significant disturbance at high speeds, introducing large positional tracking errors, and are extremely difficult to model. To fly at high speeds, feedback control must be able to account for these aerodynamic effects in real-time. This necessitates a modeling procedure that is both accurate and efficient to evaluate. Therefore, we present an approach to model aerodynamic effects using Gaussian Processes, which we incorporate into a Model Predictive Controller to achieve efficient and precise real-time feedback control, leading to up to 70% reduction in trajectory tracking error at high speeds. We verify our method by extensive comparison to a state-of-the-art linear drag model in synthetic and real-world experiments at speeds of up to 50km/h and accelerations beyond 4g.



## Supplementary Material

Video: <https://youtu.be/FHvDghUUQtc>

Code: [https://github.com/uzh-rpg/data\\_driven\\_mpc](https://github.com/uzh-rpg/data_driven_mpc)

### I.1 Introduction

Accurate trajectory tracking with quadrotors in high-speed and high-acceleration regimes is still a challenging research problem. While autonomous quadrotors have seen a significant gain in popularity and have been applied in a variety of industries ranging from agriculture to transport, security, infrastructure, entertainment, and search and rescue, they still do not exploit their full maneuverability. The ability to precisely control drones during fast and highly agile maneuvers would allow to not only fly fast in known-free environments, but also close to obstacles, humans, or through openings, where already small deviations from the reference have catastrophic consequences.

Operating a quadrotor at high speeds and controlling it through agile, high-acceleration maneuvers requires to account for complex aerodynamic effects acting on the platform. These effects are difficult to model, since they consist of a combination of propeller lift and drag dependent on the induced airstream velocity, fuselage drag, and complex or even turbulent effects due to the interaction between the propellers, the downwash of other propellers, and the fuselage. Furthermore, in the context of model-based feedback control, the model complexity is constrained by the feedback time-scale and computational capabilities of the executing platform. Therefore, it is not sufficient to find the most accurate model, but required to find an applicable trade-off between model accuracy and complexity.

[321] Very little work exists on agile control of quadrotors at speeds beyond  $5 \text{ m s}^{-1}$  and accelerations above  $2g$ , [79, 77, 162, 252, 228, 172, 92, 321]. Even though these works show agile control at various levels, none of them accounts for aerodynamic effects. This is not a limiting assumption when the quadrotor is controlled close to hover conditions, but introduces significant errors when tracking fast and agile trajectories. Other approaches use iterative learning control to perform highly aggressive trajectories [214], but they are constrained to a single maneuver and do not generalize.

The main challenge when performing aggressive flight is to identify a dynamics model of the platform that is capable of describing the aerodynamic effects while still being lightweight enough to guarantee real time performance. While there exist sophisticated computational fluid dynamics simulations that are able to model turbulent aerodynamic effects [350], they require hours of processing on a compute cluster, and still need to be abstracted in simplified models to be tractable in a control loop running at high frequency.

In this work, we propose to learn the aerodynamic effects acting on the platform from data. Inspired by [130, 157], we use Gaussian Processes to learn the residual dynamics with respect to a simplified quadrotor model that does not account for aerodynamic effects. Learning the residual dynamics simplifies the learning problem and allows describing the



**Figure I.1** – Our quadrotor platform reaches its physical limits at a pitch angle of 80 degrees while performing a lemniscate trajectory in our experiments. Throughout the trajectory, the platform reaches speeds of up to  $14\text{m s}^{-1}$  and accelerations beyond  $4g$ .

model augmentation using only a small number of inducing points for Gaussian Processes. Using such a small model allows leveraging the combined dynamics formulation in a Model Predictive Control (MPC) pipeline.

Our experiments, performed in simulation and in the real world, show that the proposed approach can significantly improve control performance for agile trajectories with speeds up to  $14\text{m s}^{-1}$  and accelerations exceeding  $4g$ . We show that the method generalizes between different trajectories and outperforms methods relying on simplified correction terms.

### Contributions

In this paper, we present a Model Predictive Control pipeline that is augmented with learned residual dynamics using Gaussian Processes. We extend the approach presented in [130, 157] to three-dimensional GP predictions for the quadrotor platform. By combining the learned GP corrections with the nominal quadrotor dynamics, we can learn an accurate dynamics model from a small number of inducing data points. Such a small model can be efficiently optimized within an MPC pipeline and allows for control frequencies greater than 100 Hz. We show that the augmented MPC improves trajectory tracking by up to 70% with respect to its nominal counterpart. We verify our method by extensive comparison to a state-of-the-art linear drag model in synthetic and real-world experiments at speeds of up to  $14\text{m s}^{-1}$  and accelerations beyond  $4g$ .

## I.2 Related Work

Performing fast and agile maneuvers with an autonomous robot requires knowledge of an accurate dynamics model of the platform. However, especially at high speeds and accelerations, such a description of the system is difficult to obtain due to hard-to-model effects caused by friction, aerodynamics or varying battery voltage. As modeling these effects significantly increases the problem complexity, controlling a robot under such conditions requires to find a trade-off between model expressivity and computational tractability. Most prior work on control of autonomous robots does not account for higher-order effects at all and treats them as external disturbances [79, 28, 201, 291, 212]. While this allows for very efficient and lightweight controller implementations, tracking performance progressively decreases for higher speeds. In [336], Nonlinear Incremental Dynamics Inversion is used to achieve robust tracking of fast trajectories. However, the reactive nature of this approach does not allow to account for future disturbances as the controller does not optimize over a horizon of actions.

A recent line of work [360, 85, 29, 193] investigates the application of dynamics learned entirely from data for a variety of applications such as robot arms, cars or fluids. These learned dynamics representations take the form of deep neural networks and substitute the nominal dynamics in the MPC. While the resulting dynamics models are very expressive, their optimization is often intractable due to local minima. A common way to overcome this challenge is to use sampling-based optimizers, which in turn scale poorly to high-dimensional input spaces.

Instead of learning the full dynamics from data, [130, 157, 301, 44] combine a nominal model with a learned correction term. This allows to limit the learned dynamics to have different dimensionality than the nominal system and provides the possibility to learn only specific effects that are difficult to capture with the nominal model.

For the particular case of quadrotor flight, the most prominent source of disturbances are aerodynamic effects originating from drag caused by the rotors and the fuselage, as well as lift effects that act on the platform at high speeds. By conducting controlled experiments in both wind tunnels as well as instrumented tracking volumes, previous work has shown a significant effect of aerodynamic forces already at linear speeds of  $5 \text{ m s}^{-1}$  [78, 331].

Neglecting other aerodynamic effects, previous work mainly studies the effect of rotor drag [36, 78, 220]. Rotor drag effects originate from blade flapping and induced drag of the rotors. These effects are typically combined as linear effects in a lumped parameter dynamical model [218]. In [78], the authors identify a linear model for the rotor drag and use it to compute feedforward terms of a PID controller. Even though they show improved trajectory tracking performance, they evaluate their linear model only up to  $5 \text{ m s}^{-1}$ . At these speeds, the linear effect of rotor drag dominates the fuselage drag. We integrate the model of [78] in an MPC pipeline to act as baseline of our approach.

Similar to our work, in [130, 157, 224, 42, 65], the authors use Gaussian Processes to improve the control performance of a robotic platform. In [224], Gaussian Processes are used on a quadrotor to correct for wind disturbances. Since instead of platform states only

observed disturbances are fed to the GPs, this approach does not learn a dynamics model and can only react to disturbances once they have been observed. In [42], the authors separately learn the translational and rotational dynamics of a quadrotor platform. As this approach learns the full model from data, it requires a large number of training points and is computationally very expensive. As a result, the prediction horizon of the MPC needs to be reduced to a single point to achieve near real-time performance.

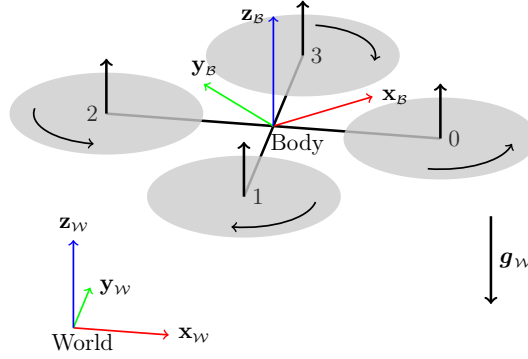
In [65], a robust experience-driven predictive controller (EPC) is proposed that uses Gaussian belief propagation to account for uncertainties in the state estimate. The controller demonstrates robust constraint satisfaction on a quadrotor platform, where it is integrated into an MPC that controls the translational dynamics of the vehicle. In [130, 157], the authors use the predictions of the Gaussian Processes to improve the tracking performance of an autonomous race car by learning the *residual* dynamics of a nominal model. Learning such residual dynamics instead of the full model allows to simplify the learning problem and as a result reduce the number of training points in the GP.

Our work is inspired by these approaches, but extends [130, 157] to three-dimensional GP predictions for the quadrotor platform. Instead of learning a mapping from observed disturbances to future disturbance as in [224], we focus on fast flight and correct for aerodynamic effects that arise due to the fast ego-motion of the platform. We tightly integrate the predictions of the Gaussian Processes into the MPC formulation. Instead of using virtual inputs such as bodyrates and collective thrust [79], our MPC models the dynamics down to the motor inputs and can therefore account for the true actuation limits of the platform. Our work is the first to combine Gaussian Processes with a full-state quadrotor MPC formulation to model aerodynamic drag effects while still being able to account for the true actuation limits of the platform.

### I.3 Methodology

#### I.3.1 Notation

We denote scalars in lowercase  $s$ , vectors in lowercase bold  $\mathbf{v}$ , and matrices in uppercase bold  $\mathbf{M}$ . We define the World  $W$  and Body  $B$  frames with orthonormal basis i.e.  $\{\mathbf{x}_W, \mathbf{y}_W, \mathbf{z}_W\}$ . The frame  $B$  is located at the center of mass of the quadrotor. Note that we assume all four rotors are situated in the  $xy$ -plane of frame  $B$ , as depicted in Fig. I.2. A vector from coordinate  $\mathbf{p}_1$  to  $\mathbf{p}_2$  expressed in the  $W$  frame is written as:  ${}_W\mathbf{v}_{12}$ . If the vector's origin coincide with the frame it is described in, we drop the frame index, e.g. the quadrotor position is denoted as  $\mathbf{p}_{WB}$ . Furthermore, we use unit quaternions  $\mathbf{q} = (q_w, q_x, q_y, q_z)$  with  $\|\mathbf{q}\| = 1$  to represent orientations, such as the attitude state of the quadrotor body  $\mathbf{q}_{WB}$ . Finally, full SE3 transformations, such as changing the frame of reference from body to world for a point  $\mathbf{p}_{B1}$ , can be described by  ${}_W\mathbf{p}_{B1} = {}_W\mathbf{t}_{WB} + \mathbf{q}_{WB} \odot \mathbf{p}_{B1}$ . Note the quaternion-vector product denoted by  $\odot$  representing a rotation of the vector by the quaternion as in  $\mathbf{q} \odot \mathbf{v} = \mathbf{q}\mathbf{v}\bar{\mathbf{q}}$ , where  $\bar{\mathbf{q}}$  is the quaternion's conjugate.



**Figure I.2** – Diagram of the quadrotor model with the world and body frames and propeller numbering convention.

### I.3.2 Nominal Quadrotor Dynamics Model

We assume the quadrotor is a 6 degree-of-freedom rigid body of mass  $m$  and diagonal moment of inertia matrix  $\mathbf{J} = \text{diag}(J_x, J_y, J_z)$ . Our model is similar to [79, 162] but we write the nominal dynamics  $\dot{\mathbf{x}}$  up to second order derivatives, leaving the quadrotors individual rotor thrusts  $T_i \forall i \in (0, 3)$  as control inputs  $\mathbf{u}$ . The state space is thus 13-dimensional and its dynamics can be written as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}}_{WB} \\ \dot{\mathbf{q}}_{WB} \\ \dot{\mathbf{v}}_{WB} \\ \dot{\boldsymbol{\omega}}_B \end{bmatrix} = \mathbf{f}_{dyn}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{v}_W \\ \mathbf{q}_{WB} \cdot \begin{bmatrix} 0 \\ \boldsymbol{\omega}_B/2 \end{bmatrix} \\ \frac{1}{m} \mathbf{q}_{WB} \odot \mathbf{T}_B + \mathbf{g}_W \\ \mathbf{J}^{-1} (\boldsymbol{\tau}_B - \boldsymbol{\omega}_B \times \mathbf{J} \boldsymbol{\omega}_B) \end{bmatrix}, \quad (\text{I.1})$$

where  $\mathbf{g}_W = [0, 0, -9.81 \text{ m/s}^2]^\top$  denotes Earth's gravity,  $\mathbf{T}_B$  is the collective thrust and  $\boldsymbol{\tau}_B$  is the body torque as in:

$$\mathbf{T}_B = \begin{bmatrix} 0 \\ 0 \\ \sum T_i \end{bmatrix} \quad \text{and} \quad \boldsymbol{\tau}_B = \begin{bmatrix} d_y(-T_0 - T_1 + T_2 + T_3) \\ d_x(-T_0 + T_1 + T_2 - T_3) \\ c_\tau(-T_0 + T_1 - T_2 + T_3) \end{bmatrix} \quad (\text{I.2})$$

where  $d_x, d_y$  are the rotor displacements and  $c_\tau$  is the rotor drag torque constant. To incorporate these dynamics in discrete time algorithms, we use an explicit Runge-Kutta method of 4th order  $\mathbf{f}_{RK4}(\mathbf{x}, \mathbf{u})$  to integrate  $\dot{\mathbf{x}}$  given an initial state  $\mathbf{x}_k$ , input  $\mathbf{u}_k$  and integration step  $\delta t$  by:

$$\mathbf{x}_{k+1} = \mathbf{f}_{RK4}(\mathbf{x}_k, \mathbf{u}_k, \delta t). \quad (\text{I.3})$$

### I.3.3 Gaussian Process-Augmented Dynamics

Inspired by [130, 157], we use Gaussian Processes to complement the nominal dynamics of the quadrotor in an MPC pipeline. In this setting, the GPs predict the error of the dynamics and correct them at every time instance  $t_k$ . Similar to most GP-based learning

## Appendix I. Data-Driven MPC for Quadrotors

---

problems, we assume the existence of the inaccessible true dynamics  $\mathbf{f}_{true}$  of the quadrotor, which we measure as  $\tilde{\mathbf{y}}_{k+1}$  through a noisy process at discrete time instances  $t_k$ :

$$\tilde{\mathbf{y}}_{k+1} = \mathbf{f}_{true}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k \quad (\text{I.4})$$

We further assume that  $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$  is Gaussian noise, where  $\mathbf{\Sigma}$  is the time-invariant and diagonal covariance matrix. This means we can effectively treat each dimension of  $\mathbf{y}_k$  independently through a separate 1-dimensional output GP. We use the Radial Basis Function (RBF) kernel

$$\kappa(\mathbf{z}_i, \mathbf{z}_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{z}_i - \mathbf{z}_j)^\top \mathbf{L}^{-2}(\mathbf{z}_i - \mathbf{z}_j)\right) + \sigma_n^2 \quad (\text{I.5})$$

where  $\mathbf{L}$  is the diagonal length scale matrix and  $\sigma_f, \sigma_n$  represent the data and prior noise variance, respectively, and  $\mathbf{z}_i, \mathbf{z}_j$  represent data features.

We redefine the system dynamics as a (corrected,  $\mathbf{f}_{corr}$ ) combination of the dynamics (I.1) plus the mean posterior of a GP,  $\boldsymbol{\mu}$ . The GP only corrects a subset of the state, determined in the selection matrix  $\mathbf{B}_d$ , using the feature vector  $\mathbf{z}_k$ , determined by selection matrix  $\mathbf{B}_z$ :

$$\mathbf{f}_{corr}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{f}_{dyn}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{B}_d \boldsymbol{\mu}(\mathbf{z}_k) \quad (\text{I.6})$$

$$\mathbf{z}_k = \mathbf{B}_z [\mathbf{x}_k^\top \quad \mathbf{u}_k^\top]^\top. \quad (\text{I.7})$$

Given the concatenated training feature samples  $\mathbf{Z}$  and the query feature samples  $\mathbf{Z}_k$ , the mean and covariance of the GP prediction can be recovered as follows:

$$\begin{aligned} \boldsymbol{\mu}(\mathbf{Z}_k) &= \mathbf{K}_k^\top \mathbf{K}^{-1} \mathbf{Z} & \boldsymbol{\Sigma}_{\mu k} &= \mathbf{K}_{kk} - \mathbf{K}_k^\top \mathbf{K}^{-1} \mathbf{K}_k \\ \text{with } \mathbf{K} &= \kappa(\mathbf{Z}, \mathbf{Z}) + \sigma_n^2 \mathbf{I} & & \\ \mathbf{K}_k &= \kappa(\mathbf{Z}, \mathbf{Z}_k) & \mathbf{K}_{kk} &= \kappa(\mathbf{Z}_k, \mathbf{Z}_k). \end{aligned} \quad (\text{I.8})$$

where  $\mathbf{K}_{ij}$ , the entry of  $\mathbf{K}$  with index  $i, j$ , is  $\mathbf{K}_{ij} = \kappa(\mathbf{z}_i, \mathbf{z}_j)$ .

Given the mean and covariance of the GP, not only is it possible to learn the corrected dynamics, but in addition we can also propagate the corrected model forward in time to use it in an MPC. To propagate the state we simply substitute the nominal dynamics  $\mathbf{f}_{dyn}$  with the corrected model  $\mathbf{f}_{corr}$  in the Runge-Kutta integration. For the propagation of the covariance, we refer to the formulation in [130, 157].

### I.3.4 MPC Formulation

In its most general form, MPC stabilizes a system subject to its dynamics  $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$  along a reference  $\mathbf{x}^*(t), \mathbf{u}^*(t)$ , by minimizing a cost  $\mathcal{L}(\mathbf{x}, \mathbf{u})$  as in:

$$\begin{aligned}
 & \min_{\mathbf{u}} \int \mathcal{L}(\mathbf{x}, \mathbf{u}) & (I.9) \\
 \text{subject to } & \dot{\mathbf{x}} = \mathbf{f}_{dyn}(\mathbf{x}, \mathbf{u}) & \mathbf{x}(t_0) = \mathbf{x}_{init} \\
 & \mathbf{r}(\mathbf{x}, \mathbf{u}) = 0 & \mathbf{h}(\mathbf{x}, \mathbf{u}) \leq 0
 \end{aligned}$$

where  $\mathbf{x}_0$  denotes the initial condition and  $\mathbf{h}$ ,  $\mathbf{r}$  can incorporate (in-)equality constraints, such as input limitations.

For our application, and as most commonly done, we specify the cost to be of quadratic form  $\mathcal{L}(\mathbf{x}, \mathbf{u}) = \|\mathbf{x} - \mathbf{x}^*\|_Q^2 + \|\mathbf{u} - \mathbf{u}^*\|_R^2$  and discretize the system into  $N$  steps over time horizon  $T$  of size  $dt = T/N$ . We account for input limitations by constraining  $0 \leq \mathbf{u} \leq u_{max}$ , and optionally include the GP predictions within the system dynamics.

$$\begin{aligned}
 & \min_{\mathbf{u}} \mathbf{x}_N^\top Q \mathbf{x}_N + \sum_{k=0}^N \mathbf{x}_k^\top Q \mathbf{x}_k + \mathbf{u}_k^\top R \mathbf{u}_k & (I.10) \\
 \text{subject to } & \mathbf{x}_{k+1} = \mathbf{f}_{RK4}(\mathbf{x}_k, \mathbf{u}_k, \delta t) \\
 & \mathbf{x}_0 = \mathbf{x}_{init} \\
 & u_{min} \leq \mathbf{u}_k \leq u_{max}
 \end{aligned}$$

where  $\mathbf{f}_{RK4}$  can be extended to the corrected dynamics  $\mathbf{f}_{cor}$ .

To solve this quadratic optimization problem we construct it using a multiple shooting scheme [67] and solve it through a sequential quadratic program (SQP) executed in a real-time iteration scheme (RTI) [67]. All implementations are done using ACADOS [352] and CasADi [9].

### I.3.5 Practical Implementation

The implementation of the learned dynamics of our GP-MPC must be designed to maximize the performance while minimizing the computational cost added to the optimization. Note that aerodynamic effects operate on the body reference frame. Likewise, the training dataset is adjusted such that the learning problem setup is to identify such a mapping from body frame velocities  ${}_B\mathbf{v}$  to body frame acceleration disturbances  ${}_B\mathbf{a}_e$ , so that  ${}_B\mathbf{a}_e = \boldsymbol{\mu}({}_B\mathbf{v})$ . Furthermore, to reduce the need for additional training samples, we reduce

the dimensionality of our input space such that the mappings are learned axis wise.

$${}^B\mathbf{a}_{ek} = \boldsymbol{\mu}({}^B\mathbf{v}_k) = \begin{bmatrix} \mu_{vx}({}^Bv_{xk}) \\ \mu_{vy}({}^Bv_{yk}) \\ \mu_{vz}({}^Bv_{zk}) \end{bmatrix} \quad (\text{I.11})$$

$$\boldsymbol{\Sigma}_{\mu}({}^B\mathbf{v}_k) = \text{diag} \left( \begin{bmatrix} \sigma_{vx}^2({}^Bv_{xk}) \\ \sigma_{vy}^2({}^Bv_{yk}) \\ \sigma_{vz}^2({}^Bv_{zk}) \end{bmatrix} \right) \quad (\text{I.12})$$

### I.3.6 Data Collection and Model Learning

To fit the GPs, real-world flight data is collected (as detailed in Sec. I.4) using the nominal dynamics model. For each sample at time  $t_k$ , the velocity at the next sample point  ${}^B\mathbf{v}_{k+1}$  and the predicted velocity at the next sample point  ${}^B\hat{\mathbf{v}}_{k+1}$  are recorded, together with the timestep  $\delta t_k$ . We can then compute the time-normalized velocity error, corresponding to the acceleration error:

$${}^B\mathbf{a}_{ek} = \frac{{}^B\mathbf{v}_{k+1} - {}^B\hat{\mathbf{v}}_{k+1}}{\delta t_k} \quad (\text{I.13})$$

To select the hyperparameters of the kernel function  $(l, \sigma_n, \sigma_f)$ , we perform maximum likelihood optimization on the collected dataset. Being a non-parametric method, the complexity of GP regression depends on the number of training points. As using the full dataset would make MPC optimization intractable in real time, we subsample the dataset and only use a small number of inducing points. To this end, we leverage the smooth nature of the aerodynamic effects by sampling these points at regular intervals in the ranges of the training set.

## I.4 Experiments and Results

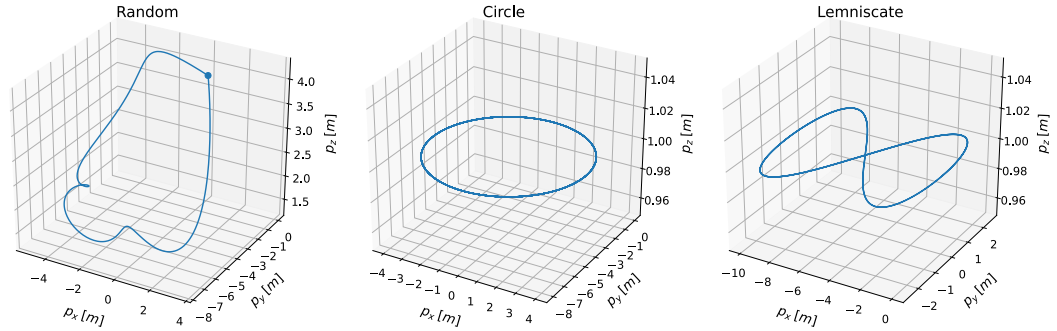
We design our evaluation procedure to address the following questions: i) What is the contribution of the learned dynamics of our GP-MPC in a closed-loop tracking task? ii) How does our GP-MPC compare to an MPC with linear aerodynamic effect compensation, as proposed in [78]? iii) How does the learned model generalize to unseen trajectories? Finally, we validate our design choices with ablation studies. We refer the reader to the attached video to understand the dynamic nature of our experiments.

### I.4.1 Experimental Setup

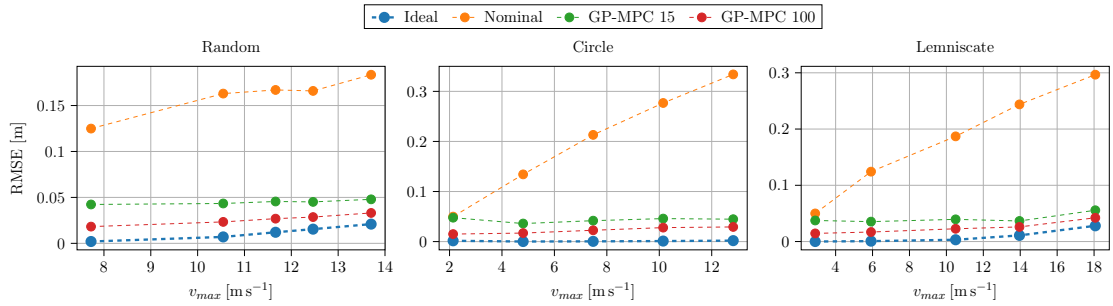
We conduct experiments both in simulation as well as on a real quadrotor platform. To assess our proposed approach, the quadrotor executes three different trajectories (Random, Circle, Lemniscate), illustrated in Fig. I.3. The lemniscate trajectory lies in the horizontal plane and is defined by  $[x(t) = 2 \cos(\sqrt{2}t); y(t) = 2 \sin(\sqrt{2}t) \cos(\sqrt{2}t)]$ .



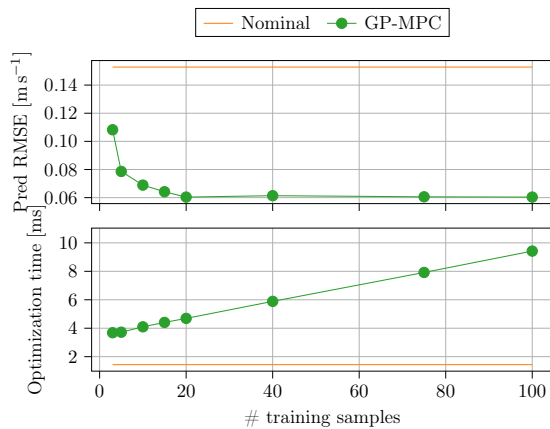
## I.4. Experiments and Results



**Figure I.3** – Trajectories considered in this work. Left: single-loop randomly-generated polynomial trajectory with motion along all axes. The tracking starts and ends at the upper-right corner. Center and right: circular and lemniscate trajectories respectively. Both have zero translation along the  $z$  axis, tracking starts at 0 velocity, ramps up until reaching a peak, and ramps down back to hover. The position references remain as shown in the figures in all cases.



**Figure I.4** – Closed-loop position tracking error as a function of maximum velocity achieved in our custom simulator. *Ideal* denotes the nominal MPC performance in a disturbance-free scenario. *Nominal* corresponds to the un-augmented MPC, and *GP-MPC 15* and *GP-MPC 100* are our GP-augmented controllers where the GP's have been trained with 15 and 100 training samples.



**Figure I.5** – Trade-off between number of training samples, GP performance (top) and optimization time (bottom). Models are trained and evaluated on data collected in our Simplified Simulation.

We compare the tracking performance on these trajectories using our MPC with the *Nominal* quadrotor model (I.1), and the improvement after adding different correction terms identified from data. We study two possible augmentations: *GP-MPC* (ours) and *RDRv*. The RDRv approach was proposed in [78] as a feed-forward PID controller term, which identifies a set of linear drag coefficients along the body axes. We instead incorporate this linear compensation into the nominal dynamics of our MPC pipeline. Note that the three control approaches only differ in the dynamics model used by the MPC, i.e. they use the same control frequency and cost matrices. Furthermore, both the GP-MPC and the RDRv are always trained on the same dataset.

For the simulation experiments, we perform a Nominal run on random polynomial trajectories of high aggressiveness to collect training samples for the GP and the coefficient identification for RDRv. With both models fitted, we deploy all three controllers, Nominal, RDRv and GP-MPC, on the test trajectories without retraining. For the real-world experiments, we first perform a run of the Nominal baseline on both circle and lemniscate trajectories, which is also used for GP training and RDRv coefficient identification. In the subsequent rollouts, we test RDRv and GP-MPC on these two trajectories in different permutations.

### I.4.2 Experiments in Simulation

We first evaluate the performance for individual maneuvers in simulation. To isolate the effects of varying MPC computation times for different models, we divide the simulation experiments into two parts: *simplified simulation* and *Gazebo simulation*. This setup allows to compare the predictive performance of arbitrary sized models without the need to correct for varying computation times.

**Simplified Simulation** This simulation is constituted of a simple forward integration of the system dynamics (I.1) using an explicit Runge-Kutta method of 4th order with a step size of 0.5 ms. We assume to have access to perfect odometry measurements of the quadrotor, ideal tracking of the commanded single-rotor thrusts, and that the MPC computation is instantaneous. The simulator models drag effects caused both by the rotors as well as the fuselage. Additionally, zero-mean Gaussian noise forces and torques are simulated that act on the quadrotor body, as well as asymmetric noise on the motor voltage signals.

In the simplified simulation, we investigate the influence of the number of training points of the GP on the predictive performance. As the choice of this hyperparameter constitutes a trade-off between model accuracy and computation time, we seek the model with the minimum number of inducing points that surpasses a desired performance threshold. This effect is investigated with two experiments: first, we analyze the trade-off between GP performance and optimization time. In the second experiment, we extend the comparison of different-sized GPs to closed-loop experiments on the three test trajectories.

Having identified the optimal size of the GP, we perform an additional set of experiments to compare the tracking performance on the circle and lemniscate test trajectories between

the Nominal and RDRv baselines and our approach. Both trajectories are executed up to varying maximum speeds, where the highest speed pushes the platform to its physical limits. The training set for both the RDRv and the GP models in this simulator is collected by executing random polynomial trajectories of high aggressiveness (such as Fig. I.3 left) with the quadrotor, up to  $16\text{m s}^{-1}$  axis-wise. This technique works well in simulation since it allows to explore densely all the ranges of operating points without risk of breaking the aircraft if tracking fails.

The results of the GP size analysis are summarized in Fig. I.5. As can be seen, the complexity of the optimization problem approximately follows a linear function with respect to the number of training points. Since the predictive performance barely increases when adding more than 20 inducing points, we identify the optimal range to be between 15 to 25 samples, corresponding to 4 – 5 ms of optimization time. For comparative purposes, we illustrate in Fig. I.4 how two of our GP models perform in closed-loop tracking for 15 and 100 training samples. It can be verified that a larger number of samples is strictly beneficial, but comes at the expense of an increase in optimization time. In fact, such a large model is not usable for a real time application of our pipeline. Based on this evidence, we chose to use 20 inducing points for the rest of this work.

The main results of the closed-loop tracking experiments are summarized in Table I.1. The table reports position tracking error in millimeters for both maneuvers at varying maximum speeds. While the *Ideal* column indicates the tracking error in case of no unmodelled disturbances (i.e. the MPC dynamics model perfectly fit the actual system), the *Nominal* column represents the baseline when no model augmentations are enabled. Note that even though the MPC controller performs very well in the Ideal scenario, it does not achieve zero tracking error due to discretization effects. Both the RDRv baseline as well as the GP-MPC significantly improve the tracking error compared to the non-augmented Nominal case. However, while RDRv performs comparably to our approach up to speeds of  $4\text{ m s}^{-1}$ , it starts to fail for higher speeds due to its inability to model higher-order aerodynamic effects such as body drag. Our approach also captures these effects very well and shows consistent improvement for the full range of tested speeds.

**Gazebo Simulation** To verify the results obtained in the simplified simulation in a well-known quadrotor simulator, we also perform closed-loop tracking experiments in Gazebo [182]. We employ the AscTec Hummingbird quadrotor model using the RotorS extension [102]. To properly evaluate the performance of our pipeline, we also use ground truth odometry measurements instead of a state estimator. We collect a dataset containing velocities in the range  $[-12, 12]\text{ m s}^{-1}$  for training our models. This dataset is obtained by tracking randomly generated aggressive trajectories, as with the simplified simulator. We execute the circle and lemniscate trajectories at increasing speeds and compare the tracking performance of the Nominal and RDRv baselines, as well as our approach. Note that once again we use completely independent training and test sets in our setup to ensure our models can generalize to new trajectories.

The main results of the Gazebo experiments are summarized in Fig. I.6. In this case, both RDRv as well as our GP-MPC achieve very similar performance over the full range

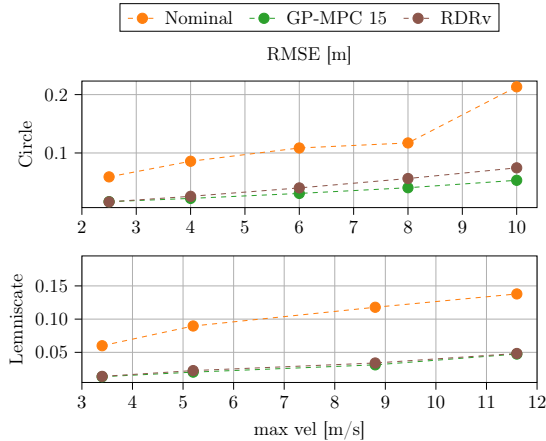


Figure I.6 – Closed-loop position tracking error as a function of maximum velocity achieved in the RotorS Gazebo simulator in the circle and the lemniscate trajectories.

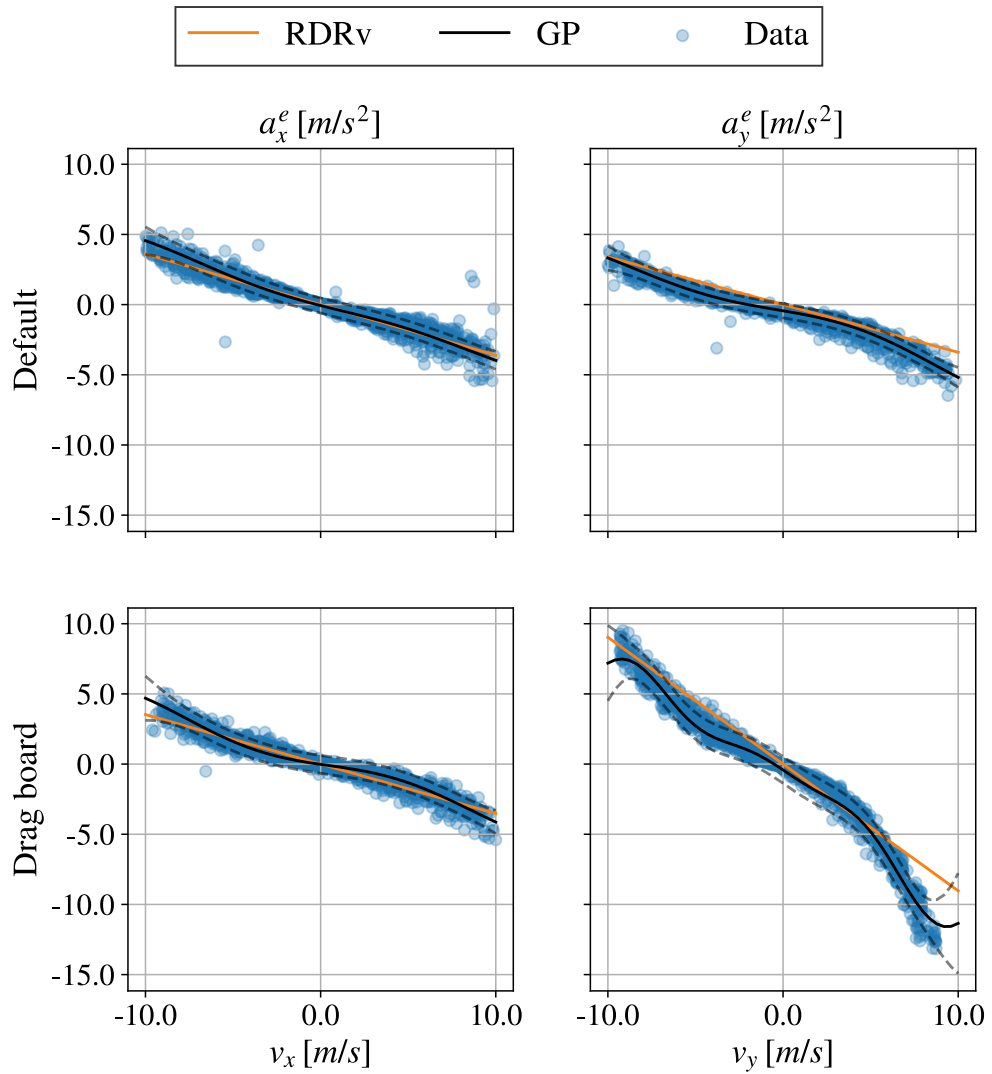
of tested speeds. This is expected, as the RotorS package implementation only simulates rotor drag as aerodynamic effect [102], which follows a linear mapping with respect to the body frame velocity [220]. The true aerodynamic effects acting on a quadrotor however are a combination of rotor drag, body drag and turbulent effects caused by the propellers. We are analyzing these effects in more detail in the following section.

### I.4.3 Experiments in the Real World

Lastly, we compare the performance of our GP-MPC against both Nominal as well as RDRv controllers on a real quadrotor. We use a custom quadrotor that weighs 0.8kg and has a thrust-to-weight ratio of 5:1. We run the controller on a laptop computer and send control commands in the form of collective thrust and desired bodyrates at 50 Hz to the quadrotor through a Laird RM024 radio module. A PID controller running onboard the quadrotor tracks the sent commands. The quadrotor flies in an indoor arena equipped with an optical tracking system that provides pose estimates at 100 Hz. Note that our control method also works with state estimates that are obtained differently than with a motion capture system. As in the simulation experiments, we compare the tracking error along both circle and lemniscate trajectories with speeds up to  $14 \text{ m s}^{-1}$ .

To demonstrate that our approach can correct for complex aerodynamic effects, we perform the real world experiments in two settings: in setting i) we perform all maneuvers with the standard quadrotor setup, while in setting ii) we extend the quadrotor body with a vertical drag board. This drag board introduces additional asymmetric aerodynamic disturbance as can be seen in Fig. I.7. For the real world experiment, we use 20 inducing points on our GP’s.

Table I.2 summarizes the results of our real world experiments in setting i). We train two GP models on the circle and lemniscate trajectories, and use them at test time in all permutations. As can be seen, our methods as well as the RDRv baseline improve tracking performance by up to 50%, with our approach slightly outperforming the RDRv



**Figure I.7** – Aerodynamic effects observed in the real quadrotor platform along body axes  $x$  (left column) and  $y$  (right column) as a function of body frame velocity. The platform was studied in its default configuration (upper row), and with an additional flat board attached along the body  $x$  axis (lower row), resulting in a significantly increased body drag effect in the  $y$  direction.

baseline. This result can be explained by the fact that the quadrotor platform used in setting i) is very compact and powerful, rendering the main source of disturbance being the rotor drag. Rotor drag is a linear effect, which can be well compensated for by the linear RDRv model augmentation. The slight improvement of GP-MPC over RDRv in this setting can be explained by the ability of the GPs to also account for imperfect thrust mappings.

Finally, we compare the circle tracking for settings i) and ii) in Table I.3. As can be seen, our approach significantly outperforms *RDRv* in setting ii), where the latter fails to capture the full nonlinearity of aerodynamic effects. This can be verified also in Fig. I.7, where the linear fit leads to significant bias.

### I.5 Conclusion

In this work, we propose the usage of Gaussian Processes to augment the nominal dynamics of a quadrotor to compensate for aerodynamic effects. This GP-based model augmentation is integrated in a Model Predictive Controller and the resulting system significantly improves positional tracking error, both in simulation and on a real quadrotor. Using data from previously recorded flights, the GP's are trained to predict the acceleration error of the nominal model given its current velocity in body frame.

In extensive experiments in simulation and the real world, we show that our approach outperforms a state-of-the-art linear drag model. Furthermore, our GP-augmented controller opens up interesting lines of follow-up research for future development. On one hand, we plan to make use of the predicted uncertainty to perform safe agile trajectories close to obstacles. On the other hand, leveraging the fast fitting time of our GP models (in the order of seconds), training and control loop can be executed in parallel on separate threads in real time during flight. This would enable to adapt the dynamics model to varying external or internal conditions such as wind disturbance or battery voltage.

**Table I.1** – Comparison of closed-loop tracking errors on the circle and lemniscate trajectories in simulation.

		Model					
		Ideal	Nominal	RDRv		GP-MPC	
Ref.	$v_{\text{peak}}$ [m s <sup>-1</sup> ]	RMSE [mm]	RMSE [mm]	RMSE [mm]	%↓	RMSE [mm]	%↓
Circle	4	0.1	114.1	18.1	84	16.1	<b>85</b>
	8	0.4	241.2	56.9	76	25.4	<b>89</b>
	12	1.2	338.3	93.0	72	28.4	<b>93</b>
Lemn.	4	0.3	104.0	15.5	<b>85</b>	16.3	84
	8	1.5	157.7	32.3	79	20.3	<b>87</b>
	12	4.2	212.4	60.6	71	24.4	<b>88</b>
	Opt. dt [ms]	1.32		1.76		4.13	

**Table I.2** – Comparison of the RDRv and GP-MPC methods in the real world experiments.

Ref.	Model RMSE [mm]						
	Nomin.	GP (circle)	%↓	GP (lemn.)	%↓	RDRv	%↓
Circle	319.7	172.9	46	141.0	<b>56</b>	168.3	47
Lemn.	396.2	254.2	<b>36</b>	266.3	33	269.3	33

**Table I.3** – Velocity-dependent tracking performance of the augmented MPC methods on the circle trajectory.

		Model				
		Nominal	RDRv		GP-MPC 20	
Config.	$v_{\text{range}}$ [m s <sup>-1</sup> ]	RMSE [m]	RMSE [m]	%↓	RMSE [m]	%↓
Default	0-2	0.087	0.130	-49	0.109	-25
	2-4	0.233	0.119	49	0.103	<b>56</b>
	4-6	0.329	0.177	46	0.129	<b>61</b>
	6-8	0.458	0.210	54	0.154	<b>66</b>
	8-10	0.531	0.192	<b>64</b>	0.203	62
Drag board	0-2	0.197	0.132	33	0.060	<b>69</b>
	2-4	0.346	0.287	17	0.078	<b>77</b>
	4-6	0.564	0.381	32	0.141	<b>75</b>
	6-8	0.837	0.463	45	0.219	<b>74</b>
	8-10	0.912	<i>Crash</i>	?	0.379	<b>59</b>





# J NeuroBEM: Hybrid Aerodynamic Quadrotor Model

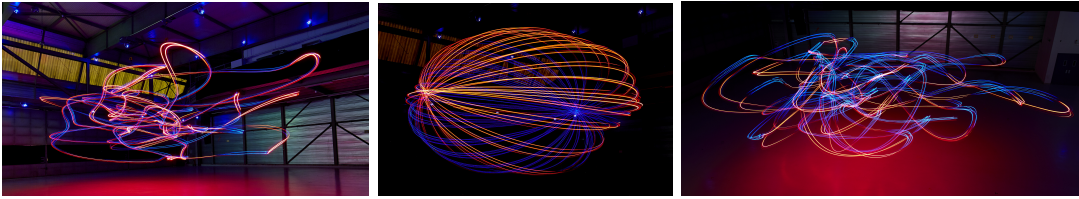
The version presented here is reprinted, with permission, from:

Leonard Bauersfeld\*, Elia Kaufmann\*, Philipp Foehn, Sihao Sun, and Davide Scaramuzza.  
“NeuroBEM: Hybrid Aerodynamic Quadrotor Model”. In: *RSS: Robotics, Science, and Systems* (2021)

# NeuroBEM: Hybrid Aerodynamic Quadrotor Model

Leonard Bauersfeld\*, Elia Kaufmann\*, Philipp Foehn, Sihao Sun, Davide Scaramuzza

**Abstract** — Quadrotors are extremely agile, so much in fact, that classic first-principle-models come to their limits. Aerodynamic effects, while insignificant at low speeds, become the dominant model defect during high speeds or agile maneuvers. Accurate modeling is needed to design robust high-performance control systems and enable flying close to the platform’s physical limits. We propose a hybrid approach fusing first principles and learning to model quadrotors and their aerodynamic effects with unprecedented accuracy. First principles fail to capture such aerodynamic effects, rendering traditional approaches inaccurate when used for simulation or controller tuning. Data-driven approaches try to capture aerodynamic effects with blackbox modeling, such as neural networks; however, they struggle to robustly generalize to arbitrary flight conditions. Our hybrid approach unifies *and outperforms* both first-principles blade-element momentum theory and learned residual dynamics. It is evaluated in one of the world’s largest motion-capture systems, using autonomous-quadrotor-flight data at speeds up to 65 km/h. The resulting model captures the aerodynamic thrust, torques, and parasitic effects with astonishing accuracy, outperforming existing models with 50% reduced prediction errors, and shows strong generalization capabilities beyond the training set.



**Figure J.1** – Long-exposure images depicting quadrotor trajectory tracking at speeds up to 65 km/h in a large-scale motion-capture system. The captured data is used to fit a hybrid quadrotor model combining blade-element-momentum (BEM) theory with a neural network compensating residual dynamics. This hybrid model reproduces the flown trajectories in simulation with a positional RMSE error reduction of over 50% compared to state-of-the-art.

## Supplementary Material

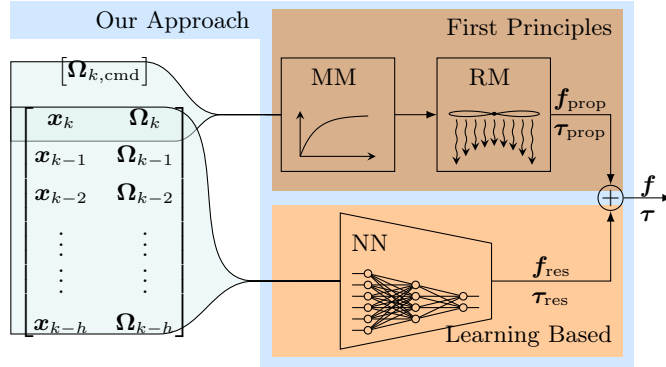
A narrated video illustrating our approach is available at <https://youtu.be/Nze1wlfmzTQ>. Code and dataset can be found at <http://rpg.ifi.uzh.ch/NeuroBEM.html>.

### J.1 Introduction

In recent years, research on fast navigation of autonomous quadrotors has made tremendous progress, continually pushing the vehicles to more aggressive maneuvers [296, 203, 172, 91] (Figure J.1). To further advance the field, several competitions have been organized, such as the autonomous drone race series at the recent IROS and NeurIPS conferences [235, 217] and the AlphaPilot challenge [92]. In the near future, estimation and control algorithms will reach the level of maturity necessary to push autonomous quadrotors to the bounds of what is physically possible. This presents the need for quadrotor models that can predict the behaviour of the platform even during highly aggressive maneuvers.

Accurately modeling quadrotors flying at their physical limits is extremely challenging and requires to capture complex effects due to aerodynamic forces, motor dynamics, and vibrations. Especially aerodynamic forces pose a challenge, as they depend on hidden state variables like airflow, which cannot be easily measured. Furthermore, the individual downwash induced by the rotors interacts with both the frame and the blades depending on the current state of the platform. The repeatability of tracking errors observed in prior work [277, 296, 17] and in this work when performing aggressive maneuvers suggests that the difficulty of learning quadrotor dynamics is not caused by stochasticity in the dynamics, but rather by unobserved state variables such as airflow.

Traditional approaches to quadrotor modeling limit the captured effects to simple linear drag approximations and quadratic thrust curves [102, 309, 78]. Such approximations are computationally efficient and describe the platform well in low-speed regimes, but exhibit increasing bias at higher velocities as they neglect the influence of the inflow velocity on the generated thrust. More elaborate models based on blade-element-momentum (BEM)



**Figure J.2** – Overview of the proposed architecture to predict aerodynamic forces and torques. The physical modeling pipeline (upper part) consists of a motor model (MM) and a rotor model (RM)—detailed in Sections J.3.3 and J.3.4. It takes the current state  $\mathbf{x}_k$ , current motor speeds  $\boldsymbol{\Omega}_k$ , and the motor speed command  $\boldsymbol{\Omega}_{k,\text{cmd}}$  as an input. Combined with the estimate of the residual forces and torques predicted by the neural network (NN) using the current and past  $h$  states, the acting force  $\mathbf{f}$  and torque  $\boldsymbol{\tau}$  are calculated.

theory manage to accurately model single rotors at high wind velocities, but they do not account for the aerodynamic interactions between rotors and the frame. Parametric gray-box models [331] aim to overcome these limitations by describing the forces and torques as a linear combination of library functions. While these models can perform well, their performance hinges on the appropriate choice of basis functions, which require human expert knowledge to design. Recent research has investigated computational fluid dynamics [350] to model the aerodynamic effects at play during different flight conditions. While being very accurate, such approaches are computationally expensive and need hours of processing on a compute cluster, rendering them impractical for experiments spanning more than a few seconds.

Accurately predicting forces acting on the quadrotor at high speeds requires to implicitly estimate the airflow around the vehicle. Although this state variable cannot be directly observed, it can be deduced from a sequence of measurements of other observable state variables. Thus, learning a high-order dynamics model requires a method for regression of a nonlinear function in a high-dimensional input space. Deep neural networks have shown to excel at such high-dimensional regression tasks and have already been applied to dynamic system modeling [277, 17, 231, 274, 115]. Despite showing promising performance, such purely-learned models require large amounts of data and require careful regularization to avoid overfitting.

### Contribution

This work proposes a quadrotor dynamics model that can accurately capture complex aerodynamic effects by *combining* a state-of-the-art rotor model based on BEM theory with learned residual force and torque terms represented by a deep neural network. The resulting hybrid model benefits from the expressive power of deep neural networks and the generalizability of first-principles modeling. The latter reduces the need for extreme amounts of training data. The model is identified using data collected from a large set of maneuvers performed on a real quadrotor platform. Leveraging one of the biggest

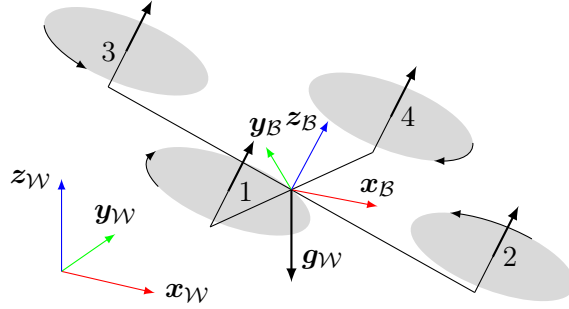
optical tracking volumes in the world, the platform’s state as well as the motor speeds are recorded during flight. The resulting dataset contains 96 flights with a cumulative time of 1h 15min and 1.8 million data points, covering the entire performance envelope of the platform up to observed speeds of  $65 \text{ km h}^{-1}$  ( $18 \text{ m s}^{-1}$ ) and accelerations of  $46.8 \text{ m s}^{-2}$ .

The proposed model is compared against state-of-the-art modeling approaches on unseen test maneuvers. The comparison is done in terms of both evaluation of predicted aerodynamic forces and torques and closed-loop integration of the model in a simulator, each evaluated against real-world reference data. In both categories, a performance increase by a factor of two is observed.

## J.2 Related Work

Traditionally, a rotor is assumed to produce thrust and axial torque proportional to the square of its angular rate with a constant coefficient [218], which is referred to hereinafter as the simple quadratic model. While these assumptions are valid for rotors on a static thrust stand and for near-hover flight, they do neither account for the case where the rotors move through air, nor for rotor-to-rotor and rotor-to-body interactions. Nevertheless, due to its simplicity, this model is still used in well-known aerial robotics simulators such as AirSim [309], Flightmare [320], RotorS [102] and others [229]. To improve the accuracy of the thrust model in non-stationary flights over the simple quadratic model, momentum theory has been used in [133, 141, 134]. Blade element theory is another approach to model a single rotor more accurately. The forces and torques acting on each infinitesimal portion of the blade are integrated over the whole propeller [276]. This theory has been adopted to model aerodynamic effects on a quadrotor in many studies [260, 36, 173, 338, 275]. However, both blade-element theory and momentum theory require the value of the induced velocity which is challenging to estimate. Hence, the blade-element-momentum (BEM) theory is proposed, which combines the above two theories to alleviate the difficulty of calculating the induced velocity. The resulting model can accurately capture aerodynamic forces and torques acting on single rotors in a wide range of operating conditions [176, 110, 111].

Even though BEM outperforms simple quadratic models and often achieves accurate predictions, it does not account for any interaction between the flow tubes of different propellers or the frame [331]. Previous work has incorporated interaction effects using either static wind tunnel tests [293, 305, 19] where the vehicle is rigidly mounted on a force sensor, or by performing fast maneuvers in instrumented tracking volumes [346]. In [346] a simple quadratic model is combined with residual forces predicted by Gaussian Processes. While this approach offers a lightweight solution to learn residual forces and can be used for control, it does not model residual torques, effectively neglecting moments caused by rotor-to-rotor interactions. In [331], the quadrotor platform is identified using a gray-box model that uses a library of polynomials as basis functions and is able to model both aerodynamic forces and torques. This method relies on the predefined function library and also contains discontinuities in the learned model. Another line of works investigates the modeling of quadrotors using computational fluid dynamics (CFD) [350, 213]. While such simulations achieve results that are highly accurate and manage to capture real-world



**Figure J.3** – Diagram of the quadrotor model depicting the world and body frames and illustrating the propeller numbering convention.

effects well, they require large amounts of computation time on high-performance compute clusters.

Due to their ability to identify patterns in large amounts of data, deep neural networks represent a promising approach to model aerodynamic effects precisely and computationally efficient. A recent line of works employs deep neural networks to learn quadrotor dynamics model purely from data, for both continuous time formulations [17, 277] as well as discrete-time formulations [231, 232, 312, 274]. While approaches relying entirely on learning-based methods have high representative power and the potential to also learn complex interaction effects, they require large amounts of data to train and careful regularization to avoid overfitting.

The approach presented in this work is inspired by [277, 17], but instead of learning the full dynamics, it combines state-of-the-art BEM modeling based on first principles with a data-driven approach to learn the residual force and torque terms. The resulting model benefits from the strong generalization performance of traditional first-principle modeling and the flexibility of learning-based function approximation.

### J.3 Quadrotor Model

This section explains the hybrid quadrotor model proposed in this work. It starts by introducing the notation and the rigid body dynamics (Figure J.3), proceeds to explaining two approaches to single-rotor modeling of increasing complexity and concludes with the learned residual model. The hybrid structure of the model, illustrated in Figure J.2, consists of a *rotor model* and a learned correction.

#### J.3.1 Notation

Scalars are denoted in non-bold  $[s, S]$ , vectors in lowercase bold  $\mathbf{v}$ , and matrices in uppercase bold  $\mathbf{M}$ . World  $\mathcal{W}$  and Body  $\mathcal{B}$  frames are defined with orthonormal basis i.e.  $\{\mathbf{x}_{\mathcal{W}}, \mathbf{y}_{\mathcal{W}}, \mathbf{z}_{\mathcal{W}}\}$ . The frame  $\mathcal{B}$  is located at the center of mass of the quadrotor. A vector from coordinate  $\mathbf{p}_1$  to  $\mathbf{p}_2$  expressed in the  $\mathcal{W}$  frame is written as:  ${}_{\mathcal{W}}\mathbf{v}_{12}$ . If the vector's origin coincides with the frame it is described in, the frame index is dropped, e.g. the quadrotor position is denoted as  $\mathbf{p}_{\mathcal{W}\mathcal{B}}$ . Furthermore, unit quaternions  $\mathbf{q} = (q_w, q_x, q_y, q_z)$

with  $\|\mathbf{q}\| = 1$  are used to represent orientations, such as the attitude state of the quadrotor body  $\mathbf{q}_{WB}$ . Finally, full SE3 transformations, such as changing the frame of reference from body to world for a point  $\mathbf{p}_{B1}$ , can be described by  ${}_{\mathcal{W}}\mathbf{p}_{B1} = {}_{\mathcal{W}}\mathbf{t}_{WB} + \mathbf{q}_{WB} \odot \mathbf{p}_{B1}$ . Note the quaternion-vector product is denoted by  $\odot$  representing a rotation of the vector by the quaternion as in  $\mathbf{q} \odot \mathbf{v} = \mathbf{q}\mathbf{v}\bar{\mathbf{q}}$ , where  $\bar{\mathbf{q}}$  is the quaternion's conjugate.

### J.3.2 Quadrotor Dynamics

The quadrotor is assumed to be a 6 degree-of-freedom rigid body of mass  $m$  and diagonal moment of inertia matrix  $\mathbf{J} = \text{diag}(J_x, J_y, J_z)$ . The state space is thus 13-dimensional and its dynamics can be written as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}}_{WB} \\ \dot{\mathbf{q}}_{WB} \\ \dot{\mathbf{v}}_{WB} \\ \dot{\boldsymbol{\omega}}_B \end{bmatrix} = \begin{bmatrix} \mathbf{v}_{\mathcal{W}} \\ \mathbf{q}_{WB} \cdot \begin{bmatrix} 0 \\ \boldsymbol{\omega}_B/2 \end{bmatrix} \\ \frac{1}{m} \left( \mathbf{q}_{WB} \odot \underbrace{(\mathbf{f}_{\text{prop}} + \mathbf{f}_{\text{res}})}_{:=\mathbf{f}} \right) + \mathbf{g}_{\mathcal{W}} \\ \mathbf{J}^{-1} \left( \underbrace{\boldsymbol{\tau}_{\text{prop}} + \boldsymbol{\tau}_{\text{res}}}_{:=\boldsymbol{\tau}} - \boldsymbol{\omega}_B \times \mathbf{J}\boldsymbol{\omega}_B \right) \end{bmatrix}, \quad (\text{J.1})$$

where  $\mathbf{g}_{\mathcal{W}} = [0, 0, -9.81 \text{ m/s}^2]^\top$  denotes earth's gravity,  $\mathbf{f}_{\text{prop}}$  is the collective force produced by the propellers including any parasitic effects the rotor model can simulate (e.g. induced drag), and  $\mathbf{f}_{\text{res}}$  denotes residual forces that are not explained by the rotor model used. Similarly,  $\boldsymbol{\tau}_{\text{prop}}$  and  $\boldsymbol{\tau}_{\text{res}}$  are the cumulative torques acting on the platform due to the propellers and residual torques that are not explained by the rotor model.

$$\mathbf{f}_{\text{prop}} = \sum_i \mathbf{f}_i \quad (\text{J.2})$$

$$\boldsymbol{\tau}_{\text{prop}} = \sum_i \boldsymbol{\tau}_i + \mathbf{r}_{P,i} \times \mathbf{f}_i, \quad (\text{J.3})$$

where  $\mathbf{r}_{P,i}$  is the location of propeller  $i$  expressed in the body frame and  $\mathbf{f}_i, \boldsymbol{\tau}_i$  are the forces and torques generated by the  $i$ -th propeller. The rotor models aim at predicting accurate estimates of the single-rotor forces and torques  $\mathbf{f}_i, \boldsymbol{\tau}_i$ , as explained in the following sections. The force and torque effects of the fuselage, body, and rotor interaction are not explicitly modeled, but should be captured by the residual dynamics  $\mathbf{f}_{\text{res}}$  and  $\boldsymbol{\tau}_{\text{res}}$ , predicted by a neural network.

### J.3.3 Rotor Model: Quadratic

The simplest model for single propeller is a quadratic fit which assumes the thrust and torque produced by a single propeller to be proportional to the square of its rotational

rate (propeller speed)  $\Omega$ .

$$\mathbf{f}_i(\Omega) = \begin{bmatrix} 0 \\ 0 \\ c_{l,q} \cdot \Omega^2 \end{bmatrix} \quad \boldsymbol{\tau}_i(\Omega) = \begin{bmatrix} 0 \\ 0 \\ c_{d,q} \cdot \Omega^2 \end{bmatrix} \quad (\text{J.4})$$

The coefficients  $c_{l,q}$  and  $c_{d,q}$  are typically identified using a static propeller test stand. This simplified model is a good approximation for near-hover flight at near-zero velocity without ceiling or ground effects [275] and explains static thrust-test stand measurements very well. However, it ignores that ego-motion impacts the lift generated by the propeller. The induced drag, which depends on the propeller speed and body-relative air velocity, is neglected as well, albeit being the dominant source of drag for quadrotors. This is sometimes mitigated by combining the model with a linear drag term such as in [102, 78].

### J.3.4 Rotor Model: BEM

Compared to the quadratic model, Blade-Element-Momentum-Theory (BEM) accounts for the effects of varying relative air speed on the rotor thrust. It assumes interaction effects between individual rotors to be negligible and describes each rotor separately. The approach presented here is based on classical propeller modeling for helicopters [276]. The modeling of the propeller lift and drag coefficients is based on [110, 72].

First, basic momentum theory is introduced and used to relate the thrust force to a given velocity difference in a flow-tube across the rotor. Then the aerodynamic blade-element model is presented. While momentum theory uses the momentum conservation to relate induced airspeed and generated thrust, a blade-element model sums the contributions of infinitesimal blade elements to the total thrust force and drag torque. Finally, the full algorithm combining momentum theory and the blade-element model is presented.

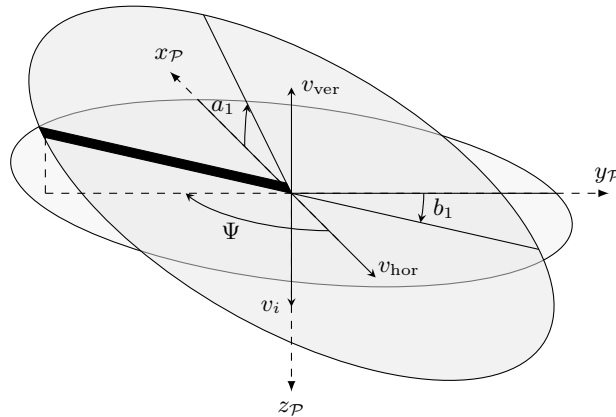
For now, the induced velocity  $v_i$  is considered to be known as momentum theory and the blade element model jointly yield this information. Together with the known ego-motion of the quadcopter, this fully determines the wind field around the individual propellers.

**Momentum Theory.** The most simple theory for analyzing rotors is momentum theory as it allows to calculate the thrust of a propeller based on a momentum balance across the rotor. This balance is done inside a flow-tube with radius  $R$  that fully contains the propeller. Assuming a known and constant induced velocity  $v_i$  across the diameter of the flow tube, the thrust  $T$  of a rotor is given by [276]:

$$T = 2v_i\rho A\sqrt{v_{\text{hor}}^2 + (v_{\text{ver}} - v_i)^2}, \quad (\text{J.5})$$

where  $v_{\text{hor}}$  and  $v_{\text{ver}}$  denote the horizontal and vertical velocity component of the flow tube. Note that momentum theory alone does not provide any means for calculating the induced velocity, it merely relates the induced velocity and the thrust based on a momentum balance and does not make any assumption on the physical process that actually accelerates the air.





**Figure J.4** – Lateral flapping  $b_1$  and longitudinal flapping  $a_1$  occur due to lift imbalance. The azimuth angle  $\Psi$  of the blade is measured ‘from the tail’ in the direction of rotation. The horizontal velocity  $v_{\text{hor}}$  and vertical velocity  $v_{\text{ver}}$  are defined opposite to the propeller frame, i.e. if the propeller moves along  $z_{\mathcal{P}}$  the relative velocity will also have a positive vertical component. Note that coning is not shown to improve the clarity of the schematic.

**Blade Element Theory.** The main purpose of a blade element model is to estimate the acting forces and torques accurately. A propeller consists of  $b$  identical blades (typically  $b = 2$  or  $b = 3$ ) attached to the rotor hub, each acting as a wing producing lift and drag forces. The propeller coordinate frame  $\mathcal{P}$  shown in Figure J.4 is defined such that the  $z_{\mathcal{P}}$ -axis points down and the  $x_{\mathcal{P}}$ -axis opposes the horizontal component  $v_{\text{hor}}$  of the incoming wind.

The finite stiffness of the blade and its hub-mount allow it to bend and deform, transmitting forces and introducing torques around the hub. It also causes the rotor-disk plane to be tilted with respect to the propeller coordinate system. This deformation can be split into a symmetric *coning* component  $a_0$  due to the overall lift produced by the propeller (not shown in Figure J.4) and an asymmetric *flapping* component that depends on the azimuth angle  $\Psi$  of the propeller. Figure J.4 illustrates this: in forward flight one side of the propeller experiences a higher relative airspeed (advancing blade) compared to the opposite side (retracting blade). Thus, there is a lift imbalance between the sides of the propeller which in turn causes the elastic propeller to bend upwards on the advancing side. This is called lateral flapping with the associated flapping angle  $b_1$ . Due to the inertia of the blade, the lateral flapping also induces a longitudinal flapping angle  $a_1$ .

Figure J.5 shows the velocities, angles and forces of a blade element. The chord of the airfoil is rotated by an angle  $\theta$  relative to the  $xy$ -plane. Together with the inflow angle  $\varphi$ , this results in a total angle of attack  $\alpha = \theta + \varphi$ . Each blade element produces a lift force  $dL$  perpendicular to the incoming airstream and a drag force  $dD$  in the direction of the incoming airflow. The thrust force  $dT$  and horizontal force  $dH$  are aligned with the propeller coordinate frame.

The tangential velocity  $U_T$  and parallel-to-motor velocity  $U_P$  are related to the angular

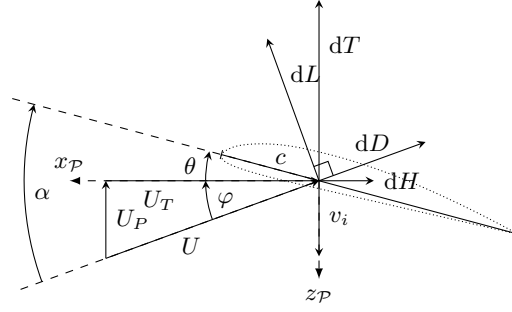


Figure J.5 – A blade element located at radius  $r$  and azimuth angle  $\Psi$ .

velocity  $\Omega$  of the propeller as

$$U_T(r, \Psi) = \Omega r + v_{\text{hor}} \sin \Psi \quad (\text{J.6})$$

$$U_P(r, \Psi) = v_{\text{ver}} - v_i - r\Omega(a_1 \sin \Psi + b_1 \cos \Psi) + v_{\text{ver}}(a_0 - a_1 \cos \Psi - b_1 \sin \Psi) \cos \Psi . \quad (\text{J.7})$$

The local angle of attack  $\alpha$  can be calculated as

$$\varphi(r, \Psi) = \arctan(U_P(r, \Psi)/U_T(r, \Psi)) \quad (\text{J.8})$$

$$\alpha(r, \Psi) = \theta_0 + \frac{r}{R}\theta_1 + \varphi(r, \Psi) , \quad (\text{J.9})$$

where  $\theta_0$  is the pitch angle of the blade and  $\theta_1$  the blade twist. The differential lift force  $dL$  and the differential drag  $dD$  can be expressed as functions of radius  $r$  and azimuth angle  $\Psi$ :

$$dL(r, \Psi) = c(r)c_l(\alpha(r, \Psi))(U_T(r, \Psi)^2 + U_P(r, \Psi)^2) \quad (\text{J.10})$$

$$dD(r, \Psi) = c(r)c_d(\alpha(r, \Psi))(U_T(r, \Psi)^2 + U_P(r, \Psi)^2), \quad (\text{J.11})$$

where  $c(r)$  is the chord length and  $c_l(\alpha)$ ,  $c_d(\alpha)$  are the angle-of-attack dependent coefficients of lift and drag respectively. The coefficients are modeled as proposed in [110, 72] as

$$c_d(\alpha) = c_{d,0} \sin^2 \alpha \quad c_l(\alpha) = c_{l,0} \sin \alpha \cos \alpha , \quad (\text{J.12})$$

where  $c_{d,0}$  and  $c_{l,0}$  are experimentally determined by measuring the lift and drag torque on a thrust test stand.

The overall thrust  $T$ , horizontal force  $H$  and drag torque  $Q$  are obtained through

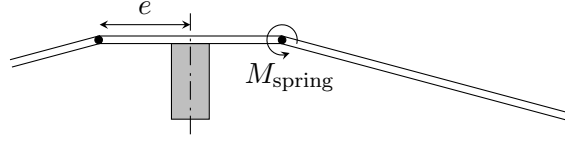


Figure J.6 – Illustration of the hinged blade model.

integration.

$$T = \frac{b\rho}{4\pi} \int_0^R \int_0^{2\pi} dL \cos \phi + dD \sin \phi \, d\Psi \, dr \quad (\text{J.13})$$

$$H = \frac{b\rho}{4\pi} \int_0^R \int_0^{2\pi} (-dL \sin \phi + dD \cos \phi) \sin \Psi \, d\Psi \, dr \quad (\text{J.14})$$

$$Q = \frac{b\rho}{4\pi} \int_0^R \int_0^{2\pi} (-dL \sin \phi + dD \cos \phi) r \, d\Psi \, dr \quad (\text{J.15})$$

**Blade Elasticity.** Standard helicopters have their rotor blades connected to the rotor hub through a hinge pin, optionally with an offset. This is not true for small rotor sizes typically found on multicopters, since the blade is fixed and elastic which breaks the assumptions made in standard helicopter literature. Therefore, the model is slightly adapted to capture the characteristics of small propellers: a blade is rigid and connected to the rotor hub with a hinge *and a torsional spring* at an offset  $e$  [133] as shown in Figure J.6. The coning angle  $a_0$  as well as the flapping angles  $a_1$  and  $b_1$  can be calculated by equating the moments acting on the rotor hub. At the hinge position, the following moment equilibrium occurs:

$$0 = M_w + M_{\text{gyro}} + M_{\text{inertial}} + M_{\text{cf}} + M_{\text{aero}} + M_{\text{spring}} , \quad (\text{J.16})$$

where the moment  $M_w$  is caused by the weight of the blade, the moment  $M_{\text{gyro}}$  is due to gyroscopic effects the blade experiences when a non-zero rollrate or pitchrate are present,  $M_{\text{inertial}}$  comes from the inertia of the blade and its angular acceleration during the flapping motion,  $M_{\text{cf}}$  is caused by centrifugal forces when the blade flaps, the moment  $M_{\text{aero}}$  is a result of the lift generated by the blade, and lastly  $M_{\text{spring}}$  is the restoring moment produced by the hinge spring. For brevity, the derivation of the coning and flapping angles are omitted here. They closely follow [276] (pp. 463). Due to the torsional spring at the hinge, the spring moment needs to be considered additionally:

$$M_{\text{spring}} = k_\beta (a_0 + a_1 \cos \Psi + b_1 \sin \Psi) , \quad (\text{J.17})$$

where  $k_\beta$  is the given spring stiffness. From (J.16), the coning and flapping angles are calculated. The resulting expression is omitted here for readability.

**Complete BEM-Model Algorithm.** Throughout above explanations, the induced velocity  $v_i$  was treated as a known quantity. However, when simulating the vehicle the induced velocity is unknown and needs to be calculated. This can be done by combining

## Appendix J. NeuroBEM: Hybrid Aerodynamic Quadrotor Model

---

the results from momentum theory and blade-element theory: (J.5) and (J.13) can both be used to calculate the thrust of the propeller. Due to the relatively stiff blade, the flapping and coning angle are small (typically less than  $1^\circ$ ) and can thus be neglected as a first approximation to calculate the induced velocity.

Due to the nature of the physical modeling process, the results from the momentum theory are only valid if the vehicle does not fly in its own downwash. This occurs when the vehicle descends with a certain speed, e.g. the propeller is in vortex-ring state [133] if and only if

$$0 < \frac{v_{\mathcal{P},z}}{v_i} < 2. \quad (\text{J.18})$$

If the results from momentum-theory are not applicable, the induced velocity can not be calculated. In [133] a solution is presented which relies on an empirical fit to calculate the induced velocity in such flight conditions. The proposed fit consists of a quartic polynomial to approximate  $v_i$  as follows:

$$\begin{aligned} \tilde{v}_i = v_{h,i} & \left( 1 + 1.125(v_{\mathcal{P},z}/v_{h,i}) - 1.372(v_{\mathcal{P},z}/v_{h,i})^2 \right. \\ & \left. + 1.718(v_{\mathcal{P},z}/v_{h,i})^3 - 0.655(v_{\mathcal{P},z}/v_{h,i})^4 \right), \end{aligned} \quad (\text{J.19})$$

where  $v_{h,i}$  is the induced velocity if the vehicle would fly horizontally in the given flight state, i.e. set  $v_{\mathcal{P},z} = 0$ . To ensure a smooth transition back to the physical modeling, the final induced velocity in vortex ring state is given as  $v_i = \max(\tilde{v}_i, v_{h,i})$ .

The algorithm thus consists of the following steps:

1. Assume  $a_0 = 0, a_1 = 0, b_1 = 0$ .
2. Find  $v_i$  such that (J.5) and (J.13) are simultaneously satisfied, i.e. the thrust calculated by momentum theory and blade element theory are identical. If inside vortex-ring state, use the approximation presented above.
3. Calculate the coning and flapping angle  $a_0, a_1$  and  $b_1$  with the previously computed induced velocity.
4. Using the previously calculated induced velocity and blade flapping angles, (J.13) – (J.15) can be evaluated again.
5. The total force of the propeller and torque around the center of the propeller are given by:

$$\mathbf{f}_{\mathcal{P}} = \begin{bmatrix} -(H + \sin a_1 T) \\ \pm \sin b_1 T \\ -T \cos a_0 \end{bmatrix} \quad \boldsymbol{\tau}_{\mathcal{P}} = \begin{bmatrix} \pm k_{\beta} b_1 \\ k_{\beta} a_1 \\ \mp Q \end{bmatrix},$$

where the upper sign of  $\pm$  and  $\mp$  corresponds to propeller rotating clockwise and the lower sign needs to be used for a counter-clockwise spinning propeller.

**Table J.1** – Comparison of different network architectures with respect to RMSE of force and torque prediction on a held-out test set.

Architecture	Force RMSE [N]	Torque RMSE [Nm]	# Param
TCN small	0.365	$6.525 \times 10^{-3}$	12k
TCN medium	0.352	$5.274 \times 10^{-3}$	25k
TCN large	0.355	$4.674 \times 10^{-3}$	72k
MLP	0.356	$5.172 \times 10^{-3}$	30k

### J.3.5 Learned Residual Dynamics

Both rotor models presented in Sections J.3.3 and J.3.4 do not account for aerodynamic forces and torques caused by the quadrotor body or interaction effects between the propellers. In this work, these residual dynamics are approximated by a deep neural network. Modeling such effects accurately requires to implicitly estimate the airflow around the vehicle. Considering the airflow as hidden state of the system, it can be estimated by measuring a history of observable state variables. In this work, the angular and linear velocities, as well as motor speeds are used as input features for the neural network. A history length of  $h = 20$ , with temporary equally-spaced samples with a  $\delta t = 2.5$  ms is used, effectively giving information of the platform evolution over the past 50 ms.

The network architecture is empirically validated by minimizing the prediction error on an unseen test set. The candidate architectures consist of temporal-convolutional (TCN) encoders [259] and fully-connected (MLP) encoders, which both are combined with two fully-connected heads, one for the residual force prediction and one for the residual torque prediction. Each architecture uses leaky-ReLU activations and a linear output layer. Training is performed in a supervised fashion using the Adam optimizer by minimizing the RMSE loss on forces and torques between predictions and labels. Table J.1 shows the main results of these ablation experiments. Due to its favourable performance versus inference time trade off, the medium-sized temporal-convolutional encoder (TCN-medium) was selected for all subsequent experiments.

## J.4 Experimental Setup

### J.4.1 Data Collection

To train the model and to verify its accuracy, real world measurement data is needed. It is recorded in a flying arena equipped with a motion tracking system with a usable volume of  $25 \text{ m} \times 25 \text{ m} \times 8 \text{ m}$ . The Vicon<sup>1</sup> motion tracking system allows to record accurate position and attitude measurements at 400 Hz. Additionally, onboard IMU measurements and motor speeds are recorded at 1 kHz by the low-level flight controller. This onboard data and the pose measurements need to be synchronized and fused in post processing. For this purpose interpolating cubic splines are fitted to the datapoints, which allows fusing

<sup>1</sup><https://www.vicon.com/>

the asynchronous measurements from both data sources, and recovers the full dynamic state. Furthermore, to estimate the unobserved linear velocity and angular acceleration, differentiation of the fitted splines provides less noisy estimates than direct differentiation of the discrete, noisy measurements. For the means of time synchronization, offset and clock skew are estimated through the correlation quality of the axis-wise angular rate measurement from the IMU with the spline. Gyroscope measurements are used because they provide better noise characteristics than the accelerometer data. The clock skew was typically observed to be 2.4%. The motor data is smoothed with a finite-impulse-response fourth-order Butterworth low-pass filter with a cutoff frequency corresponding to the time-constant of the motors, identified from the step response of the motors. This ensures that noise is suppressed without attenuating high-frequency motor signals more than 3 dB.

The resulting dataset contains 1.8 million data points recorded from 96 flights covering 1h:15min of flight time. The dataset is split into 70% training, 20% validation, and 10% test set. Each subsets contains trajectories that cover the full range of speeds and accelerations observed in the full data set.

### J.4.2 Quadrotor Platform

The real-world flights are performed with a custom-made quadrotor platform. It features an Armattan Chameleon 6 inch main frame, equipped with Hobbywing XRotor 2306 motors and 5 inch, three-bladed propellers. The platform has a total weight of 752 g and can produce a maximum static thrust of approximately 33 N, which results in a static thrust-to-weight ratio of 4.5. 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. In all real world flights, control commands in the form of collective thrust and bodyrates are computed on a laptop computer and sent via a Laird module to the Jetson TX2. The Jetson then forwards these commands to a commercial flight controller running BetaFlight<sup>2</sup>, which produces single-rotor commands that are fed to a 4-in-1 electronic speed controller.

### J.4.3 Control System

Control commands are produced by a control pipeline consisting of two levels: (i) a high-level non-linear quadratic MPC controller generating bodyrate and collective thrust commands at 100 Hz, and (ii) a low-level Betaflight controller tracking the desired bodyrate setpoint at 1 kHz. To ensure repeatability, *BetaFlight* features targeted at human piloted drones (feed-forward terms) are disabled, and the controller is reduced to a PID with equal parameters for simulation and real-world flight. BetaFlight is run as a software module within the simulation, resembling the real control system. Both controllers are kept equal in the simulation with respect to the real-world experiments, to guarantee equal performance in both scenarios.

---

<sup>2</sup><https://github.com/betaflight/betaflight>

#### J.4.4 Simulator Extension

To compare different models, their resulting simulation accuracy is evaluated with respect to the real-world trajectory. For this purpose, the closed-loop system is simulated forward in time. While the rigid-body dynamics are given by (J.1) and the aerodynamic force and torque are provided by the model in question, there are two further components needed for an accurate simulation: integration and motor dynamics.

**Integration.** The integration is performed by a symplectic Euler scheme with a timestep of 1 ms using the rigid body dynamics (J.1), the modeled linear and angular accelerations of the tested model, and the motor dynamics explained in the following section. The advantage of the symplectic Euler scheme is its energy conservation property, which is invalidated with other integration schemes, such as the standard Euler methods or the Runge-Kutta family of integrators.

**Motor Dynamics.** Since the aerodynamic model is based on the angular speed of the propeller, the motors are modeled as a first-order system according to

$$\frac{\delta}{\delta t} \Omega = \frac{1}{\tau_{\Omega}} (\Omega_{cmd} - \Omega) \tag{J.20}$$

where  $\Omega_{cmd}$  is the commanded propeller speed and  $\tau_{\Omega}$  is the motor time constant. For the quadrotor platform used throughout the experiments, the time constant was identified to be  $\tau_{\Omega} = 33$  ms.

## J.5 Experiments and Results

The evaluation procedure is designed to address the following questions: (i) When does the classical approach to quadrotor modeling based on quadratic thrust and torque curves start to break down? (ii) How do the forces and torques predicted by a model based on BEM compare with respect to a simple quadratic model? (iii) What is the contribution of a learned residual dynamics component? The reader is encouraged to watch the attached video to understand the highly dynamic nature of the experiments.

### J.5.1 Experimental Setup

Throughout the experiments multiple models are fitted using the training and validation data, and evaluated using the test data. The predictive performance of these models is compared in two different settings, covered in the subsequent sections: in Section J.5.2, the RMSE of predicted forces and torques is compared on unseen flight data; in Section J.5.3, different dynamics models are evaluated in conjunction with a known controller to determine the mismatch between simulation and the real world in a closed-loop scenario. Both types of comparisons are performed on unseen test trajectories that cover the entire performance envelope of the platform. Each trajectory has been performed on the real platform in an instrumented tracking volume as explained in Section J.4.

## Appendix J. NeuroBEM: Hybrid Aerodynamic Quadrotor Model

**Table J.2** – Comparison of model performance in terms of RMSE on an unseen test set. Approaches marked with an asterisk are trained on a reduced training set to compare generalization performance.

Model	$F_{xy}$ [N]	$F_z$ [N]	$M_{xy}$ [Nm]	$M_z$ [Nm]	$F$ [N]	$M$ [Nm]
None	1.549	13.618	0.036	0.006	7.964	0.029
Fit	1.536	1.381	0.104	0.033	1.486	0.087
BEM	0.803	1.265	0.090	0.017	0.982	0.074
PolyFit [331]	0.453	0.832	0.027	0.008	0.606	0.022
None+NN	0.236	0.681	0.017	<b>0.002</b>	0.438	0.014
Fit+NN	0.232	0.722	0.017	0.004	0.458	0.014
BEM+NN (ours)	<b>0.204</b>	<b>0.504</b>	<b>0.014</b>	0.004	<b>0.335</b>	<b>0.012</b>
PolyFit* [331]	1.450	6.637	2.815	0.164	4.011	2.301
None+NN*	0.470	1.959	<b>0.007</b>	<b>0.002</b>	1.194	<b>0.006</b>
Fit+NN*	0.501	1.225	0.024	0.013	0.817	0.021
BEM+NN* (ours)	<b>0.344</b>	<b>0.816</b>	0.025	0.008	<b>0.549</b>	0.021

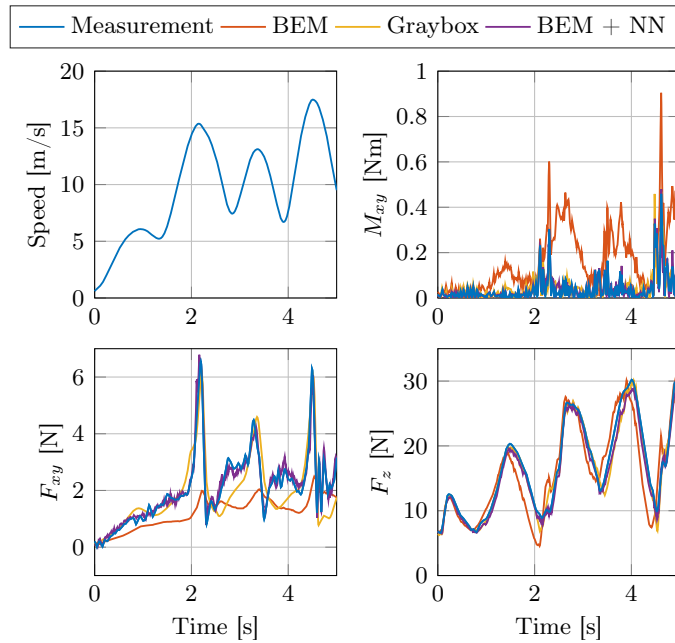
### J.5.2 Comparison of Predictive Performance

In a first set of experiments, the presented models are compared in terms of predicted forces and torques on unseen trajectories. These trajectories cover the entire performance envelope of the quadrotor, ranging from slow near-hover trajectories with speeds below  $5 \text{ m s}^{-1}$  to aggressive trajectories at the limit of the platform’s capabilities, exceeding speeds of  $18 \text{ m s}^{-1}$  and accelerations up to  $46.8 \text{ m s}^{-2}$ . The models compared in this experiment consist of the quadratic model (*Fit*), the BEM model (*BEM*) and a naive model predicting all zeros (*None*). Each of these models is augmented with a learned residual correction using a neural network, marked with *+NN*. While the *None* model represents a naive baseline to better understand the magnitude of prediction errors, *None+NN* illustrates the performance of a purely learned model. Finally, the approach presented in [331] is compared, denoted as *PolyFit* as it uses automatically selected polynomial basis functions to fit the model.

To evaluate generalization performance, each approach is trained on two datasets: the entire training set as explained in Section J.4 and a reduced dataset that only covers linear speeds up to  $5 \text{ m s}^{-1}$ . This reduced dataset is used to identify new parameters for each approach, which are marked with an asterisk.

Table J.2 summarizes the results of this experiment, while Figure J.7 illustrates performance on a highly aggressive maneuver. The proposed hybrid model based on BEM and learned residual dynamics consistently outperforms all other models on the predicted forces. Note that the trajectories performed in this work are designed to minimize yaw rate, and as a result only cover extremely small yaw torques  $M_z$  (the largest yaw torque in the dataset is  $0.072 \text{ Nm}$ ), as discussed in Section J.6. It is evident that thrust and torque along the body  $z_B$ -axis are more challenging to predict accurately. The reason for this

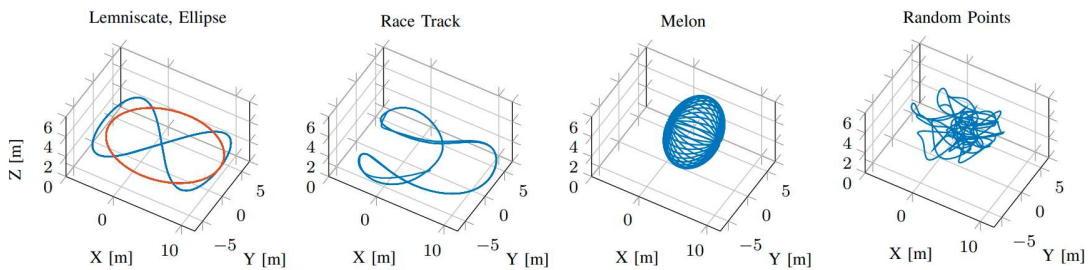




**Figure J.7** – The plot illustrates the results presented in Table J.2. The plots show a highly aggressive maneuver (from the test dataset) where only models with neural net augmentation predict the forces and torques well.

is two-fold: First, all linear acceleration actuation lies in the  $z_B$  direction, contributing actuation noise predominantly along this axis. Second, all trajectories are designed to minimize yaw torque, since this is the least actuated torque direction, and significantly limits the acceleration envelope, and therefore the attainable agility. This explains the good performance of even very naive baselines with respect to this metric.

When comparing on the reduced dataset (Table J.2, bottom), the performance of the proposed approach gracefully degrades, still outperforming the baselines in terms of predicted forces. Note that the *PolyFit* baseline completely breaks down in this setting, indicating poor generalization to unseen data. The purely learning-based baseline outperforms all other approaches in the predicted torques, but also fails to generalize the force predictions to the new data.



**Figure J.8** – Trajectories used for testing. Each trajectory is flown multiple times with varying speeds.

## Appendix J. NeuroBEM: Hybrid Aerodynamic Quadrotor Model

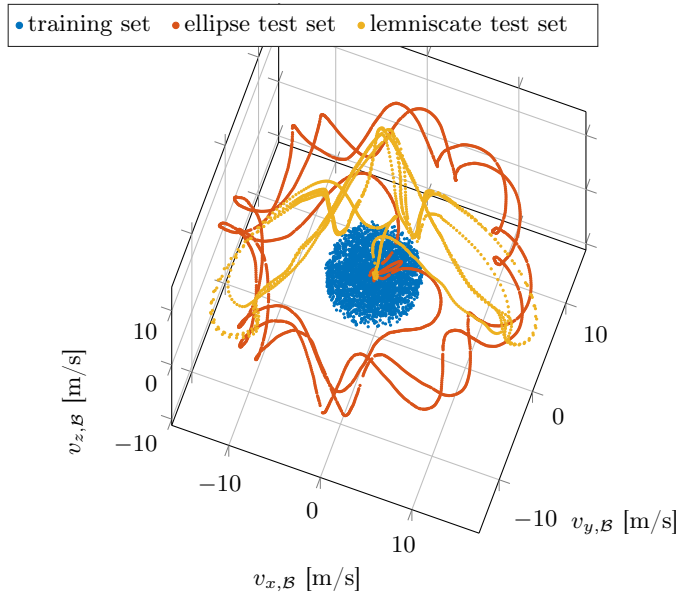
**Table J.3** – Comparison of closed-loop simulation performance on an unseen test set of different trajectories. Results show the positional RMSE between trajectories flown in simulation with different dynamics models and the same set of trajectories flown on the real platform. Models marked with an asterisk (\*) were trained only on slow data up to  $5 \text{ m s}^{-1}$ .

	$v_{\text{mean}}$ [m/s]	$v_{\text{max}}$ [m/s]	Fit	BEM	PolyFit [331]	None+NN	Fit+NN	BEM+NN (ours)	PolyFit*	None+NN*	Fit+NN*	BEM+NN* (ours)	RotorS [102]
Lemnis.	1.7	3.5	0.061	0.059	<b>0.043</b>	0.046	0.049	0.059	<b>0.046</b>	0.049	0.048	0.053	0.11
Random	2.3	8.2	0.167	0.138	0.130	crash	<b>0.126</b>	0.141	crash	0.130	<b>0.123</b>	0.134	0.24
Lemn.	3.2	7.0	0.183	0.146	<b>0.102</b>	0.103	0.112	0.109	crash	<b>0.100</b>	0.112	0.112	0.15
Melon	3.5	7.6	0.229	0.163	<b>0.117</b>	0.126	0.126	0.133	crash	<b>0.127</b>	0.134	0.137	0.19
Sl. Circ.	6.9	10.7	0.381	0.232	<b>0.166</b>	0.172	0.168	0.167	crash	0.210	0.193	<b>0.185</b>	0.26
Lin. Osc.	7.2	16.9	0.506	0.438	0.172	0.270	0.234	<b>0.171</b>	crash	0.258	0.263	<b>0.216</b>	0.56
Race	7.6	13.1	0.414	0.286	0.233	0.283	0.223	<b>0.214</b>	crash	0.257	0.262	<b>0.240</b>	0.42
Melon	7.7	13.5	0.431	0.221	0.239	0.179	0.164	<b>0.155</b>	crash	0.263	0.207	<b>0.185</b>	0.40
Sl. Circ.	8.5	13.3	0.531	0.217	0.255	0.206	0.197	<b>0.192</b>	crash	0.340	0.268	<b>0.180</b>	0.40
Race	9.9	17.8	0.617	0.408	0.820	0.447	0.320	<b>0.301</b>	crash	0.446	0.420	<b>0.370</b>	0.71
Lemnis.	12.0	19.8	0.762	0.549	0.316	0.782	0.352	<b>0.286</b>	crash	0.469	0.423	<b>0.371</b>	1.16
Ellipse	15.0	19.2	0.855	0.369	crash	0.347	0.402	<b>0.285</b>	crash	0.605	0.481	<b>0.290</b>	0.65

### J.5.3 Closed-Loop Comparison

To demonstrate the benefits of an accurate force and torque model, a second set of experiments presents a comparison of closed-loop simulation performance. Using the simulation setup explained in Section J.4.4, a set of unseen trajectories (Figure J.8) is flown in simulation and the resulting flight path is compared with the data obtained from executing the same set of trajectories on the real platform. As for the previous set of experiments, also this comparison is performed for models identified on the full training set, as well as a reduced training set to compare generalization performance. Additionally to the baselines already used in the previous experiments, this experiment also compares against RotorS [102]. To do this, the standard model in RotorS is updated with the parameters identified from the real platform (i.e. mass, inertia, dimensions, lift and drag coefficients).

Table J.3 illustrates the results of the closed-loop experiment. For each trajectory, the accumulated positional error between the simulated flight and the data observed in the real world is reported. As can be seen, all models achieve similar performance for trajectories close to hover. The *Fit* model exhibits increasing bias with higher speeds, with the error exceeding the worst performance of the proposed approach already at average speeds below  $7 \text{ m s}^{-1}$ . In contrast, the *BEM* model is able to maintain competitive performance up to the fastest trajectories. At low speeds, the *PolyFit* baseline performs very well, but exhibits increasing bias for higher speeds, even resulting in a crash on the fastest trajectory. The RotorS baseline performs inferior on all trajectories, achieving results comparable to the *Fit* baseline. The proposed approach combining BEM with a learned residual term (*BEM+NN*) achieves competitive performance on the slow trajectories and



**Figure J.9** – Visualization of the reduced training set, the ellipse test set and the fastest lemniscate test set in the body-frame velocity space. Although the test set mostly covers regions of the state space that are not part of the training set, the trained BEM+NN\* model still provides good accuracy in simulating the trajectory. This demonstrates its remarkable generalization capability.

outperforms all baselines on the faster trajectories. Compared to *Fit+NN*, *BEM+NN* achieves consistently better performance for fast maneuvers.

When trained on the reduced dataset, all models show decreased performance. However, while approaches such as *PolyFit* completely break down, the proposed approach experiences only a minor performance reduction around 20%, outperforming the baselines on all faster trajectories. This result highlights the ability of the proposed approach to generalize beyond the data it was trained on (Figure J.9).

## J.6 Discussion

The results obtained in this work show that the proposed hybrid dynamics model, combining first-principles based on blade-element-momentum theory with a learning-based residual term, outperforms state-of-the-art modeling for quadrotors with a 50% decreased aerodynamic force and torque prediction error. Furthermore, evaluation in controlled experiments on a large real-world dataset shows that such a complementary modeling approach outperforms each of its compositional submodules.

In fact, not only does the performance of the proposed hybrid model structure improve with a more capable rotor model, but the learned residual dynamics also increase in accuracy if a broader envelope of effects can already be captured using first principles. Specifically, the learned residual prediction achieves up to 30% better performance when

## Appendix J. NeuroBEM: Hybrid Aerodynamic Quadrotor Model

---

combined with a rotor model based on blade-element-momentum theory.

While the proposed approach significantly improves upon state-of-the-art in quadrotor modeling for highly aggressive maneuvers by up to 60%, its advantages for slow speed trajectories below  $5 \text{ m s}^{-1}$  are limited. Our experiments indicate that for such slow trajectories, a traditional parametric approach such as [331] achieves very strong performance at a lower computational cost. While the quadratic and polynomial fits can be evaluated in a mere  $1 \mu\text{s}$ , even on a micro processor, the BEM model requires in the order of  $100 \mu\text{s}$  on a modern Intel-architecture CPU, and a forward pass of the network averages also at around  $100 \mu\text{s}$  on a modern NVidia GPU. The reason for the dominant runtime of the BEM model is the necessary implicit solution for the induced velocity equation. Even though our approach is not optimized for runtime, simulations can be run at an arbitrary timescale, where most applications gladly trade-off real-time evaluation for improved accuracy.

The results of the closed-loop simulation using the proposed model could be further improved by refining the following aspects of the control pipeline: (i) The experimental platform currently relies on the BetaFlight inner-loop low-level controller, which is optimized for human pilots. However, as such, it only takes a throttle command and a body rate command as inputs, and relies on the inner-loop to track the rate command. Furthermore, it performs filtering and interpolation of the control signals to ensure a consistent flight feeling for human pilots, which introduces undesirable control-loop shaping. An MPC outer-loop controller directly outputting single-rotor motor speeds that are tracked by an inner loop motor speed controller would improve the accuracy of our approach further as this would minimize the differences between the simulation and the actual experiments. (ii) Modeling the latency from the motion capture pose filtering, the data transmission to the drone, and the communication to the flight controller in simulation would also reduce the error as it improves the realism of the simulator. The authors expect the results in Table J.3 to be even more favorable for their approach in such an ideal setting. The accuracy of the force and torque predictions shown in Table J.2 would also benefit from a custom low-level controller providing more precise and less noisy motor-speed information.

Compared to a purely learning-based approach such as *None+NN*, the proposed approach performs 25% better for all non-trivial trajectories with average speeds above  $4 \text{ m s}^{-1}$ , and extrapolates well to unseen flight data, as opposed to e.g. the *PolyFit* baseline. The authors expect the performance of learning-only approaches to improve with more data. However, in a real world setting, where high-quality data is sparse, pure learning-based approaches fall short of traditional methods. Additionally, accurate aerodynamic force and torque prediction does not necessarily translate to good closed-loop performance, as demonstrated in our evaluation. Moreover, this study observed cases where a purely-learned residual component introduced a feedback loop on the predicted torques that led to a crash. In such cases, support through first-principles is vital for accurate and robust modeling.

## J.7 Conclusion

This work proposes a novel method to model quadrotors by combining modeling based on first principles with a learning-based residual term represented by a neural network. The proposed method is able to accurately model quadrotors even throughout aggressive trajectories pushing the platform to its limits. This hybrid model outperforms its compositional modules with up to 50% error reduction, including baseline methods that utilize only first-principles modeling, as well as purely learning-based methods. The method shows strong generalization beyond the training set used to identify the model and predicts accurate forces and torques where other methods break down. Controlled experiments indicate that the fusion of learned dynamics with first-principles is a powerful combination, where the learned dynamics-residual benefits from high-fidelity models, such as the BEM. Applied to simulations, our approach enables unprecedented accuracy, reducing positional RMSE from  $\sim 0.8$  m for state-of-the-art approaches, down to below 0.3 m. This could tremendously speed up development and testing of advanced control and navigation strategies for quadrotors, without the need of the tedious and crash-prone trial-and-error strategy on real systems.

## Acknowledgement

This work was supported by the National Centre of Competence in Research (NCCR) Robotics through the Swiss National Science Foundation (SNSF), the Intel Network on Intelligent Systems, the European Union’s Horizon 2020 Research and Innovation Programme under grant agreement No. 871479 (AERIAL-CORE) and the European Research Council (ERC) under grant agreement No. 864042 (AGILEFLIGHT).



# K Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

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

Philipp Foehn\*, Elia Kaufmann\*, Angel Romero, Robert Penicka, Sihao Sun, Leonard Bauersfeld, Thomas Laengle, Yunlong Song, Antonio Loquercio, and Davide Scaramuzza. “Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight”. In: *Science Robotics* (2021). under review

# Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

Philipp Foehn, Elia Kaufmann, Angel Romero, Robert Penicka, Sihao Sun,  
Leonard Bauersfeld, Thomas Laengle, Yunlong Song, Antonio Loquercio, Davide  
Scaramuzza

**Abstract** — Autonomous, agile quadrotor flight raises fundamental challenges for robotics research in terms of perception, planning, learning, and control. A versatile and standardized platform is needed to accelerate research and let practitioners focus on the core problems. To this end, we present Agilicious, a co-designed hardware and software framework tailored to autonomous, agile quadrotor flight. It is completely open-source and open-hardware and supports both model-based and neural-network-based controllers. Also, it provides high thrust-to-weight and torque-to-inertia ratios for agility, onboard vision sensors, GPU-accelerated compute hardware for real-time perception and neural-network inference, a real-time flight controller, and a versatile software stack. In contrast to existing frameworks, Agilicious offers a unique combination of flexible software stack and high-performance hardware. We compare Agilicious with prior works and demonstrate it on different agile tasks, using both model-based and neural-network-based controllers. Our demonstrators include trajectory tracking at up to 5 g and 70 km h<sup>-1</sup> in a motion-capture system, and vision-based acrobatic flight and obstacle avoidance in both structured and unstructured environments using solely onboard perception. Finally, we demonstrate its use for hardware-in-the-loop simulation in virtual-reality environments. Thanks to its versatility, we believe that Agilicious supports the next generation of scientific and industrial quadrotor research.



## K.1 Introduction

Quadrotors are extremely agile vehicles. Exploiting their agility in combination with full autonomy is crucial for time-critical missions, such as search and rescue, aerial delivery, and even flying cars. For this reason, over the past decade, research on autonomous, agile quadrotor flight has continually pushed platforms to higher speeds and agility [228, 203, 170, 233, 373, 92, 197, 253, 172, 91].

To further advance the field, several competitions have been organized—such as the autonomous drone racing series at the recent IROS and NeurIPS conferences [235, 53, 170, 217] and the AlphaPilot challenge [117, 92]—with the goal to develop autonomous systems that will eventually outperform expert human pilots. Million-dollar projects, such as AgileFlight[57] and Fast Light Autonomy (FLA)[233], have also been funded by the European Research Council and the United States government, respectively, to further push research.

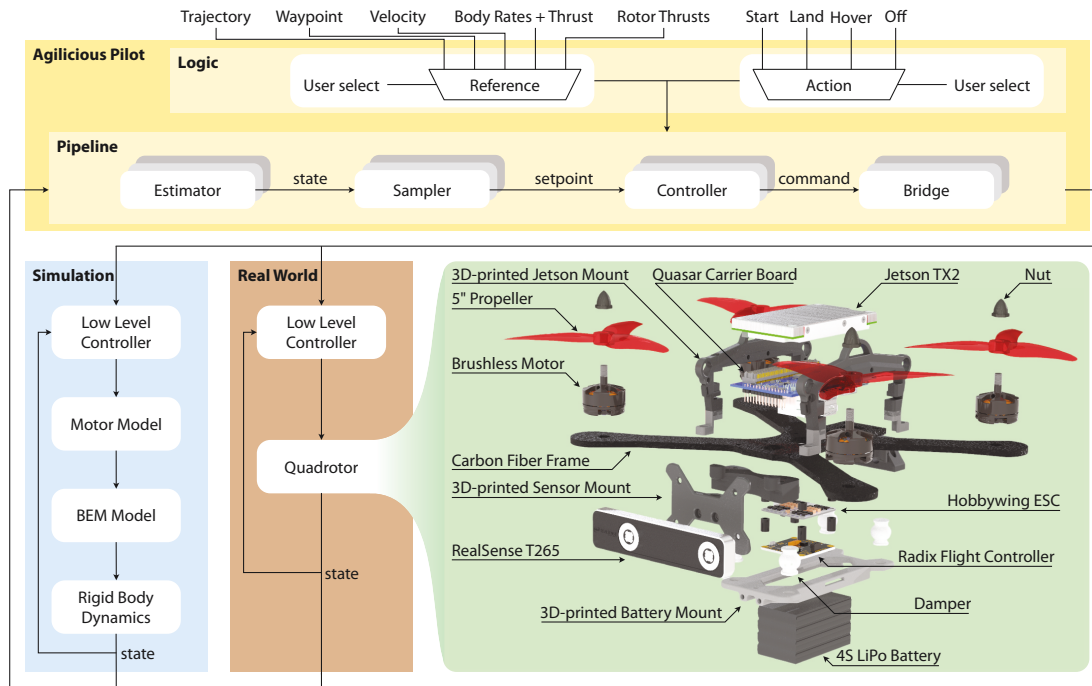
Agile flight comes with ever-increasing engineering challenges since performing faster maneuvers with an autonomous system requires more capable algorithms, specialized hardware, and proficiency in system integration. As a result, only a small number of research groups have undertaken the significant overhead of hardware and software engineering, and have developed the expertise and resources to design quadrotor platforms that fulfill the requirements on weight, sensing, and computational budget necessary for autonomous agile flight. This work aims to bridge this gap through an open-source agile flight platform, enabling everyone to work on agile autonomy with minimal engineering overhead.

The platforms and software stacks developed by research groups [226, 203, 233, 297, 78, 256, 14, 126] vary strongly in their choice of hardware and software tools. This is expected, as optimizing a robot with respect to different tasks based on individual experience in a closed-source research environment leads to a fragmentation of the research community. For example, even though many research groups use the Robot Operating System middleware to accelerate development, publications are often difficult to reproduce or verify since they build on a plethora of previous implementations of the authoring research group. In the worst case, building on an imperfect or even faulty closed-source foundation can lead to wrong or non-reproducible conclusions, slowing down research progress. To break this vicious cycle and to democratize research on fast autonomous flight, the robotics community needs an open-source and open-hardware quadrotor platform that provides the versatility and performance needed for a wide range of agile flight tasks. Such an open and agile platform does not yet exist, which is why we present Agilicious, an open-source and open-hardware agile quadrotor flight stack summarized in Figure K.1.

To reach the goal of creating an agile, autonomous, and versatile quadrotor research platform, three main design requirements must be met by the quadrotor: (i) it must carry the required compute hardware needed for autonomous operation, (ii) it must be crash resilient to allow fast prototyping, and (iii) must be capable of agile flight.

To meet the first requirement on computing resources needed for true autonomy, a quadro-

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight



**Figure K.1** – The Agilicious software and hardware quadrotor platform are tailored for agile flight while featuring powerful onboard compute capabilities through an NVIDIA Jetson TX2. The versatile sensor mount allows for rapid prototyping with a wide set of camera sensors such as the RealSense T265. As a key feature, the software of Agilicious is built in a modular fashion, allowing rapid software prototyping in simulation and seamless transition to real-world experiments. The Agilicious Pilot encapsulates all logic required for agile flight, while exposing a rich set of interfaces to the user, from high-level pose commands to direct motor commands. The software stack can be used in conjunction with a custom modular simulator which supports highly accurate aerodynamics based on blade-element momentum theory [23], or with RotorS [102], hardware-in-the-loop, and rendering engines such as Flightmare [320]. Deployment on the physical platform only requires selecting a different bridge and a sensor-compatible estimator.

tor must have sufficient compute capability to concurrently run estimation, planning, and control algorithms onboard. With the emergence of learning-based methods, also efficient hardware acceleration for neural network inference is required. As more compute typically entails increased size and weight, a conservative approach to fulfill the compute demand would negatively affect requirements (ii) and (iii), which try to maximize the platform's crash resilience and agility. When pushing quadrotors to their limits, research on agile flight sooner or later inevitably leads to crashes. To meet the second requirement on crash resilience, the platform must withstand such crashes without requiring major repairs. To satisfy the third requirement, the platform must deliver an adequate thrust-to-weight ratio and torque-to-inertia ratio. While the thrust-to-weight ratio can often be enhanced using sufficiently large motors, more powerful motors require larger propellers and thus larger size of the platform. However, the torque-to-inertia ratio typically decreases with higher weight and size, since the moment of inertia increases quadratic with the size, and linearly with the weight. Therefore, it is desirable to design a lightweight and small platform [187, 88] to maximize agility (i.e. maximize both thrust-to-weight ratio (thrust-to-weight) and torque-to-inertia ratio (torque-to-inertia)). These three design objectives require optimizing the platform to meet the best trade-off, since maximizing compute resources competes against the other two design goals. Apart from hardware design considerations, a quadrotor research platform needs to provide the software framework for flexible usage and reproducible research. This entails the abstraction of hardware interfaces and a general co-design of software and hardware necessary to exploit the platform's full potential. Such co-design must account for the capabilities and limitations of each system component, such as the complementary real-time capabilities of common operating systems and embedded systems, communication latencies and bandwidths, system dynamics bandwidth limitations, and efficient usage of hardware accelerators. In addition to optimally using hardware resources, the software should be built in a modular fashion to enable rapid prototyping through a simple exchange of components, both in simulation and real-world applications. This modularity enables researchers to test and experiment with their research code, without the requirement to develop an entire flight stack, accelerating time to development and facilitating reproducibility of results. Finally, the software stack should run on a broad set of computing boards, be efficient, easy to transfer and adapt by having minimal dependencies and provide known interfaces, such as the widely-used Robot Operating System (ROS).

The complex set of constraints and design objectives is difficult to meet. There exists a variety of previously published open-source research platforms, which, while well designed for low-agility tasks, could only satisfy a subset of the aforementioned hardware and software constraints. In the following section, we list and analyze prominent examples such as the FLA platform [233], the MRS quadrotor [14], the ASL-Flight [297], the MIT-Quad [336], the GRASP-Quad[203], or our previous work [78].

The FLA platform [233] relies on many sensors, including Lidars and laser-range finders in conjunction with a powerful onboard computer. While this platform can easily meet autonomous flight computation and sensing requirements, it does not allow to perform agile flight beyond 2.4g of thrust, limiting the flight envelope to near-hover flight. The MRS platform [14] provides an accompanying software stack and features a variety of sensors. Even though this hardware and software solution allows fully autonomous flight,

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

---

the actuation renders the system not agile with a maximum thrust-to-weight of 2.5. The ASL-Flight [297] is built on the DJI Matrice 100 platform and features an Intel NUC as the main compute resource. Similarly to the MRS platform, the ASL-Flight has very limited agility due to its weight being on the edge of the platform’s takeoff capability. The comparably smaller GRASP-Quad proposed in [203] operates with only onboard IMU and monocular camera while having a weight of only 250 g. Nevertheless, the used Qualcomm Snapdragon board lacks computational power and the reported maximal accelerations are only up to 1.5g. Motivated by drone racing, the MIT-Quad [336] reported accelerations of up to 2.1g while it was further equipped with NVIDIA Jetson TX2 in [13], however, it does not reach the agility of the Agilicious and contains proprietary electronics. Finally, the quadrotor proposed in [78] is a research platform designed explicitly for agile flight. Although the quadrotor featured a high thrust-to-weight ratio of up to 4, its compute resources are very limited, prohibiting truly autonomous operation. All these platforms are optimized for either relatively heavy sensor setups or for agile flight in non-autonomous settings. While the former platforms lack the required actuation power to push the state of the art in autonomous agile flight, the latter have insufficient compute resources to achieve true autonomy.

Finally, several mentioned platforms rely on either Pixhawk-PX4 [225], the Parrot [300] or DJI [69] low-level controllers, which are mostly treated as blackboxes. This, together with the proprietary nature of the DJI limits control over the low-level flight characteristics, which negatively impacts agility and predictability. Full control over the complete pipeline is necessary to truly understand aerodynamic and high-frequency effects, model and control them, and exploit the platform to its full potential.

Apart from platforms mainly developed by research labs, several quadrotor designs are proposed by industry (Skydio [317], DJI [69], Parrot [300]) and open-source projects (PX4 [225], Paparazzi [126], Crazyflie [109]). While Skydio [317] and DJI [69] both develop platforms featuring a high level of autonomy, they do not support interfacing with custom code and therefore are of limited value for research and development purposes. Parrot [300] provides a set of quadrotor platforms tailored for inspection and surveillance tasks that are accompanied by limited software development kits that allow researchers to program custom flight missions. In contrast, PX4 [225] provides an entire ecosystem of open-source software and hardware as well as simulation. While these features are extremely valuable especially for low-speed flight, both cross-platform hardware and software are not suited to push the quadrotor to agile maneuvers. Similarly, Paparazzi [126] is an open-source project for drones, which supports various hardware platforms. However, the supported autopilots have very limited onboard compute capability, rendering them unsuited for agile autonomous flight. The Crazyflie [109] is an extremely lightweight quadrotor platform with a takeoff weight of only 27 g. The minimal hardware setup leaves no margin for additional sensing or computation, prohibiting any non-trivial navigation task.

To address the requirements of agile flight, the shortcomings of existing works, and to enable the research community to progress fast towards agile flight, we present an open-source and open-hardware quadrotor platform for agile flight at more than 5g acceleration while providing excellent onboard compute and versatile software. The hardware design leverages recent advances in motor, battery, and frame design initiated by the first-person-

## K.1. Introduction

framework	open-source	simulation	onboard computer	low-level controller	CPU mark (higher is better)	GPU	maximum speed (0-100 km h <sup>-1</sup> time)	thrust/weight
PX4 [225]	SW and HW	✓	✗	custom open source	-	✗	-	-
Paparazzi [126]	SW and HW	✓	✗	custom open source	-	✗	-	-
DJI [69]	-	✗	✗	proprietary	-	✗	140 km h <sup>-1</sup> (2.0 s)	≈ 5.00
Skydio [317]	-	✗	✗	proprietary	-	✗	58 km h <sup>-1</sup>	-
Parrot [300]	SW	✗	✗	proprietary	-	✗	55 km h <sup>-1</sup>	-
Crazyflie [109]	SW and HW	✓	✗	custom open source	-	✗	-	≈ 2.26
FLA-Quad [233]	SW and HW	✗	✓	PX4	3,383	✗	-	≈ 2.38
GRASP-Quad [203]	-	✗	✓	custom	625	✗	-	≈ 1.80
MIT-Quad [13]	-	✗	✓	custom	1,343	✓	-	≈ 2.33
ASL-Flight [297]	SW and HW	✗	✓	DJI	3,383	✗	-	≈ 2.32
RPG-Quad [78]	SW and HW	✓	✓	Betaflight	633	✗	-	≈ 4.00
MRS UAV [14]	SW and HW	✓	✓	PX4	8,846	✗	-	≈ 2.50
<b>Agilicious (Ours)</b>	SW and HW	✓	✓	custom open source	1,343	✓	131 km h <sup>-1</sup> (1.01 s)	≈ 5.00

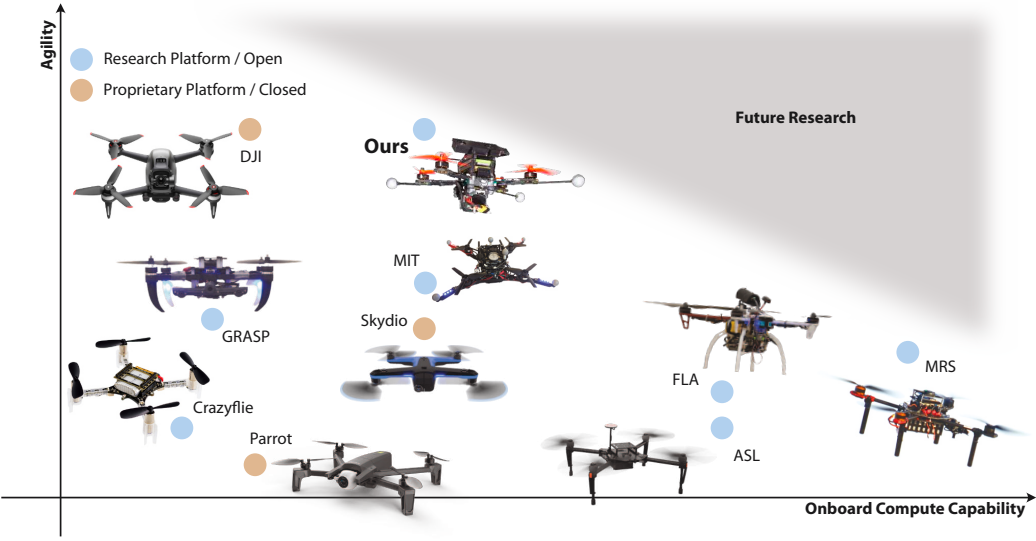
**Table K.1** – Comparison of quadrotor platforms used in research and industry. The platforms are compared based on their openness to the community, support of simulation and onboard computation, used low-level controller, CPU power (reported according to publicly available benchmarks <https://www.cpubenchmark.net> and corresponding to the speed of solving a set of benchmark algorithms that represent a generic program), and the availability of onboard general-purpose GPU. The agility of the platforms is expressed in the terms of thrust-to-weight ratio; however, we also report the maximal velocity as an agility indicator due to limited information about the commercial platforms.

view (FPV) racing community. The design objectives resulted in creating a lightweight 750 g platform with maximal speed of 131 km h<sup>-1</sup>. This high-performance drone hardware is combined with a powerful onboard computer that features an integrated GPU, enabling complex neural network architectures to run at high frequency while concurrently running optimization-based control algorithms at low latency onboard the drone. The most important features of the Agilicious framework are summarized and compared with relevant research and industrial platforms in Table K.1. A qualitative comparison of mutually contradicting onboard computational power and agility is presented in Figure K.2.

In co-design with the hardware, we complete the drone design with a modular and versatile software stack, called Agilicious. It provides a software architecture that allows to easily transfer algorithms from prototyping in simulation to real-world deployment in instrumented experiment setups, and even pure onboard-sensing applications in unknown and unstructured environments. This modularity is key for fast development and high-quality research, since it allows to quickly substitute existing components with new prototypes and enables all software components to be used in standalone testing applications, experiments, benchmarks, or even completely different applications.

The hardware and software presented in this work have been developed, tested, and refined throughout a series of research publications [79, 170, 172, 209, 328, 91, 254, 322]. All these publications share the ambition to push autonomous drones to their physical limits. The experiments, performed in a diverse set of environments demonstrate the versatility of Agilicious by deploying different estimation, control, and planning strategies on a single physical platform. The flexibility to easily combine and replace both hard- and software components in the flight stack while operating on a standardized platform facilitates testing new algorithms and accelerates research on fast autonomous flight.

# Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight



**Figure K.2** – A qualitative comparison of different available consumer and research platforms with respect to available onboard compute capability and agility. The open-source frameworks FLA[233], ASL[297], and MRS[14] have relatively large weight and low agility. The DJI[69], Skydio[317], and Parrot[300] are closed-source commercial products that are not intended for research purposes. The Crazyflie[109] does not allow for sufficient onboard compute or sensing, while the MIT[13] and GRASP[203] platforms are not available open-source. Finally, our proposed Agilicious framework provides agile flight performance, onboard GPU-accelerated compute capabilities, as well as open-source and open-hardware availability.

## K.2 Results

The capabilities of the proposed Agilicious platform are demonstrated in a set of experiments that require, due to their individual challenges, broad flexibility of the hardware and software stack. To this end, we showcase the performance of our platform in three different scenarios. First, we focus on experiments conducted in one of the world’s largest tracking arenas, where we show that our platform can be pushed to its physical limits and is able to outperform even professional human pilots. Next, we present the possibility of using Agilicious in a hardware-in-the-loop simulation, enabling the potential to test algorithms in an unlimited number of photorealistic virtual test environments while flying in the real world. Finally, we demonstrate our system’s capabilities to fly autonomously in the wild, using only onboard sensing and computation. For details regarding the hardware and software design of Agilicious, we refer the reader to Section K.4.

### K.2.1 Agile Flight in a Tracking Arena

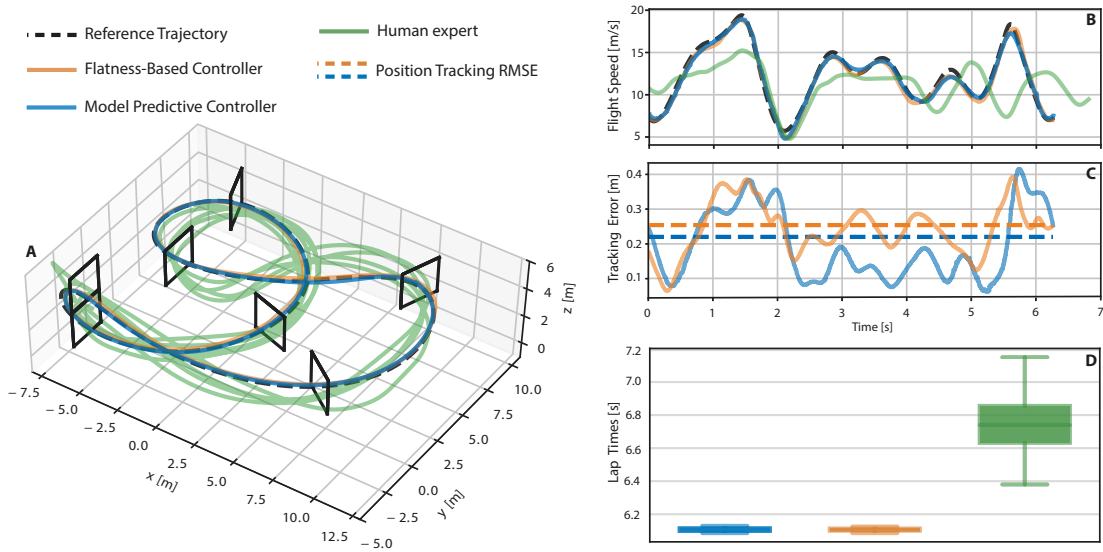
In this section, we demonstrate the versatility of Agilicious and the performance of our platform by tracking an aggressive time-optimal trajectory in a drone racing challenge. Additionally, to benchmark our planning and control algorithms, we compete against a world-class drone racing pilot FPV pilot [91]. As illustrated in Fig. , our drone racing track K.3, consists of seven gates that need to be traversed in a pre-defined order as fast as possible.

Flying through gates at high speed requires precise state estimates, which is still an open challenge using vision-based state estimators [63]. For this reason, we conduct these experiments in our tracking arena, equipped with 36 VICON cameras that provide precise pose measurements at 400 Hz. The large flight volume of  $30 \times 30 \times 8$  m ( $7,200$  m<sup>3</sup>) makes it possible to safely test many trajectories and use-cases. By virtue of the modularity offered by Agilicious, we can bypass the state estimation problem and independently focus on the planning and control tasks to extract the maximum actuation power of our platform.

To effectively complete the drone racing track, we first pre-compute the desired time-optimal trajectory using a state-of-the-art planner proposed by [91]. Then we use various non-linear flight control approaches to accurately track the trajectory (see further for more details).

Given the position of the gates, we generate the time-optimal trajectory offline by solving a non-linear optimization problem that automatically allocates the time at which the drone passes through each gate. Since this algorithm leverages the full-non-linear quadrotor dynamic model and the actuation constraints, the optimized trajectory can fully exploit the capability of our drone platform. However, accurately tracking such aggressive trajectories poses considerable challenges with respect to the controller design, which usually requires several iterations of algorithm development and considerable tuning effort.

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight



**Figure K.3** – Autonomous time-optimal flight for drone racing and comparison against a world-class drone racing pilot. A comparison between two different control approaches and the human pilot is displayed. A: a 3D view of the placement of the gates and the flown trajectories. B: velocity tracking of the two controllers, together with the velocity of the human. Notice how our platform can achieve speeds of almost  $20 \text{ m s}^{-1}$  (i.e.,  $70 \text{ km h}^{-1}$ ). C: tracking error and the RMSE of the two controlled approaches. D: distribution of lap times of the controlled platform in comparison with that of the professional human pilot.

The proposed Agilicious flight stack allows us to easily design, test, and deploy different control methods. First, we implement the control algorithm directly within the flight stack, specifically using the "Controller" box, as displayed in Fig. K.1. Then we test and tune the controller in the high-fidelity simulation environment that is built inside Agilicious. A big benefit of using our Agilicious flight stack is that after testing and tuning in the simulator, the algorithm is ready to be directly deployed and fine-tuned in the real platform, and no source code changes are necessary to transfer from the simulation stage to the real-world deployment.

Agilicious' modular structure allows for a number of different cascaded control architectures to be deployed. Among these possible combinations, in this paper we showcase two different control approaches as the outer loop: a differential-flatness-based non-linear controller [336], and a non-linear model predictive controller [346]. Both methods are designed considering the full non-linear model of the platform, the true actuator limits, and the aerodynamic drag forces. For the inner loop, in order to be able to track our platform's fast attitude dynamics, we combine the previous controllers with an incremental non-linear dynamic inversion (INDI) method [318]. This combination results in significant improvements in the robustness against model mismatch and external disturbances [329]. The INDI inner-loop converts the collective thrust and angular acceleration commands from the aforementioned controllers to rotor speed commands, which are directly commanded to the motors.

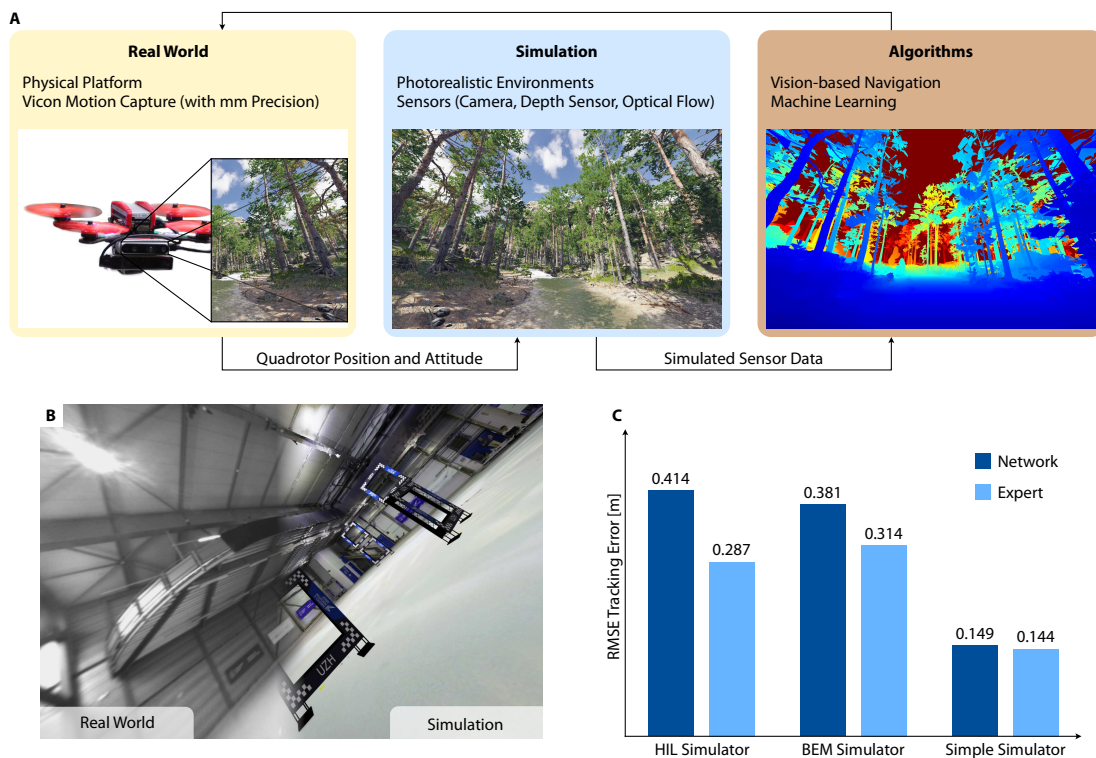
Figure K.3 shows the real-world comparison of our platform performance (using two



different controllers) against a world-class drone racing pilot. In terms of lap times (shown in Figure K.3D), our platform outperforms the human pilot and shows significantly more consistent performance. Both controllers implemented in our flight stack show good tracking performance, with less than 0.4 m tracking error while flying at almost  $20 \text{ m s}^{-1}$  (i.e.,  $70 \text{ km h}^{-1}$ ) (see Figure K.3B-C). Note that, in this example, the human pilot is not supposed to track any predefined reference trajectory.

All things considered, the Agilicious flight stack’s modularity and versatility allow us to develop a wide range of control algorithms and significantly reduce the number of iterations from writing a new algorithm to deploying it in real-world flight. These two intrinsic characteristics of our flight stack have proven to be of extreme importance when pushing the frontiers of the state-of-the-art in agile drone flight.

### K.2.2 Hardware in the Loop Simulation



**Figure K.4** – A: The hardware-in-the-loop simulation of Agilicious consists of a real quadrotor flying in a motion capture system and photorealistic simulation of complex 3D environments. Multiple sensors can then be simulated while virtually flying in various simulated environments. Such hardware-in-the-loop simulation offers a modular framework for prototyping robust vision-based algorithms safely, efficiently, and inexpensively. B-C: Hardware-in-the-loop is used to evaluate the performance of a neural policy trained on the task of drone racing and tested in different simulators available in Agilicious. The network predicts control commands (body rates and collective thrust) from simulated inertial and visual information. The network is trained by imitating a model-predictive-control expert tracking a time-optimal trajectory.

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

---

Developing vision-based perception and navigation algorithms for agile flight is slow, expensive, and unsafe. This is due, on one side, to the large amount of data required to train and test perception algorithms for operation in diverse real-world settings and, on the other side, to the high speeds of the drone, which, sooner or later, inevitably leads to crashes. This motivates the Agilicious framework to support hardware-in-the-loop simulation, which consists of flying a physical quadrotor in a motion-capture system while observing virtual photorealistic environments, as previously shown in [117].

The key advantage of hardware-in-the-loop simulation over classical quadrotor simulation [102]—where both quadrotor physics and sensors are simulated—is the usage of real-world dynamics and proprioceptive sensors (e.g., inertial measurement unit) of the physical platform instead of an idealized dynamic model based on estimated physical properties of the quadrotor. Combining a physical platform during real-world flight with photorealistic, simulated onboard camera images allows for fast, cheap, and safe development and testing of robust vision-based algorithms while minimizing the risk of crashing the quadrotor.

The simulation of complex 3D environments and realistic exteroceptive sensors is achieved using our high-fidelity quadrotor simulator [320] that is developed with Unity [153]. The simulator can offer a rich and configurable sensor suite, including RGB cameras, depth sensors, optical flow, and image segmentation. In addition, it is possible to simulate different sensor noise, e.g., motion blur or lens distortions, and diverse environmental factors, e.g., wind and rain. Thanks to the rich amount of assets on the Unity Asset Store [348], the implementation of different testing environments is fast and cheap. For example, the user can switch from a city to a forest environment in under one minute.

The Agilicious hardware-in-the-loop simulation uses the position and attitude of the quadrotor measured by a VICON motion capture system to render sensor data in the frame of simulated sensors attached to the quadrotor. The simulation of synthetic environments and the rendering of images are performed by a powerful workstation that features a high-performance GPU. Such offboard rendered sensory data can then be sent to the quadrotor in real-time and used for testing vision algorithms onboard. Alternatively, when prototyping algorithms that require expensive computations, the simulated data can be processed directly offboard, and low bandwidth results such as control commands are then sent to the quadrotor. Hence, Agilicious allows for rapid development of vision-based perception, estimation, planning, and control algorithms, such as visual(-inertial) odometry, SLAM, image segmentation, and end-to-end control via neural network policies.

We demonstrate the advantages of the hardware-in-the-loop simulation by using it for testing an end-to-end neural network policy on the task of vision-based drone racing. Following the approach in [208], we train the network by imitating a model predictive controller tracking a time-optimal trajectory [91]. Similarly to [172], the network directly predicts control commands, i.e. body rates and collective thrust, from visual and inertial measurements. The network is trained from experience collected in the simple simulator and evaluated in high-fidelity simulators including hardware-in-the-loop. We use the tracking error with respect to the time-optimal reference trajectory as a performance

metric.

From the results presented in Figure K.4C, two main conclusions can be drawn. First, when flying in a realistic BEM simulator or in hardware-in-the-loop, the performance of both the expert and the network significantly drops (up to 65%). This is justified by the fact that the simple simulator, used by both the expert and the network, does not account for the majority of aerodynamic effects acting on the platform, which are particularly relevant at high speeds. Second, we observe that the BEM simulator of Agilicious can closely predict the behavior of the closed-loop system in the physical world. Note that the hardware-in-the-loop simulation played a fundamental role in drawing such conclusions, since it enables to only vary the physical model between the experiments, keeping the sensor observations unchanged. Additionally, hardware-in-the-loop allows us to test vision-in-the-loop algorithms safely regardless of large tracking errors, which could otherwise lead to crashing into physical gates.

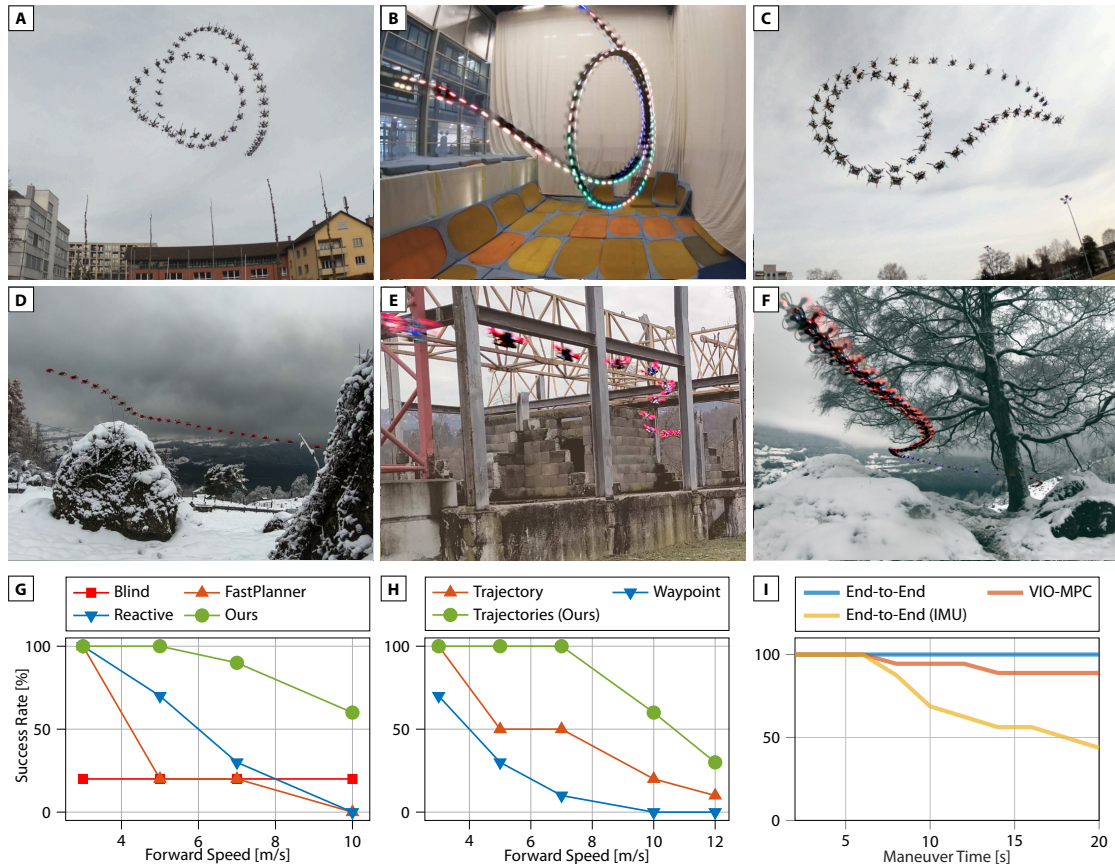
Overall, the integration of our agile quadrotor platform and high-fidelity visual simulation provides an efficient framework for the rapid development of vision-based navigation systems in complex and unstructured environments. Moreover, the framework opens up opportunities for transferring end-to-end neural network policies to the real world by combining the real aerodynamics and system delays with almost unlimited synthetic images.

### K.2.3 Vision-based Agile Flight with Onboard Sensing and Computation

When a quadrotor can only rely on onboard vision and computation, perception needs to be effective, low-latency, and robust to disturbances. Violating this requirement may lead to crashes. Therefore, agile flight with only onboard resources pushes the boundaries of existing navigation systems and offers a very interesting venue for research. Demonstrations of our recent works on agile flight with onboard vision and computation [172, 209], enabled by the Agilicious flight stack, are shown in Figure K.5A-F. Key highlights are acrobatic flight with accelerations up to 3g and obstacle avoidance in both structured and unstructured environments with speeds up to  $10 \text{ m s}^{-1}$ . During these maneuvers, the actuators, sensors, and all physical components of the platform are tested to their limits. Latency has to be low and robustness to perception disturbances must be high.

Vision systems either exhibit reduced accuracy or completely fail at high speeds due to perception disturbances such as motion blur, large pixel displacements, and quick illumination changes [41]. To overcome these challenges, vision-based navigation systems generally build upon two different paradigms. The first uses the traditional perception-planning-and-control pipeline, generally represented by standalone blocks which are executed sequentially and designed independently [257, 89, 83, 373, 83, 203]. Works in the second category substitute either parts or the complete perception-planning-and-control pipeline with learning-based methods [367, 368, 18, 136, 172, 371, 290, 298, 206, 106]. We deploy the Agilicious flight stack to quantitatively compare in simulation traditional and learning-based methods using the aforementioned navigation tasks.

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight



**Figure K.5** – The Agilicious platform is deployed in a diverse set of environments while only relying on onboard sensing and computation. A-C: The quadrotor performs a set of acrobatic maneuvers using a learned control policy. D-F: By leveraging zero-shot sim2real transfer, the quadrotor platform performs agile navigation through cluttered environments. G-I: The flexibility of Agilicious enables rapid development and benchmarking of model-based and learning-based methods. To demonstrate this flexibility, we evaluate state-of-the-art approaches in the task of navigation in a previously unknown forest environment (G, H), and acrobatic flight (I) by consecutively repeating the barrel-roll maneuver (A) multiple times. Courtesy of Kaufmann et al. [172] and Loquercio et al. [209].

We first consider the problem of high-speed obstacle avoidance in a previously unknown forest environment. As a baseline, we select two state-of-the-art approaches operating with the traditional perception-planning-control pipeline: FastPlanner [373], which builds a map of the environment and uses it with an A\* planner, and Reactive [89], that instantaneously selects the best trajectory from a set of pre-defined motion primitives. We compare these methods to a learning-based approach that selects a collision-free trajectory from depth and inertial measurements (Ours) [209]. For all approaches, we use the Agilicious flight stack to track the predicted trajectory. The results of this experiment, presented in Figure K.5G, show an interesting pattern: at low speeds ( $3 \text{ m s}^{-1}$ ) all methods perform similarly. However, as the speed increases, the traditional methods' performances quickly drop, and already at  $5 \text{ m s}^{-1}$  they are not able to complete all runs without crashing. In contrast, the learning-based method can reliably fly at high speeds through all environments, achieving a success rate of 60% at  $10 \text{ m s}^{-1}$ . The drop in performance of the traditional methods can be justified by their sensitivity to perception disturbances and limited expressive power. By contrast, our data-driven approach enables leveraging regularities in the data to predict collision-free trajectories from noisy data, which makes it both more expressive and more robust to sensor noise than the baselines [209].

We further use the possibility to seamlessly switch between different input modalities of Agilicious (Figure K.1) for ablating the action representation of the learning-based policy. From depth images and inertial measurements, the base policy predicts a set of three trajectories to account for the multi-modality of the avoidance task [209]. We compare this output representation to a single trajectory or a waypoint while keeping the rest of the approach and underlying stack unchanged. The results in Figure K.5H indicate that without accounting for the multi-modality of the task the performance is degraded. This is because the  $l_2$ -loss pushes predicted trajectories toward the average of the expert trajectories, which is often in collision with obstacles. In addition, substituting a trajectory to the less complex action representation of a single waypoint leads to similar drops in performance.

Finally, we perform a comparison between the traditional approach of state-estimation and control (VIO-MPC) [79] and a learning-based method (End-to-End) [172] in the task of acrobatic flight. We further compare to a blind policy with no access to visual information (End-to-End (IMU)). Figure K.5H analyzes the evolution of success rates of different methods over time while flying a barrel roll maneuver [172]. A neural policy that only relies on inertial data for sensing can safely fly, but only for short periods (approximately 6 s). Similarly, the traditional estimation and control pipeline has no failures for shorter maneuvers, but its performance gracefully degrades when flight time increases. Conversely, the controller that has access to both visual and inertial data (*End-to-End*) is able to perform barrel rolls for 20 seconds without a single failure. These results indicate that, despite the challenges of reliable perception at high speeds, visual input is fundamental to reduce drift and increase robustness for long maneuvers.

Overall, the Agilicious hardware is a perfect fit for vision-based agile flight. Indeed, it provides reliable and low-latency interfacing to a wide range of sensors, from frame-based to event-based cameras [104], and high-performance compute hardware (c.f. Section K.4.1). In addition, our software framework provides the flexibility to quickly develop and

## **Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight**

---

benchmark different vision-based navigation paradigms, including model-based, learning-based, or hybrid approaches.

## K.3 Discussion

The presented Agilicious framework substantially advances the published state of the art in autonomous quadrotor platform research. It offers advanced computing capabilities combined with the most powerful open-source and open-hardware quadrotor platform created to date, opening the door for research on the next generation of autonomous robots. We see three main axes for future research based on our work.

First, we hypothesize that future flying robots will be smaller, lighter, cheaper, and consuming less power than what is possible today, increasing battery life, crash-resilience, as well as thrust-to-weight ratio and torque-to-inertia ratio [88]. This miniaturization is evident in state-of-the-art research towards direct hardware implementations of modern algorithms in the form of application-specific integrated circuit (ASIC)s, such as the Navion [327] or the PULP processor [262]. These highly specialized in-silicon implementations are typically magnitudes smaller and more efficient than general compute units. Their success is rooted in the specific structure many algorithmic problems exhibit, such as the parallel nature of image data or the factor-graph representations used in estimation, planning, and control algorithms, like SLAM, model-predictive control, and neural network inference. This raises the question of whether these problems could be solved using ASICs and whether the overall system benefits from miniaturization. Navion [327] and modern neural-network accelerators, such as the Movidius [146] or the PULP architecture [262, 263] already hint at the potential gains such in-silicon implementations could bring to drones, and Agilicious provides the modular and flexible environment necessary to prototype and test such solutions.

Second, the presented framework was mainly demonstrated with fixed-shape quadrotors. This is an advantage as the platform is easier to model and control, and less susceptible to hardware failures. Nevertheless, platforms with a dynamic morphology are by design more adaptable to the environment and power efficient [4, 45, 81, 21]. For example, to increase flight time, a quadrotor might transform to a fixed-wing aircraft [58]. Due to its flexibility, Agilicious is the ideal tool for the future development of morphable and soft aerial systems that, like many birds, change their shape to efficiently move and interact with the environment.

Finally vision-based agile flight is still in the early stages and has not yet reached the performance of professional human pilots. The main challenges lie in handling complex aerodynamics, e.g. transient torques or rotor inflow, low-latency perception and state estimation, and recovery from failures at high speeds. In the last few years, considerable progress has been made by leveraging data-driven algorithms [23, 172, 209] and novel sensors as event-based cameras [82, 328], that provide a high dynamic range, low latency, and low battery consumption [104]. A major opportunity for future work is to complement the existing capabilities of Agilicious with novel compute devices such as the Intel Loihi [59, 73, 354] or SynSense Dynap [238] neuromorphic processing architecture, which are specifically designed to operate in an event-driven compute scheme. Due to the modular nature of Agilicious, individual software components can be replaced by these novel computing architectures, supporting rapid iteration and testing.

## **Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight**

---

In summary, Agilicious offers a unique quadrotor testbed to accelerate current and future research on fast autonomous flight. Its versatility in both hardware and software allows deployment in a wide variety of tasks, ranging from exploration or search and rescue missions to acrobatic flight. Furthermore, the modularity of the hardware setup allows integrating novel sensors or even novel compute hardware, enabling to test such hardware on an autonomous agile vehicle. By open-sourcing Agilicious, we provide the research and industrial community access to a highly agile, versatile, and extendable quadrotor platform.



## K.4 Materials and Methods

Designing a versatile and agile quadrotor research platform requires to co-design its hardware and software, while carefully trading off competing design objectives such as onboard computing capacity and platform agility. In the following, the design choices that resulted in the flight hardware, compute hardware, and software design of Agilicious (see Fig. K.1) are explained in detail.








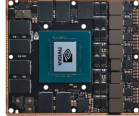
### K.4.1 Compute Hardware

To exploit the full potential of highly unstable quadrotor dynamics, a high-frequency low-latency controller is needed. Both of these requirements are difficult to meet with general-purpose operating systems, which typically come without any real-time execution guarantees. Therefore, we deploy a low-level controller with limited compute capabilities but reliable real-time performance, which stabilizes high-bandwidth dynamics, such as the motor speeds or the vehicle’s bodyrate. This allows complementing the system with a general-purpose high-level compute unit that can run Linux for versatile software deployment, with significantly relaxed real-time requirements.

**High-Level Compute Board** The high-level of the system architecture provides all the necessary compute performance to run the full flight stack, including estimation, planning, optimization-based control, neural network inference, or other demanding experimental applications. Therefore, the main goal is to provide general-purpose computing power, while complying with the strict size and weight limits. We evaluate a multitude of different compute modules made from system-on-a-chip (SoC) solutions since they allow inherently small footprints. We exclude the evaluation of two popular contenders: (a) the Intel NUC platform, since it neither provides any size and weight advantage over the Jetson Xavier AGX nor provides a general-purpose GPU; and (b) the Raspberry Pi compute modules since they do not offer any compute advantages over the Odroid and UpBoard, and no size and weight advantage over the NanoPi product family.

As we target general flight applications, fast prototyping, and experimentation, it is important to support a wide variety of software, which is why we chose a Linux-based system. An overview of existing compute modules is shown in Tab. K.2. The required module should provide enough computational power to run state-of-the-art estimation and control algorithms, support hardware-accelerated neural network inference for real-time deployment and other heterogeneous compute applications, and provide simple interfacing with other hardware. TensorFlow [221] and PyTorch [266] are some of the most prominent frameworks with hardware-accelerated neural network inference. Both of them support accelerated inference on the Nvidia CUDA general-purpose GPU architecture, which renders nVidia products favorable, as other products have no or poorly-supported accelerators. Therefore, four valid options remain, listed in the second row of Tab. K.2. While the Jetson Xavier AGX is beyond our size and weight goals, the Jetson Nano provides no advantage over the Xavier NX, rendering both the Jetson TX2 and Xavier NX viable solutions. Since these two CUDA-enabled compute modules

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

without General-Purpose GPU				
				
Product	Odroid XU4	Intel UpBoard	NanoPi Neo 3	NanoPi Neo air
CPU	8 × 32-bit ARM 2.1 GHz	4 × 64-bit Atom 1.92 GHz	4 × 64-bit ARM 1.5 GHz	4 × 32-bit ARM 1.2 GHz
RAM	2 GB LPDDR3	4 GB LPDDR3	2 GB LPDDR4	512 MB LPDDR3
GPU	Mali-T628	Intel HD400	Mali-450 MP4	Mali-400 MP2
FLOPS	~120 GFLOPS	~ 115 GFLOPS	~40 GFLOPS	~10 GFLOPS
Storage	up to 128 GB EMMC	up to 64 GB EMMC	only SD card	8 GB EMMC
Interfaces	USB, Ethernet, Serial, I2C, SPI, GPIO	USB, Ethernet, Serial, I2C, SPI, GPIO, 1 camera	USB, Ethernet, Serial, I2C, SPI, GPIO	USB, Ethernet, WIFI, Serial, I2C, SPI, GPIO, 1 camera
Size	83 × 58 × 19 mm	85 × 57 × 20 mm	40 × 40 × 23 mm	40 × 40 × 10 mm
Weight	59 g	79 g	36 g	24 g
with General-Purpose GPU				
				
Product	nVidia Jetson Nano	nVidia Jetson TX2	nVidia Jetson Xavier NX	nVidia Jetson AGX Xavier
CPU	4 × 64-bit ARM 1.43 GHz	6 × 64-bit ARM 2.0 GHz	6 × 64-bit ARM 1.9 GHz	8 × 64-bit ARM 2.26 GHz
RAM	4 GB LPDDR4	8 GB LPDDR4	8 GB LPDDR4	32 GB LPDDR4
GPU	128 × Maxwell CUDA	256 × Pascal CUDA	384 × Volta CUDA	512 × Volta CUDA
FLOPS	472 GFLOPS	1.33 TFLOPS	2.12 TFLOPS	11 TFLOPS
Storage	16 GB EMMC	32 GB EMMC	16 GB EMMC	32 GB EMMC
Interfaces	USB, Ethernet, Serial, I2C, SPI, GPIO, 4 cameras	USB, Ethernet, WIFI, Serial, I2C, SPI, GPIO, 6 cameras	USB, Ethernet, Serial, I2C, SPI, GPIO, 6 cameras	USB, Ethernet, Serial, I2C, SPI, GPIO, 6 cameras
Size	69.9 × 45 × 22 mm	87 × 50 × 34 mm	69.9 × 45 × 22 mm	100 × 87 × 58 mm
Weight	63 g	154 g	79 g	650 g

**Table K.2** – Overview of compute hardware commonly used on autonomous flying vehicles. Due to the emerging trend of deploying learning-based methods onboard, hardware solutions are grouped according to the presence of a general-purpose GPU, enabling real-time inference.

require a breakout board to connect to peripherals, our first choice is the TX2 due to the better availability and diversity of such adapter boards, smaller footprint, and support of onboard WiFi networking. The opted Jetson TX2 is used together with the breakout board ConnectTech Quasar[55], providing multiple USB ports, Ethernet, serial ports, and other interfaces for sensors and cameras.

**Low-Level Flight Controller** The Low-Level Flight Controller must provide real-time low-latency interfacing and control of the vehicle’s and actuator’s high-frequency dynamics. A simple and widespread option is the open-source BetaFlight [340] software which runs on many commercially available flight controllers, such as the Radix[211]. However, BetaFlight is made for human-piloted drones with a control pipeline optimized for a good human flight feeling, but not for autonomous operation. Furthermore, even though it uses high-speed IMU readings for the control loop, it only provides very limited sensor readings at only 10 Hz. Therefore, Agilicious provides its own low-level flight controller implementation called "agiNuttX", reusing the same hardware as the BetaFlight controllers. This means that the wide variety of commercially available products can be bought and reflashed with agiNuttX to provide a low-level controller suited for autonomous agile flight.

In particular, we recommend using the BrainFPV Radix [211] controller, to deploy our agiNuttX software. The agiNuttX is based on the open-source NuttX [339] real-time operating system, optimized to run on embedded microcontrollers such as the STM32F4 used in many BetaFlight products. Our agiNuttX implementation interfaces with the motor’s ESC over the digital bi-directional Dshot protocol, allowing not only to command the motors, but also readout their rotational speed in real-time. This feedback is provided to the high-level controller together with IMU and battery voltage measurements, and quadrotor status information over a 1Mbaud serial bus at 500 Hz. The agiNuttX also provides closed-loop motor speed control, bodyrate control, and time synchronization for precise time-stamped measurements, allowing estimation and control algorithms to take full advantage of the available hardware.

### K.4.2 Flight Hardware

To maximize the agility of the drone, it needs to be designed as lightweight and small as possible [187] while still being able to accommodate the Jetson TX2 compute unit. This section gives an overview of the components used (for details, see Tab. K.2) and design choices made. The Armattan Chameleon 6 inch frame is used as a base because its freestyle FPV design provides more space for the compute hardware than a conventional FPV racing frame. Being made out of carbon fiber, it is very durable while weighing only 86 g. The other structural parts of the quadrotor are custom-designed plastic parts (PLA, TPU) and produced using a 3D printer. Most components are made out of PLA which is stiffer and only parts that act as impact protectors or as predetermined breaking points are made out of TPU. For propulsion, a 5.1 inch three-bladed propeller is used since a larger propeller would not leave enough space to fit the compute and sensing components. Three-bladed propellers are favored in drone racing as they provide more lift per area and have lower rotational inertia compared to a two-bladed propeller with a similar lift curve. To achieve a high thrust output, fast-spinning brushless DC motors with a high maximum power rating of 758 W and a high KV-number (2400KV) are selected. The chosen motor-propeller combination achieves a continuous static thrust of  $4 \times 9.5 \text{ N}$  on the quadrotor and consumes about 400 W of power per motor. To match the high power demand of the motors, a lithium-polymer (LiPo) battery with 1,800 mA h and a rating of 120C is used. Therefore, the total peak current of 110 A is well within the 216 A limit of the battery.

For agile flight, the low-level flight controller (FC) and electronic speed controller (ESC) are integral components of the overall design. To have access to high-frequency motor speed measurements, an ESC supporting the Dshot protocol is required. The Hobbywing XRotor ESC is selected due to its compact form factor, its high current rating (60A per motor), and support of the very fast Dshot 1200 protocol. The Radix from Brainfpv is used as a flight controller because it comes with an ARM Cortex-M4 based, a widely used STM32F4 MCU for which the custom firmware was developed. The FC can be interfaced with the high-level controller and other sensors (e.g. GPS) through 6 serial connectors.

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

Component	Product	Specification
Frame	Armattan Chameleon 6 inch	4 mm carbon fiber, 86 g
Motor	Xrotor 2306	23×6 mm stator, 2,400 kV, 758 W, 4× 27.5 g
Propeller	Azure Power SFP 5148	5.1 inch length and 4.8 inch pitch, 4× 5 g
Battery	Tattoo R-Line 1800	4× 3.7 V, 1,800 mA h, 199 g
Flight Controller	BrainFPV radix	BetaFlight or custom firmware, 6 g
Motor Controller	HobbyWing XRotor	DSHOT protocol, 4× 60 A, 15 g
Compute Unit	nVidia Jetson TX2	6× ARM 2.0 GHz, 256× CUDA cores, 8 GB, 154 g

Table K.3 – Overview of the components of the flight hardware design.

### K.4.3 The Agilicious Flight Stack Software

To exploit the full potential of our platform and enable fast prototyping, we provide the Agilicious flight stack as an open-source software package. The main development goals for Agilicious are aligned with our overall design goals: high versatility and modularity, low latency for agile flight, and transferability between simulation and multiple real-world platforms. These goals are met by splitting the software stack into two parts.

The core library, called "agilib", is built with minimal dependencies but provides all functionality needed for agile flight, implemented as individual modules (illustrated in Fig. K.1). It can be deployed on a large range of computing platforms, from lightweight low-power devices to parallel neural network training farms built on heterogeneous server architectures. This is enabled by avoiding dependencies on other software components that could introduce compatibility issues and rely only on the core C++-17 standard and the Eigen library for linear algebra. Additionally, agilib includes a standalone set of unit tests and benchmarks that can be run independently, with minimal dependencies, and in a self-contained manner.

To provide compatibility to existing systems and software, the second component is a ROS-wrapper, called "agiros", which enables networked communication, data logging, provides a simple GUI interface to command the vehicle and allows for integration with other software components. This abstraction between "agiros" and the core library "agilib" allows a more flexible deployment on systems or in environments where ROS is not available, not needed, or communication overhead must be avoided. On the other hand, the ROS-enabled Agilicious provides versatility and modularity due to a vast number of open-source ROS packages provided by the research community.

For flexible and fast development, "agilib" uses modular software components unified in an umbrella structure called "pipeline" and orchestrated by a control logic, called "Pilot". The modules consist of an "estimator", "sampler", "controllers", and a "bridge", all working together to track a so-called "reference". These modules are executed in sequential order within a forward pass of the pipeline, corresponding to one control cycle. However, each module can spawn its individual threads to perform parallel tasks, e.g. asynchronous sensor data processing. Agilicious provides a collection of state-of-the-art implementations for each module, inherited from base classes of the modules. This allows to not only create new modules within Agilicious, but also outside of the core library.

These external modules can be registered and linked into the pilot at runtime without changing the core library, allowing for rapid prototyping and independent development. This way, Agilicious is not only capable to control a drone when running onboard the vehicle, but can also run offboard on computationally more capable hardware and send commands to the drone over low-latency wireless serial interfaces.

Finally, the core library is completed by a physics simulator. While this might seem redundant due to the vast variety of simulation pipelines available [102, 309, 320], it allows to use high-fidelity models (e.g. for aerodynamics), evaluates software prototypes without having to interface with other frameworks, avoids dependencies, and enables even simulation-based continuous integration testing that can run on literally any platform. Moreover, it also allows us to experiment with our own built-in simulators, and extend them with state-of-the-art functionalities. The pilot, software modules, and simulator are all described in the following sections.

### Pilot

The pilot contains the main logic needed for flight operation, handling of the individual modules, and interfaces to manage references and task commands. In its core, it loads and configures the software modules according to YAML[25] parameter files, runs the control loop, and provides simplified user interfaces and ROS [279] bindings to manage flight tasks, such as position and velocity control or the generation and execution of trajectories. For all state descriptions, we use a right-handed coordinate system located in the center of gravity, with the  ${}_B\mathbf{e}_z$  pointing in body-relative upward thrust direction, and  ${}_B\mathbf{e}_x$  pointing along with the drone's forward direction. Motion is represented with respect to an inertial world frame with  ${}_I\mathbf{e}_z$  pointing against the gravity direction, where translational derivatives (e.g. velocity) are expressed in the world frame and rotational derivatives (e.g. bodyrate) are expressed in the body frame.

**Estimator** The first module in the pipeline is the estimator, which provides a time-stamped state estimate for the subsequent software modules in the control cycle. A state estimate  $\mathbf{x} = [\mathbf{p}, \mathbf{q}, \mathbf{v}, \boldsymbol{\omega}, \mathbf{a}, \boldsymbol{\tau}, \mathbf{j}, \mathbf{s}, \mathbf{b}_\omega, \mathbf{b}_a, \mathbf{f}_d, \mathbf{f}]$ , representing position  $\mathbf{p}$ , orientation unit quaternion  $\mathbf{q}$ , velocity  $\mathbf{v}$ , bodyrate  $\boldsymbol{\omega}$ , linear  $\mathbf{a}$  and angular  $\boldsymbol{\tau}$  accelerations, jerk  $\mathbf{j}$ , snap  $\mathbf{s}$ , gyroscope and accelerometer bias  $\mathbf{b}_\omega$  and  $\mathbf{b}_a$ , and desired and actual single rotor thrusts  $\mathbf{f}_d$  and  $\mathbf{f}$ . Most of the state components are optional, with the minimal required set being position  $\mathbf{p}$ , orientation  $\mathbf{q}$ , and velocity  $\mathbf{v}$ . Agilicious provides a feed-through estimator to include external estimates or ground-truth from a simulation, as well as two extended Kalman filters, one with IMU filtering, and one using the IMU as propagation model. These estimators can easily be replaced or extended to work with additional measurement sources, such as GPS or altimeters, other estimation systems, or even implement complex localization pipelines such as visual-inertial odometry.

**Sampler** For trajectory tracking using a state estimate from the aforementioned estimator, the controller module needs to be provided with a subset of points of the trajectory

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

---

that encode the desired progress along it. A sampler performs this subsampling of a reference trajectory. Agilicious implements two types of samplers: a time-based sampling scheme that computes progress along the trajectory based on the time since trajectory start, and a position-based sampling scheme that selects trajectory progress based on the current position of the platform, trading off temporally accurate tracking for higher robustness and lower positional tracking error.

**Controller** To control the vehicle along the sampled reference setpoints, a multitude of controllers are available, which provide the closed-loop commands for the low-level controller. We provide a state-of-the-art model-predictive control (MPC) that uses the full non-linear model of the platform and which allows to track highly agile references using single-rotor thrust commands or bodyrate control, compensates aerodynamic effects. Additionally, we include a cascaded geometric controller based on the quadrotor’s differential flatness[227]. The pipeline can cascade two controllers, which even allows combining the aforementioned MPC or geometric approaches with an intermediate controller for which we provide an L1 adaptive controller[123] and an incremental non-linear dynamic inversion controller[329].

**Bridge** A bridge serves as an interface to hardware or software by sending control commands to a low-level controller or other means of communication sinks. Low-level commands can either be single rotor thrusts or bodyrates in combination with a collective thrust. Agilicious provides a set of bridges to communicate via commonly used protocols such as ROS, SBUS, and serial. While the ROS-bridge can be used to easily integrate Agilicious in an existing software stack that relies on ROS, the SBUS protocol is a widely used standard in the FPV community and therefore allows to interface Agilicious to off-the-shelf flight controllers such as BetaFlight[340]. For simple simulation, there is a specific bridge to interface with the popular RotorS [102] simulator, which is however less accurate than our own simulation described in Sec. K.4.3. As Agilicious is written in a general abstract way, it runs on onboard compute modules and offboard, for which case we provide a bridge to interface with the LAIRD[188] wireless serial interface. Finally, Agilicious also provides a bridge to talk to the custom low-level controller described in Sec. K.4.1. This provides the advantage of gaining access to closed-loop single rotor speed control, high-frequency IMU, rotor speed, and voltage measurements at 500 Hz, all provided to the user through the bridge.

**References** References are used in conjunction with a controller to encode the desired flight path of a quadrotor. In Agilicious, a reference is fed to the sampler, which generates a receding-horizon vector of setpoints that are then passed to the controller. The software stack implements a set of reference types, consisting of *Hover*, *Velocity*, *Polynomial*, and *Sampled*. While *Hover* references are uniquely defined by a reference position and a yaw angle, a *Velocity* reference specifies a desired linear velocity with a yaw rate. By exploiting the differential flatness of the quadrotor platform, *Polynomial* references describe the position and yaw of the quadrotor as polynomial functions of time. Sampled references provide the most general reference representations. Agilicious provides interfaces to

generate, and receive such sampled references and also defines a message and file format to store references to a file. By defining such formats, a wide variety of trajectories can be generated, communicated, saved, and executed using Python or other languages. Finally, to simplify the integration and deployment of other control approaches, Agilicious also exposes a command feedthrough, that allows taking direct control over the applied low-level commands.

## Simulation

The Agilicious software stack includes a simulator that allows simulating quadrotor dynamics at various levels of fidelity to accelerate prototyping and testing. Specifically, Agilicious models motor dynamics and aerodynamics acting on the platform. To also incorporate the different, possibly off-the-shelf, low-level controllers that can be used on the quadrotors, the simulator can optionally simulate the behavior of low-level controllers. One simulator update, typically called at 1 kHz, includes a call to the simulated low-level controller, the motor model, the aerodynamics model, and the rigid body dynamics model in a sequential fashion. Each of these components is explained in the following.

**Low-Level Controller & Motor Model** Simulated low-level controllers run at simulation frequency and convert collective thrust and bodyrates commands into individual motor speed commands. The usage of a simulated low-level controller is optional if the computed control commands are already in the form of individual rotor thrusts. In this case, the thrusts are mapped to motor speed commands and then directly fed to the simulated motor model. The motors are modeled as a first-order system with a time constant which can be identified on a thrust test stand.

**Aerodynamics** The simulated aerodynamics model lift and drag produced by the rotors from the current ego-motion of the platform and the individual rotor speeds. Agilicious implements two rotor models: *Quadratic* and *BEM*. The *Quadratic* model implements a simple quadratic mapping from rotor rotational speed to produced thrust, as commonly done in quadrotor simulators [102, 309, 320]. While such a model does not account for effects imposed by the movement of a rotor through the air, it is highly efficient to compute. In contrast, the *BEM* model leverages Blade-Element-Momentum-Theory (BEM) to account for the effects of varying relative airspeed on the rotor thrust. To further increase the fidelity of the simulation, a neural network predicting the residual forces and torques (e.g. unmodeled rotor to rotor interactions and turbulence) can be integrated into the aerodynamics model. For details regarding the *BEM* model and the neural network augmentation, we refer the reader to [23].

**Rigid Body Dynamics** Provided with a model of the forces and torques acting on the platform predicted by the aerodynamics model, the system dynamics of the quadrotor are integrated using a 4th order Runge-Kutta scheme with a step size of 1 ms. Agilicious also implements different integrators such as explicit Euler or symplectic Euler.

## Appendix K. Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight

---

Apart from providing its own state-of-the-art quadrotor simulator, Agilicious can also be interfaced with external simulators. Interfaces to the widely-used RotorS quadrotor simulator [102] and Flightmare [320], including the hardware-in-the-loop (HIL) simulator, are already provided in the software stack.



# L Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

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

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

# Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

Titus Cieslewski, Elia Kaufmann, Davide Scaramuzza

**Abstract** — Exploring and mapping previously unknown environments while avoiding collisions with obstacles is a fundamental task for autonomous robots. In scenarios where this needs to be done rapidly, multi-rotors are a good choice for the task, as they can cover ground at potentially very high velocities. Flying at high velocities, however, implies the ability to rapidly plan trajectories and to react to new information quickly. In this paper, we propose an extension to classical frontier-based exploration that facilitates exploration at high speeds. The extension consists of a reactive mode in which the multi-rotor rapidly selects a goal frontier from its field of view. The goal frontier is selected in a way that minimizes the change in velocity necessary to reach it. While this approach can increase the total path length, it significantly reduces the exploration time, since the multi-rotor can fly at consistently higher speeds.

## Multimedia Material

A video attachment to this work is available at <https://youtu.be/54s6gGZLpJo>.

### L.1 Introduction

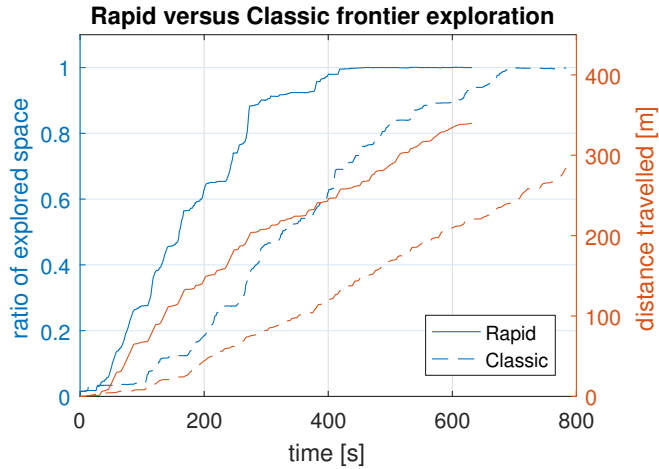
Exploring and mapping previously unknown environments is a fundamental task for autonomous robots. Given an environment with free (traversable) and occupied (untraversable) space, the task is to detect free space within a target area. This can be used to map previously unseen environments or to search for objects or people, such as in search and rescue scenarios. The task can be further specified depending on further requirements. In search and rescue, for instance, it is important to find survivors rapidly. Thus, an objective would be to cover the area as quickly as possible. A related objective would be to expend as little energy as possible, for example when mapping with an energy-constrained robot.

Another objective would be to minimize the uncertainty of the map. Many exploration algorithms assume that pose estimation and free space detection are good enough that effects of uncertainty can be ignored, or reduced to the uncertainty of depth sensors only. This assumption, however, cannot be made generally, since real systems do exhibit uncertainty. Exploration algorithms that take uncertainty into account are usually more complex and less generally applicable than algorithms that make the simplifying assumption, since they need to consider specific aspects of the underlying mapping algorithm. In our work, we make this assumption and focus on the case where exploration is performed with a single multi-rotor robot. Most related work in the same setting designs the exploration algorithm in a way that minimizes the distance traveled. At the same time, it is often assumed that the multi-rotor flies at low velocities, which relaxes design constraints when it comes to execution time and trajectory generation. However, a multi-rotor expends energy on hovering and thus its limited energy supply is best used when flying at velocities that are not near-hover.

In this paper, we propose an exploration algorithm that is designed to fly at high velocities as much as possible. To that end, we introduce a reactive mode, which, instead of planning trajectories generates instantaneous velocity commands based on currently observed frontiers. This control loop is able to run at high frequency, allowing high velocity flight and rapid incorporation of new information. Frontiers that drop out of the current field of view are added to a data structure with global frontiers and we fall back to classical frontier-based exploration as soon as no frontiers are left in the field of view. We evaluate our approach, compare it to previous approaches and show that while it can result in larger distances traveled, the exploration time is lower than with other approaches, in particular at higher velocities.

## Appendix L. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

---



**Figure L.1** – We propose an algorithm for exploration with multi-rotors at high speeds. While our approach sometimes ends up traveling a larger distance, it is able to complete the exploration task faster, as it is capable of flying at higher speeds, as can be seen in this data of a single run. Note that a lot of time is spent at almost 100% coverage, as the robot needs to travel some distance to complete the final portions of undiscovered space.

### L.2 Related work

As previously stated, exploration is the task of detecting free space within a given area. This is achieved with a robot that is capable of detecting free space, typically with a depth sensor such as an RGBD camera or a laser range finder. Since a robot normally cannot perceive the whole environment from one position, exploration reduces to repeatedly deciding where to move next at a given time when the area is already partially explored. This is essentially the same as the next-best-view (NBV) problem, which has been studied for reconstructing 3D objects with a sequence of depth scans for several decades [56, 223].

An adaptation of the NBV problem to a robotic context has been performed in [113]. It consists of adding two constraints: Firstly, a view may only be considered if a safely navigable path to it exists. Secondly, enough overlap between the current and next view must exist, such that the robot can register the two views. The proposed solution is to sample poses in the known free space, and to calculate for each sample a utility function that consists of the expected gain in terms of area that can be discovered and the cost to reach it. Then, from reachable samples that satisfy the overlap constraint, the one that maximizes the utility function is chosen. The cost is typically set to be proportional to the distance to the sample, but can also encode practical considerations such as minimizing behaviors that increase uncertainty [347]. In our approach, we assign a high cost to changes in velocity.

Frontier-based exploration is an alternative to NBV approaches that was proposed in [364]. The environment is discretized into a 2D or 3D grid, where each cell is labeled as occupied, free or unknown. Frontier cells are defined as free cells that are adjacent to unknown cells. Instead of sampling candidate views, frontier-based exploration assumes that simply navigating to a frontier will result in the exploration of new space. In [364], the robot navigates to the closest frontier found by depth-first search. Frontier-based exploration

has become a popular exploration approach that has been extended in several publications, for example to support exploration using multiple robots [365, 40]. Extensions also exist for single robots systems, but it has been repeatedly shown that classic frontier-based exploration remains competitive [135] or even outperforms [151] these extensions. [135] also shows that the frontier-based approach outperforms the NBV approach presented in [113]. As a consequence, we will use as one baseline the nearest-frontier approach as implemented in [151]. For situations where uncertainty cannot be neglected, [151] shows that algorithms designed for that scenario, such as [219] or [152] result in a lower map error.

Exploration specifically using multi-rotors has been shown in [98], [310] and [30], among others. In [98], the authors propose a frontier-based exploration strategy for quadrotors using the 3D occupancy map OctoMap [137]. As in [135], this occupancy map is continuously updated, and thus more information is captured. Furthermore, before targeting a new frontier, flood-fill is used to reject unreachable frontiers, reducing time spent in trying to reach them. [310] proposes a novel way of determining frontier locations, which does not rely on explicitly representing the full space; only occupied space is represented. Rather than representing free and unknown space, particles are sampled within known free space. These particles are then used to simulate a gas that is contained by the known occupied space, and frontiers are identified as the places in which the gas expands. In terms of exploration speed, however, their experiments show that the frontier exploration approach by [40] performs better. [30] revisits NBV-based exploration. Instead of sampling poses in the free space, feasible trajectories are sampled using RRT\* [166]. They do not execute a full trajectory, but rather the first part of the planning tree that is common to most trajectories with high utility. Then, the planning step is repeated in a receding horizon fashion. The authors compare their approach with [113] and show superior performance with faster calculation times and more explored space. We will use their publically available method as a second baseline for our comparisons.

### L.3 Flight velocity for optimal energy use

An important motivation for our work is that the limited multi-rotor energy supply is not well used when flying near-hover. This is intuitive: when hovering, energy is used while no new ground is being covered. As we put more energy into motion, less is wasted on hovering. However, very large velocities are also inefficient due to aerodynamic drag. Thus, given a direction of motion, there must be an optimal velocity  $v^*$  at which most distance is traveled per energy use. Let us consider horizontal motion. The optimal velocity can be expressed as:

$$v^* = \arg \max_v \frac{v}{P} \sim \arg \max_v \frac{v}{T} \tag{L.1}$$

where  $P$  is the power used. Since  $P$  is roughly proportional to the thrust  $T$ , we can  $P$  with  $T$ . To estimate a typical  $v^*$ , we consider the multi-rotor model from [258]. We extend their translational dynamics model with an approximation of the aerodynamic drag of the quadrotor body, see Fig. L.2:

## Appendix L. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

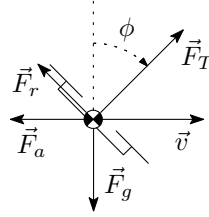


Figure L.2 – Forces acting on the multi-rotor during horizontal flight.

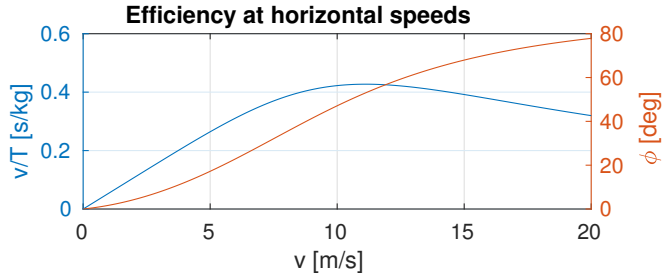


Figure L.3 – Efficiency  $\frac{v}{T}$  and corresponding pitch  $\phi$  as a function of the horizontal velocity  $v$  calculated using the model (L.3). The parameters  $m = 1.9\text{kg}$ ,  $k = \frac{0.21}{gs}$  and  $c_{\text{aero}} = 0.15$  are taken from the quadrotor specifications in [258]. While the values do not represent all multi-rotors, they give an idea of the magnitude of the optimal velocity.

$$\begin{aligned} m\vec{a} &= \vec{F}_T + \vec{F}_g + \vec{F}_r + \vec{F}_a \\ &= T\mathbf{R}\vec{e}_3 - mg\vec{e}_3 - Tk\mathbf{R}\vec{v}\frac{\vec{v}\cdot\mathbf{R}\vec{v}}{|\vec{v}|} - |\vec{v}|c_{\text{aero}}, \end{aligned} \quad (\text{L.2})$$

with  $m$  the multi-rotor mass,  $\vec{a}$  the acceleration,  $\mathbf{R}$  the rotation matrix representing the multi-rotor attitude,  $\vec{e}_3$  the unit vector in  $z$ -direction of the world frame,  $g$  the gravitational constant,  $k$  a first-order drag coefficient due to rotors, derived in [258] and  $c_{\text{aero}}$  the second-order drag coefficient due to the multi-rotor body. We make the assumption that  $c_{\text{aero}}$  is isotropic. In steady-state motion,  $\vec{a} = 0$  and all forces act in a plane spanned by  $\vec{v}$  and  $\vec{e}_3$ . Then, considering horizontal motion, (L.2) can be decomposed into a horizontal and vertical component:

$$\begin{aligned} c_{\text{aero}}v^2 + \cos^2\phi kTv &= \sin\phi T \\ \sin\phi \cos\phi kTv + \cos\phi T &= mg \end{aligned} \quad (\text{L.3})$$

where  $\phi$  is the pitch angle. Using the multi-rotor specifications from [258], we solve this system of equations for different  $v$  and plot  $\frac{v}{T}(v)$  in Fig. L.3. As we can see,  $v^*$  is just above  $10\frac{m}{s}$ , with the pitch just above  $50^\circ$ , which motivates flight at high speeds. In practice, there is an upper limit on the velocity  $v_{\text{max}}$  due to thrust limits, obstacle avoidance reaction time and speed limits of state estimation. We will henceforth assume  $v^* > v_{\text{max}}$  and consider  $v_{\text{max}}$  a parameter.

To enable effective obstacle avoidance, we align the depth sensor, for which we assume a limited field of view, with the flight direction  $\vec{v}$  as much as is possible using only yaw.

## L.4 Methodology

We approximate the environment with a regular 3D voxel grid  $V = \{\vec{x}\}$  with voxel dimensions  $s^3$ , where each voxel is represented by its centroid. Voxels are labeled free, occupied or unknown:  $V = V_{\text{free}} \cup V_{\text{occ}} \cup V_{\text{unk}}$ . The labeling is obtained from the robot’s RGBD sensor measurements using OctoMap [137], assuming a perfect pose estimate but noisy depth measurements. We define the global set of frontiers

$$\mathcal{F}_g := \{\vec{x}_i \in V_{\text{free}} : \exists \vec{x}_j \in V_{\text{unk}} : |\vec{x}_i - \vec{x}_j| = s\}. \quad (\text{L.4})$$

Previous frontier-based approaches typically decide where to move next by considering all  $\mathcal{F}_g$ . One approach is to run the Dijkstra algorithm [68], starting from the position of the robot and allowing transitions between adjacent  $\vec{x} \in V_{\text{free}}$  until a frontier  $\vec{x} \in \mathcal{F}_g$  is reached [151]. [30] samples possible trajectories using RRT\* and moves towards the most promising direction. What can be observed with these approaches is that they require time to calculate, resulting in a stop-and-go behavior in which the robot moves for a while, stops to perform the calculation, then moves again. Furthermore, the goal frontier often lies behind the robot. A multi-rotor flying at a non-zero velocity would thus need to reverse its flight direction and re-visit ground that it has already covered. Evidently, all frontiers need to be visited sooner or later. But we find that choosing a direction and maintaining it, and only later returning, can result in faster coverage than going back and forth in a breadth-first-search manner, especially when flying at high speeds.

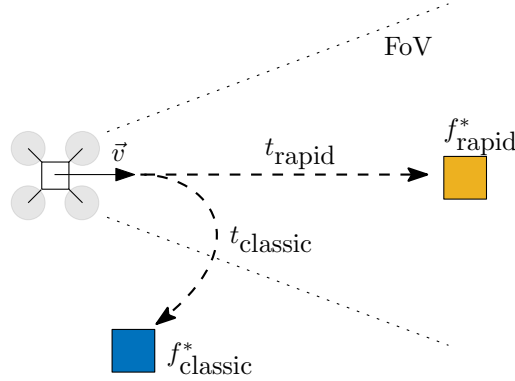
To avoid these problems, we propose to restrict the considered frontiers to those that are in the current field of view:  $\mathcal{F}_v \subseteq \mathcal{F}_g$ . This has three advantages: firstly,  $\mathcal{F}_v$  can be calculated very efficiently from the operations that anyways need to be performed at every depth measurement. Secondly, the size of  $\mathcal{F}_v$  is small enough that it allows a rapid decision and makes path planning inexpensive. Finally, any  $\vec{x} \in \mathcal{F}_v$  will lie in flight direction of the multi-rotor, avoiding the effort to change the current velocity and avoiding flying back and forth, see Fig. L.4. One could consider a different restriction, such as picking frontiers that lie in front of the robot, but that would require to perform a query in Octomap, whereas obtaining  $\mathcal{F}_v$  can be easily combined with the most recent incorporation of depth measurements. Furthermore,  $\mathcal{F}_v$  is guaranteed not to contain any frontiers that are currently occluded and thus not directly reachable. Note that  $\mathcal{F}_v$  depends on the field of view of the robot - a broader field of view will result in a larger  $\mathcal{F}_v$ .

From  $\mathcal{F}_v$  we select the frontier  $\vec{x}_i^*$  with the lowest cost  $c_i = |\vec{v}_i - \vec{v}_{\text{curr}}|$ , the norm of the difference between the current velocity and the desired velocity at the frontier,  $\vec{v}_i$ . Recall from Section L.3 that we on one hand aim to fly at  $v_{\text{max}}$  as much as possible. On the other hand, we want to have a velocity that allows us to react to yet unknown obstacles. Thus, we define the desired velocity for a frontier as

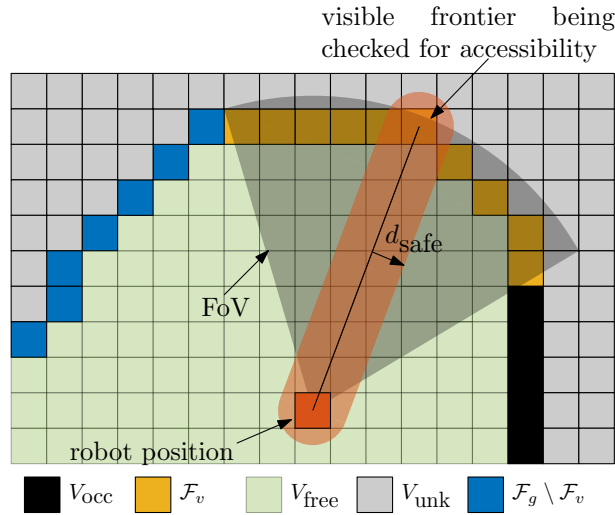
$$\vec{v}(\vec{x}_i) = (\vec{x}_i - \vec{x}_r) \cdot \frac{v_{\text{max}}}{r_{\text{sensor}}}, \quad (\text{L.5})$$

$\vec{x}_r$  being the robot position. For frontiers that are at the detection range  $r_{\text{sensor}}$  of the

## Appendix L. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight



**Figure L.4** – The gist of our approach is that in frontier selection, we prefer frontiers which lie in flight direction. This allows continuous flight at high speeds and reduces the time spent flying through known areas.



**Figure L.5** – Two dimensional illustration of determining the accessibility of members of the visible frontier set.

depth sensor, the desired velocity will be  $v_{\max}$  and pointing towards the unknown volume, while for frontiers closer to the robot the desired velocity will be lower. This imposes a slower approach of frontiers behind which there might be nearby obstacles.

Once we have determined  $\vec{x}_i^*$ , we set the robot velocity to  $\vec{v}_i$ . Lest the robot tries to reach a physically unreachable frontier, we further restrict the choice of frontiers to the ones that are accessible from the current position:

$$\mathcal{F}_a = \{\vec{x} \in \mathcal{F}_v : \mathcal{A}(\vec{x})\} \quad (\text{L.6})$$

$$\mathcal{A}(\vec{x}) : \nexists \vec{x}_{occ} \in V_{occ}, \vec{x}_s \in S_r(\vec{x}) : |\vec{x}_{occ} - \vec{x}_s| < d_{\text{safe}} \quad (\text{L.7})$$

with  $S_r(\vec{x})$  the set of points on the segment between the current position and the frontier  $\vec{x}$  and  $d_{\text{safe}}$  a minimum safety radius (see Fig. L.5). The control loop runs every 20ms.

As soon as  $\mathcal{F}_a = \emptyset$ , the algorithm switches to a classical frontier selection method, which is implemented as discussed in [151]: a shortest path with waypoints  $W(\vec{x}_i) =$



$\{\vec{w}_1, \vec{w}_2, \dots, \vec{w}_m\}$  to any frontier  $\vec{x}_i \in \mathcal{F}_g$  is sought using the Dijkstra algorithm. If no such path is found, exploration is considered complete. Otherwise, the robot will track  $W$ . This path tracking is aborted if one of two conditions occurs: if  $\vec{w}_m$  has been reached or is unreachable, the frontier  $\vec{x}_i$  that had generated the path is removed from  $\mathcal{F}_g$  and a new path  $W$  is calculated. Alternatively, if at any point of the waypoint tracking  $\mathcal{F}_a \neq \emptyset$ , the control switches back to the rapid frontier selection method.

In order to fly close to  $v_{\max}$  during  $W$  tracking, the multi-rotor velocity is set such that it flies at the fastest safe speed towards the furthest accessible waypoint:

$$\vec{v}_r \leftarrow \vec{v}(\arg \max_{\vec{w}_j} j : \mathcal{A}(\vec{w}_j)) \quad (\text{L.8})$$

where  $\vec{v}(\vec{x})$  is the same as in (L.5), with the norm truncated to  $v_{\max}$ .

## L.5 Experiments

In order to evaluate and compare the performance of the proposed approach, simulation studies have been performed using RotorS [102] and Gazebo. Furthermore, a real world experiment has been conducted to demonstrate the rapid exploration algorithm on a real quadrotor. In all experiments we use the control architecture from [76].

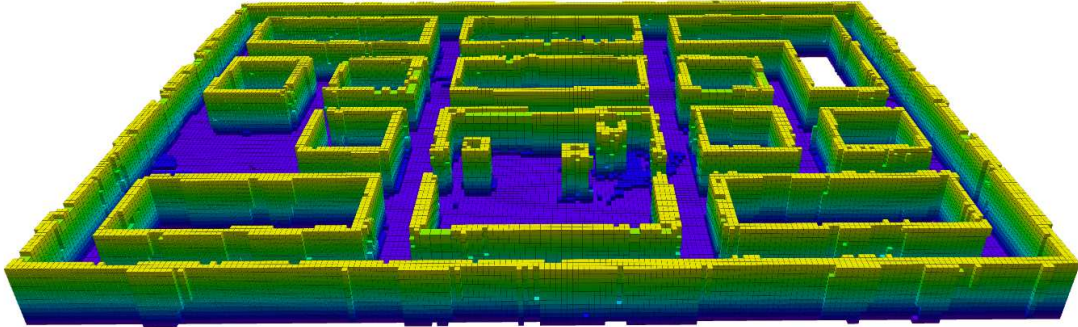
### L.5.1 Simulation

The simulated multi-rotor is equipped with a depth camera mounted in a forward-looking configuration. The stereo camera has a field of view (FoV) of  $[60, 115]^\circ$  in vertical and horizontal direction. The proposed algorithm (*Rapid*) is compared with the classic frontier-based exploration as implemented in [151] (*Classic*) as well as with the next-best-view approach presented in [30] (*NBV*). To make a fair comparison, we have tried to make the quadrotor fly at the same maximum speed  $v_{\max}$  for all approaches, as far as this is possible with the selected trajectories. To that end, we employ the same waypoint-based navigation for *Classic* as discussed in Section L.4. *NBV* is limited to low speeds and we were not able to deploy it at the full range of velocities that we evaluate. The reason for this speed limitation is the way the quadrotor navigates to its new goal. Instead of generating a smooth trajectory, the quadrotor attempts to navigate to the next node of the generated random tree in a straight line. Since this results in instantaneous changes of direction at every node of the tree, for velocities above 0.7 meters per second the quadrotor fails to follow the trajectory. This often results in a crash into nearby obstacles. Furthermore, the maximum yaw rate was limited to 0.75 rad/s for *NBV* for the same reasons.

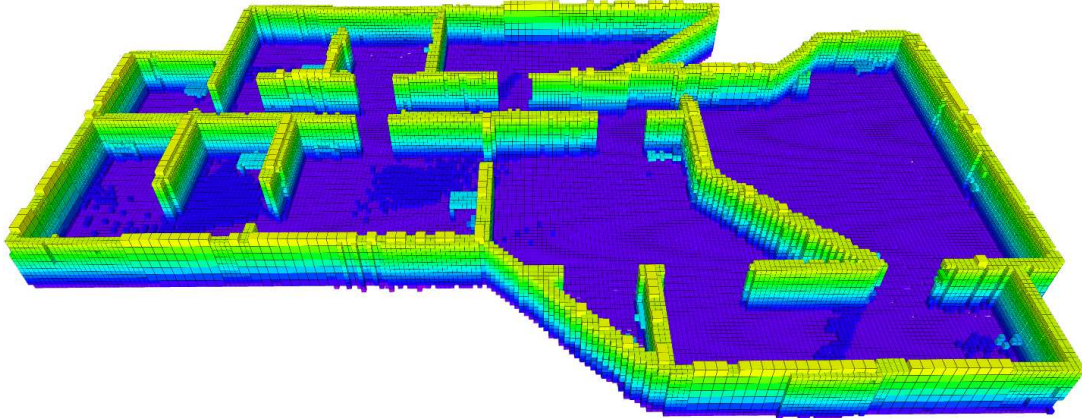
We perform simulations on three different scenarios: **Juliá**, an environment mimicking *Scenario 2* from [151], see Fig. L.6. This 2D scenario is dominated by corridors and contains two open spaces. **Office**, another 2D scenario with a more open, office-like layout as depicted in Fig. L.7. And finally **Powerplant**, a 3D scenario of a powerplant

**Appendix L. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight**

---



**Figure L.6** – Fully explored Juliá scenario, with dimensions 38x26x3 m. For visualization reasons, the map is truncated at a height of 2.5 meters.



**Figure L.7** – Fully explored Office scenario, with dimensions 38x23x3 m. For visualization reasons, the map is truncated at a height of 2.5 meters.

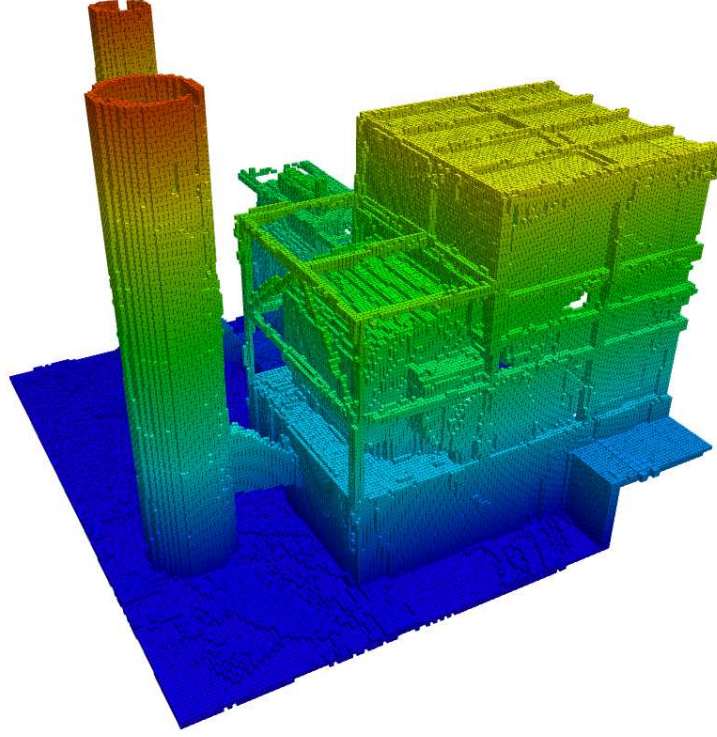


Figure L.8 – Fully explored Powerplant scenario, with dimensions 33x31x26 m.

Parameter	Value	Parameter	Value
$v_{\max}$	{0.3, 0.7, 1.5, 2.5}m/s	FoV	[60x115] $^{\circ}$
Resolution	0.2m	$\dot{\varphi}_{\max}$	1.5rad/s
$d_{\text{safe}}$	0.6m	$\dot{\varphi}_{\max}^{\text{NBV}}$	0.75rad/s

Table L.1 – Parameters common in the different simulation scenarios.

obtained from the Gazebo model library<sup>1</sup>. The original powerplant model is cropped to smaller dimensions as depicted in Fig. L.8. Specific parameters for the scenarios are listed in tables L.1 and L.2. Parameters  $d_{\max}^{\text{planner}}$ ,  $\lambda$ ,  $N_{\max}$  and the maximum edge length of the RRT tree refer to the setup of NBV and are explained in [30]. Since exploration performance can be dependent on the initial position of the robot, simulations were performed for multiple initial positions. Furthermore, the maximum velocity of the quadrotor was varied and a complete set of six simulations was performed for each velocity. Note that our method is defined in 3D. Thus, the 2D scenarios are extruded to

<sup>1</sup>[https://bitbucket.org/osrf/gazebo\\_models/src](https://bitbucket.org/osrf/gazebo_models/src)

	Juliá	Office	Powerplant
<b>Dimensions [m]</b>	38x26x3	38x23x3	33x31x26
$d_{\max}^{\text{sensor}}$ [m]	5.0	5.0	7.0
$d_{\max}^{\text{planner}}$ [m]	1.5	1.5	2.0
$\lambda$	0.3	0.2	0.2
$N_{\max}$	20	20	30
<b>RRT max edge length [m]</b>	1.0	1.0	3.0

Table L.2 – Parameters changing for the different scenarios.

## Appendix L. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

Parameter	Value	Parameter	Value
$v_{\max}$	$\{0.3, 0.7, 1, 1.5, 2\}$ m/s	FoV	$[43 \times 56]^\circ$
Resolution	0.2m	$\dot{\varphi}_{\max}$	1.5rad/s
$r_{\text{sensor}}$	$\{3, 3.5, 4\}$ m	$d_{\text{safe}}$	0.7m

Table L.3 – Parameters used for the indoor experiment.

a height of  $2.5m$  and closed with floor and ceiling. The height is low enough so that the area can be covered while flying at a single altitude.

### L.5.2 Real World Experiments

To verify that the speeds achieved in simulation can also be reached in the real world, we implemented our exploration algorithm on a real quadrotor, and made it explore both an indoor and an outdoor scenario: The indoor scenario is a room with dimensions  $6.5 \times 6.8 \times 2.6$  meters, see Fig. L.9, and the outdoor scenario is a forest, see Fig. L.10. Since there were no natural bounds in the outdoor scenario, we artificially restricted the motion of the quadrotor to a bounding box expressed relative to its starting point. This results in an inconsistent amount of free space between different runs, and we had to change the experiment location frequently due to lighting conditions, so we only present quantitative results for the indoor scenario. The depth sensor used for both real world scenarios was the Intel RealSense R200 with a FoV of  $[56, 43]$  degrees in horizontal and vertical direction, respectively, and was mounted in a forward-looking configuration at a pitch angle of 0 degrees. A summary of the parameters applied in the indoor scenario is given in Table L.3. Note that the sensor range is set at different values that are all below the actual range of the sensor. We have done this to require the quadrotor to move – had we used the full sensor range, the quadrotor would not have needed to move very far to explore the room at hand. At higher velocities, however, we need to increase the sensor range such that the quadrotor is able to react to obstacles. The outdoor experiments were performed at  $\{1, 1.5, 2\} \frac{m}{s}$  and with  $r_{\text{sensor}} = 5m$ . For state estimation, we use the pipeline described in [76], which relies on visual odometry by SVO [96] that is fused with an IMU estimate using MSF [215].

### L.5.3 Measurements

In each experiment, the count of cells currently estimated free  $|V_{\text{free}}|(t)$ , the state at which the robot is in (rapid exploration, calculating  $W$  or tracking  $W$ ) as well as the current position and velocity of the quadrotor are sampled at  $5Hz$ . From this, the coverage ratio  $\frac{|V_{\text{free}}|(t)}{|V_{\text{free}}^*|}$ , where  $V_{\text{free}}^*$  is the actual free space, and the distance traveled  $d_{\text{tot}}(t)$  are measured for all samples at times  $t$ . To compare the approaches, we report as main performance metric the time at which the scenario is fully explored  $t_{\text{max}}$ , the total distance traveled  $d_{\text{max}}$  and the expected voxel discovery time

$$t_{\text{exp}} = \frac{1}{|V_{\text{free}}^*|} \sum_{\vec{x} \in V_{\text{free}}^*} \min\{t : \vec{x} \in V_{\text{free}}(t)\}. \quad (\text{L.9})$$



Figure L.9 – Quadrotor flying in the indoor real world scenario.



Figure L.10 – Quadrotor flying in the outdoor real world scenario.

## Appendix L. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

		Rapid			NBV Planner			Classic Frontier		
	$v_{\max}[\frac{m}{s}]$	$d_{\max}[m]$	$t_{\max}[s]$	$t_{\exp}[s]$	$d_{\max}[m]$	$t_{\max}[s]$	$t_{\exp}[s]$	$d_{\max}[m]$	$t_{\max}[s]$	$t_{\exp}[s]$
Julia	0.3	262 ± 8.9	1074 ± 55	379 ± 31				268 ± 24	1261 ± 124	490 ± 34
	0.7	273 ± 15	553 ± 59	202 ± 13				258 ± 14	636 ± 28	264 ± 15
	1.5	310 ± 28	408 ± 49	121 ± 15				263 ± 19	531 ± 78	221 ± 35
	2.5	315 ± 30	360 ± 39	105 ± 8.7				260 ± 26	505 ± 58	211 ± 14
Office	0.3	223 ± 14	866 ± 55	318 ± 29	466 ± 30	1698 ± 158	573 ± 91	275 ± 11	1266 ± 50	452 ± 74
	0.7	237 ± 20	471 ± 53	178 ± 22	474 ± 54	983 ± 96	346 ± 49	272 ± 26	657 ± 73	278 ± 30
	1.5	253 ± 12	332 ± 29	117 ± 4.1				261 ± 25	433 ± 38	176 ± 41
Powerplant	0.7	692 ± 53	1245 ± 151	364 ± 31	1363 ± 290	2104 ± 406	613 ± 131	692 ± 57	2397 ± 170	852 ± 86
	1.5	710 ± 65	717 ± 94	198 ± 2.1				692 ± 32	1519 ± 88	567 ± 31
	2.5	728 ± 49	582 ± 26	150 ± 3.9				684 ± 56	1437 ± 123	565 ± 80

**Table L.4** – Mean and standard deviation for the total distance traveled  $s_{\max}$ , the total time spent exploring  $t_{\max}$  and the expected cell discovery time  $t_{\exp}$  across all experiments.

The last metric is more meaningful than  $t_{\max}$  for scenarios where the goal is to find several objects in an unknown environment as quickly as possible (see related discussion in [151]).

We have found that with our implementation of Dijkstra, the robot often spends a significant amount of time calculating the path  $W$ . Especially in later stages of the exploration process, where the path to the next frontier tends to become longer, the path calculation can take up to 10 seconds. In order to estimate only the quality of the resulting flight behavior, we adjust for this calculation time by not counting time spent on it when calculating the above performance metrics. This mainly benefits  $t_{\max}$  and  $t_{\text{mean}}$  of the classic frontier approach. The calculation time of both the reactive mode and of the NBV method are negligible, and so we do not need to adjust for them.

## L.6 Results

### L.6.1 Simulation

The main results are reported in Table L.4. For the office scenario, the comparison of the main performance metrics for different values of  $v_{\max}$  are visualized in Figs. L.11, L.12 and L.13. As can be seen, our approach consistently outperforms classical Frontier-based exploration and next-best-view exploration across all scenarios with respect to  $t_{\max}$ . The improvement is even more significant for  $t_{\exp}$ , as in our approach there is often a significant amount of time at the end used only for re-visiting frontiers that have been previously skipped. This behavior is well illustrated in Fig. L.1, which shows the evolution of  $\frac{|V_{\text{free}}|}{|V_{\text{free}}^*|}(t)$  and  $d_{\text{tot}}(t)$  for one instance of *Rapid* and *Classic* each in scenario Juliá,  $v_{\max} = 2.5$ . Within the approaches,  $t_{\max}$  and  $t_{\exp}$  decrease as  $v_{\max}$  increases, as can be expected. However, as our approach is designed for high speeds, its decrease in

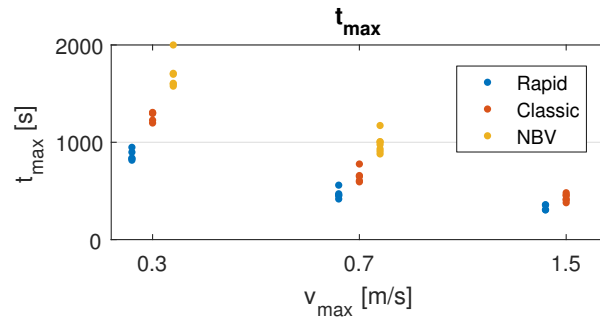


Figure L.11 – Exploration times in office scenario. Each dot represents one of six runs.

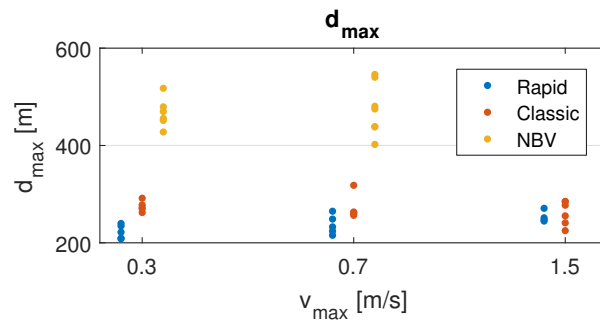


Figure L.12 – Traveled path in office scenario. Each dot represents one of six runs.

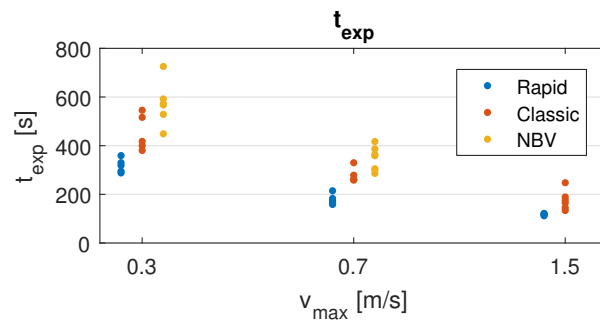


Figure L.13 – Expected voxel exploration time for office scenario. Each dot represents one of six runs.

## Appendix L. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

---

$t_{\max}$  and  $t_{\exp}$  is most significant, in particular in the Juliá and Powerplant scenarios.

We are surprised by the poor results that we obtained from NBV, in spite of parameter tuning. In particular, in Juliá it consistently gets stuck in one part of the map and does not move towards the unexplored regions. Thus, we cannot report any results for that scenario. For the scenarios in which it did succeed, its performance was mostly inferior to the other two approaches. This is consistent with the results in [135], which shows longer exploration times for a previous NBV-based approach [113], compared to a classic frontier-based approach. In Fig. L.14 it can be seen that the NBV trajectory is less smooth than the trajectories performed by the other two approaches.

Of particular interest is the behavior of  $d_{\max}$ . Within the runs of both the NBV planner and classic frontier-based exploration,  $d_{\max}$  is not significantly affected by  $v_{\max}$ . For the proposed exploration algorithm, however, it increases with  $v_{\max}$ . We assume that this happens because a higher  $v_{\max}$  will cause the multi-rotor both to overshoot in dead ends and to make wider turns. The comparison of  $d_{\max}$  between Rapid and Classic Frontier exploration is not consistent. For Juliá and Powerplant,  $d_{\max}$  is generally higher for our approach than for Classic Frontier exploration. However,  $d_{\max}$  is lower for our approach in the Office scenario. Here, we discern two effects: on one hand, our approach does not minimize distance traveled as does the Classic Frontier-based approach. On the other hand, our approach, which is more reactive, results in smoother trajectories. As can be seen in Fig. L.14, this is beneficial in the Office scenario. In a scenario which is more cluttered and which imposes more narrow turns, we would expect a smaller difference in overall performance between our approach and classic frontier based exploration.

Another observation that can be made is that classical frontier based exploration outperforms NBV exploration in the two dimensional case of office exploration. In the 3D case, however, classic frontier based exploration shows poor performance. A reason for this drop in performance is the limited field of view of the sensor. While in the 2D case, the sensor usually manages to simultaneously detect floor and ceiling at the same time, in the large 3D environment this is no longer the case. As a result, the quadrotor selects frontiers on the border of the view frustum of the camera and moves step-wise up or down, without large movements in the  $xy$ -plane. The NBV controller is better able to handle this and outperforms the classical frontier-based method. Rapid exploration, however, again shows the best performance in the 3D case, since the quadrotor will fly with the maximum velocity for long distances.

### L.6.2 Real World Experiments

The quadrotor successfully managed to map the real world scenarios at the tested speeds. Table L.5 shows the performance of the proposed approach on the indoor scenario with respect to the three metrics for various  $v_{\max}$ . Apart from  $v_{\max} = \{0.3, 0.7\} \frac{m}{s}$ , the total exploration time  $t_{\max}$ , as well as the expected exploration time  $t_{\exp}$  decrease with larger velocities. Since only one run was performed for each velocity, we consider  $v_{\max} = \{0.3, 0.7\} \frac{m}{s}$  to be outliers.



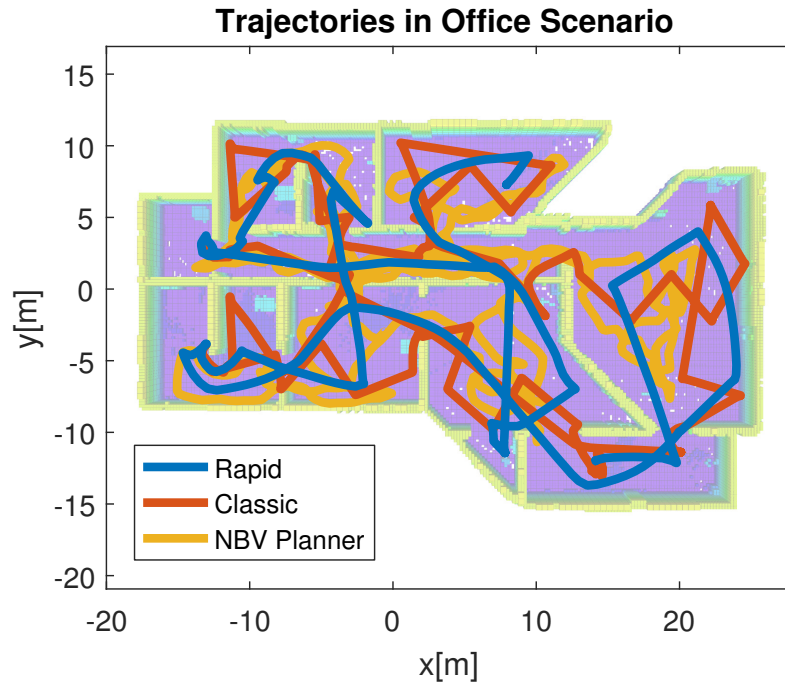


Figure L.14 – Comparison of trajectories in Office scenario.

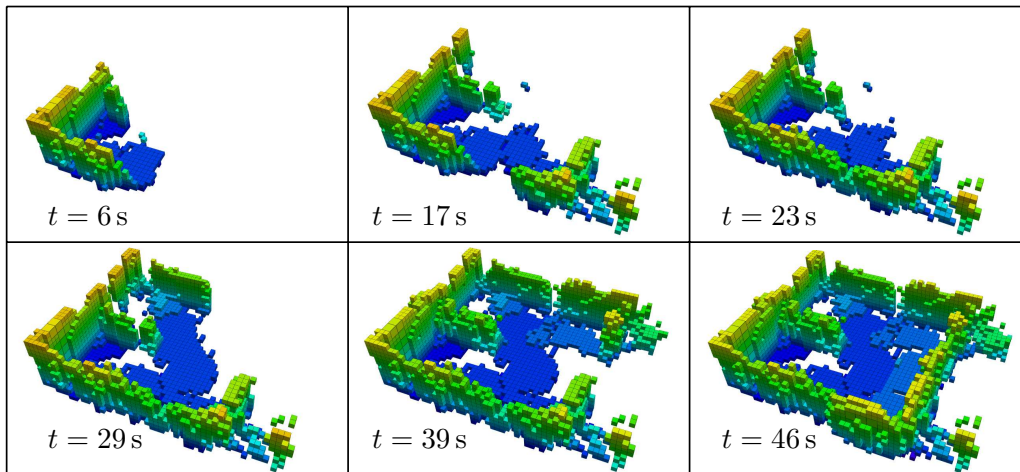


Figure L.15 – Real world exploration of an empty room that is halfway separated by a wall. The top speed of the quadrotor is  $2 \frac{m}{s}$ .

## Appendix L. Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High-Speed Flight

$v_{\max} [\frac{m}{s}]$	Rapid		
	$d_{\max} [m]$	$t_{\max} [s]$	$t_{\exp} [s]$
<b>0.3</b>	29.4	125.7	42.6
<b>0.5</b>	21.3	66.9	26.2
<b>0.7</b>	27.7	77.6	27.8
<b>1.0</b>	24.1	63.3	23.5
<b>1.5</b>	26.4	54.4	19.9
<b>2.0</b>	28.7	52.5	23.6

**Table L.5** – Total distance traveled  $d_{\max}$ , the total time spent exploring  $t_{\max}$  and the expected cell discovery time  $t_{\exp}$  obtained for the indoor real world experiment.

Most of the failures that were experienced in the real world can be attributed to one of three causes: firstly, pose estimation failures, particularly with rapid changes in attitude above terrain. This is due to the visual odometry being based on the image of a down-looking camera, whose image would rapidly change. To prevent such failures indoors, we set the room up such that attitude above terrain changes would be minimal. Outdoors, we chose locations dominated by tree trunks, with little vegetation on the ground. Secondly, the RealSense sensor would sometimes fail to see objects, in particular the curtain that we had on one side of the room, which would rapidly flap as the quadrotor would approach it. Clamping the bottom of the curtain significantly decreased the frequency at which this problem occurred. Thirdly, the geometry of the sensor placement and field of view resulted in a blind spot for obstacles at certain low-radius turns executed in the reactive mode. This problem was solved by artificially restricting the field of view when considering the set of visible frontiers  $\mathcal{F}_v$ .

Fig. L.15 depicts the indoor exploration process at six distinct time instances and illustrates the growth of the map representing the environment at  $v_{\max} = 2 \frac{m}{s}$ . Note that after starting the exploration task, the quadrotor waits for 3 seconds to assure a sufficient OctoMap representation of the environment before it starts to navigate to frontiers. For more insights, we invite the reader to take a look at the multimedia material at <https://youtu.be/54s6gGZLpJo>.

### L.7 Conclusion

Within this work, an exploration algorithm was proposed that is designed specifically for multi-rotor exploration at high speeds. The reactive behavior of the algorithm allows for fast incorporation of new information and results in efficient trajectories. Compared to classic frontier-based exploration, the approach can occasionally exhibit a small increase in the total path traveled to explore an area, but at the same time achieves smaller exploration times for the same maximum velocity constraint. These properties are demonstrated in multiple simulations for different exploration scenarios and validated with real world experiments.

# Bibliography

- [1] “A.I. Is Flying Drones (Very, Very Slowly)”. In: *The New York Times* (Mar. 26, 2019). URL: <https://www.nytimes.com/2019/03/26/technology/alphapilot-ai-drone-racing.html> (visited on 04/16/2022).
- [2] 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.
- [3] Evan Ackermann. *AI-Powered Drone Learns Extreme Acrobatics*. <https://spectrum.ieee.org/automaton/robotics/drones/ai-powered-drone-extreme-acrobatics>. [Online; accessed 21.6.2021]. 2020.
- [4] Enrico Ajanic, Mir Feroskhan, Stefano Mintchev, Flavio Noca, and Dario Floreano. “Bioinspired wing and tail morphing extends drone flight capabilities”. In: *Science Robotics* 5.47 (2020). DOI: 10.1126/scirobotics.abc2897. eprint: <https://robotics.sciencemag.org/content/5/47/eabc2897.full.pdf>. URL: <https://robotics.sciencemag.org/content/5/47/eabc2897>.
- [5] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, and Raphael Ribas. “Solving rubik’s cube with a robot hand”. In: *arXiv preprint arXiv:1910.07113* (2019).
- [6] R. Allen and M. Pavone. “A Real-Time Framework for Kinodynamic Planning with Application to Quadrotor Obstacle Avoidance”. In: *AIAA Guidance, Navigation, and Control Conference*. 2016. DOI: 10.2514/6.2016-1374.
- [7] “AlphaPilot AI Drone Innovation Challenge”. In: <https://lockheedmartin.com/en-us/news/events/ai-innovation-challenge.html> (Jan. 2020). URL: <https://lockheedmartin.com/en-us/news/events/ai-innovation-challenge.html>.
- [8] Gokul Anandayuvraj. *Drones: The Future Of Business?* <https://www.forbes.com/sites/forbesbusinesscouncil/2020/06/08/drones-the-future-of-business/>. [Online; accessed 6.10.2021]. 2020.
- [9] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. “CasADi – A software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* (2018).
- [10] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. “An Introduction to MCMC for Machine Learning”. In: *Mach. Learn.* 50.1-2 (2003), pp. 5–43.

## Bibliography

---

- [11] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. “Learning to learn by gradient descent by gradient descent”. In: *Advances in neural information processing systems* 29 (2016).
- [12] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, and Alex Ray. “Learning dexterous in-hand manipulation”. In: *Int. J. Robot. Research* (2020).
- [13] Amado Antonini, Winter Guerra, Varun Murali, Thomas Sayre-McCord, and Sertac Karaman. “The Blackbird UAV dataset”. In: *Int. J. Robot. Research* 39.10-11 (2020), pp. 1346–136.
- [14] Tomas Baca, Matej Petrlik, Matous Vrba, Vojtech Spurny, Robert Penicka, Daniel Hert, and Martin Saska. “The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles”. In: *J. Intell. Rob. Syst.* 102.1 (Apr. 2021), p. 26. ISSN: 1573-0409. DOI: [10.1007/s10846-021-01383-5](https://doi.org/10.1007/s10846-021-01383-5).
- [15] 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).
- [16] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. “Emergent tool use from multi-agent autocurricula”. In: *arXiv preprint arXiv:1909.07528* (2019).
- [17] Somil Bansal, Anayo K Akametalu, Frank J Jiang, Forrest Laine, and Claire J Tomlin. “Learning quadrotor dynamics using neural network for flight control”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 4653–4660.
- [18] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire J. Tomlin. “Combining Optimal Control and Learning for Visual Navigation in Novel Environments”. In: *Conference on Robot Learning, CoRL 2019*. Vol. 100. Proceedings of Machine Learning Research. PMLR, 2019, pp. 420–429. URL: <http://proceedings.mlr.press/v100/bansal20a.html>.
- [19] Engin Baris, Colin P Britcher, and George Altamirano. “Wind Tunnel Testing of Static and Dynamic Aerodynamic Characteristics of a Quadcopter”. In: *AIAA Aviation 2019 Forum*. 2019, p. 2973.
- [20] 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](https://doi.org/10.1002/rob.21741).
- [21] L. Bauersfeld, L. Spannagl, G. Ducard, and C. Onder. “MPC Flight Control for a Tilt-rotor VTOL Aircraft”. In: *IEEE Transactions on Aerospace and Electronic Systems* (2021), pp. 1–13. DOI: [10.1109/TAES.2021.3061819](https://doi.org/10.1109/TAES.2021.3061819).
- [22] Leonard Bauersfeld and Davide Scaramuzza. “Range, Endurance, and Optimal Speed Estimates for Multicopters”. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2953–2960. DOI: [10.1109/LRA.2022.3145063](https://doi.org/10.1109/LRA.2022.3145063).

- 
- [23] Leonard Bauersfeld\*, Elia Kaufmann\*, Philipp Foehn, Sihao Sun, and Davide Scaramuzza. “NeuroBEM: Hybrid Aerodynamic Quadrotor Model”. In: *RSS: Robotics, Science, and Systems* (2021).
- [24] Suneel Belkhale, Rachel Li, Gregory Kahn, Rowan McAllister, Roberto Calandra, and Sergey Levine. “Model-based meta-reinforcement learning for flight with suspended payloads”. In: *IEEE Robot. Autom. Lett.* (2021).
- [25] Oren Ben-Kiki, Clark Evans, and Ingy Döt Net. *YAML Ain’t Markup Language (YAML™) Version 1.2*. <https://yaml.org/spec/1.2/spec.pdf>. Accessed: 2021-7-20. Oct. 2009.
- [26] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Jozefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d.O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. “Dota 2 with large scale deep reinforcement learning”. In: *arXiv preprint arXiv:1912.06680* (2019).
- [27] Dimitri P Bertsekas. *Dynamic programming and optimal control*. Vol. 1. Athena scientific Belmont, MA, 1995.
- [28] Davide Bicego, Jacopo Mazzetto, Ruggero Carli, Marcello Farina, and Antonio Franchi. “Nonlinear model predictive control with enhanced actuator model for multi-rotor aerial vehicles with generic designs”. In: *J. Intell. Robot. Syst.* (2020).
- [29] Katharina Bieker, Sebastian Peitz, Steven L Brunton, J Nathan Kutz, and Michael Dellnitz. “Deep model predictive flow control with limited sensor data and online learning”. In: *Theoretical and Computational Fluid Dynamics* (2020).
- [30] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. “Receding horizon “next-best-view” planner for 3D exploration”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1462–1468.
- [31] Christopher M Bishop. *Mixture density networks*. Aston University, 1994.
- [32] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. “Robust Visual Inertial Odometry Using a Direct EKF-Based Approach”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2015.
- [33] Michael Blösch, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. “Vision based MAV navigation in unknown and unstructured environments”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2010, pp. 21–28.
- [34] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, and Kurt Konolige. “Using simulation and domain adaptation to improve efficiency of deep robotic grasping”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018.
- [35] Dario Brescianini and Raffaello D’Andrea. “Tilt-prioritized quadcopter attitude control”. In: *IEEE Transactions on Control Systems Technology* (2018). URL: <https://ieeexplore.ieee.org/document/8556372>.

## Bibliography

---

- [36] Pierre-Jean Bristeau, Philippe Martin, Erwan Salaün, and Nicolas Petit. “The role of propeller aerodynamics in the model of a quadrotor UAV”. In: *2009 European control conference (ECC)*. IEEE. 2009, pp. 683–688.
- [37] Adam Bry, Abraham Bachrach, and Nicholas Roy. “State estimation for aggressive flight in GPS-denied environments using onboard sensing”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. IEEE. 2012, pp. 1–8.
- [38] Adam Bry, Charles Richter, Abraham Bachrach, and Nicholas Roy. “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.
- [39] Adam Bry and Nicholas Roy. “Rapidly-exploring Random Belief Trees for motion planning under uncertainty”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2011.
- [40] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. “Coordinated multi-robot exploration”. In: *IEEE Trans. Robot.* 21.3 (2005), pp. 376–386.
- [41] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age”. In: *IEEE Trans. Robot.* (2016).
- [42] Gang Cao, Edmund M-K Lai, and Fakhrol Alam. “Gaussian process model predictive control of an unmanned quadrotor”. In: *J. Intell. Robot. Syst.* (2017).
- [43] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. “Realtime multi-person 2d pose estimation using part affinity fields”. In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2017, pp. 7291–7299.
- [44] Andrea Carron, Elena Arcari, Martin Wermelinger, Lukas Hewing, Marco Hutter, and Melanie N Zeilinger. “Data-driven model predictive control for trajectory tracking with a robotic arm”. In: *IEEE Robot. Autom. Lett.* (2019).
- [45] Eric Chang, Laura Y. Matloff, Amanda K. Stowers, and David Lentink. “Soft biohybrid morphing wings with feathers underactuated by wrist and finger motion”. In: *Science Robotics* 5.38 (2020). DOI: [10.1126/scirobotics.aay1246](https://doi.org/10.1126/scirobotics.aay1246). eprint: <https://robotics.sciencemag.org/content/5/38/eaay1246.full.pdf>. URL: <https://robotics.sciencemag.org/content/5/38/eaay1246>.
- [46] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. “Deepdriving: Learning affordance for direct perception in autonomous driving”. In: *International Conference on Computer Vision (ICCV)*. 2015.
- [47] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. “Learning by Cheating”. In: *Conference on Robot Learning (CoRL)*. 2019.
- [48] 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.
- [49] Su Yeon Choi and Dowan Cha. “Unmanned aerial vehicles using machine learning for autonomous flight; state-of-the-art”. In: *Advanced Robotics* 33.6 (2019), pp. 265–277.

- [50] 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.
- [51] Ronald Clark, Sen Wang, Hongkai Wen, Andrew Markham, and Niki Trigoni. “VINet: Visual-inertial odometry as a sequence-to-sequence learning problem”. In: *AAAI Conference on Artificial Intelligence*. 2017.
- [52] Ignasi Clavera, David Held, and Pieter Abbeel. “Policy Transfer via Modularity and Reward Guiding”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017.
- [53] José Arturo Cocoma-Ortega and José Martínez-Carranza. “Towards high-speed localisation for autonomous drone racing”. In: *Mexican International Conference on Artificial Intelligence*. Springer. 2019.
- [54] Toby Collins and Adrien Bartoli. “Infinitesimal plane-based pose estimation”. In: *Int. J. Comput. Vis.* 109.3 (2014), pp. 252–286.
- [55] Connect Tech Inc. *Quasar Carrier Board*. <https://connecttech.com/product/quasar-carrier-nvidia-jetson-tx2/>. Accessed: 2021-7-20. Aug. 2019.
- [56] Cl Connolly. “The determination of next best views”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. Vol. 2. 1985, pp. 432–435. DOI: [10.1109/ROBOT.1985.1087372](https://doi.org/10.1109/ROBOT.1985.1087372).
- [57] CORDIS - European Commission. *AgileFlight*. <https://cordis.europa.eu/project/id/864042>. Accessed: 2021-7-30.
- [58] Ruben D’Sa, Devon Jenson, and Nikolaos Papanikolopoulos. “SUAV:Q - a hybrid approach to solar-powered flight”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 3288–3294. DOI: [10.1109/ICRA.2016.7487501](https://doi.org/10.1109/ICRA.2016.7487501).
- [59] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. “Loihi: A Neuro-morphic Manycore Processor with On-Chip Learning”. In: *IEEE Micro* 38.1 (Jan. 2018), pp. 82–99. DOI: [10.1109/MM.2018.112130359](https://doi.org/10.1109/MM.2018.112130359).
- [60] Brian L. Day and Richard C. Fitzpatrick. “The vestibular system”. In: *Current Biology* 15.15 (2005), R583–R586. DOI: [10.1016/j.cub.2005.07.053](https://doi.org/10.1016/j.cub.2005.07.053). URL: <https://doi.org/10.1016/j.cub.2005.07.053>.
- [61] C De Wagter, F Paredes-Vallés, N Sheth, and G de Croon. “Learning fast in autonomous drone racing”. In: *Nature Machine Intelligence* 3.10 (2021), pp. 923–923.
- [62] Christophe De Wagter, Federico Paredes-Valles, Nilay Sheth, and Guido de Croon. “The Artificial Intelligence behind the winning entry to the 2019 AI Robotic Racing Competition”. In: *arXiv preprint arXiv:2109.14985* (2021).
- [63] Jeffrey Delmerico, Titus Cieslewski, Henri Rebecq, Matthias Faessler, and Davide Scaramuzza. “Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2019.

## Bibliography

---

- [64] Jeffrey Delmerico and Davide Scaramuzza. “A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)* (2018). DOI: [10.1109/ICRA.2018.8460664](https://doi.org/10.1109/ICRA.2018.8460664).
- [65] Vishnu R Desaraju, Alexander E Spitzer, Cormac O’Meadhra, Lauren Lieu, and Nathan Michael. “Leveraging experience for robust, adaptive nonlinear MPC on computationally constrained systems with time-varying state uncertainty”. In: *Int. J. Robot. Research* (2018).
- [66] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. “Learning modular neural network policies for multi-task and multi-robot transfer”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.
- [67] M. Diehl, H. G. Bock, H. Diedam, and P. B. Wieber. “Fast direct multiple shooting algorithms for optimal robot control”. In: *Fast motions in biomechanics and robotics*. Springer, 2006.
- [68] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.
- [69] DJI. *DJI Digital FPV System*. <https://www.dji.com/fpv>. Accessed: 2021-7-20.
- [70] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. “CARLA: An Open Urban Driving Simulator”. In: *Conference on Robot Learning (CORL)*. 2017, pp. 1–16.
- [71] Paul Drews, Grady Williams, Brian Goldfain, Evangelos A Theodorou, and James M Rehg. “Aggressive deep driving: Combining convolutional neural networks and model predictive control”. In: *Conf. on Robotics Learning (CoRL)*. 2017.
- [72] Guillaume Ducard and Minh-Duc Hua. “Modeling of an unmanned hybrid aerial vehicle”. In: *2014 IEEE Conference on Control Applications (CCA)*. IEEE. 2014, pp. 1011–1016.
- [73] J. Dupeyroux, J. Hagenars, F. Paredes-Valles, and G. de Croon. “Neuromorphic control for optic-flow-based landings of MAVs using the Loihi processor”. In: *Arxiv:2011.00534*. 2020.
- [74] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. “First return, then explore”. In: *Nature* 590.7847 (2021), pp. 580–586. DOI: [10.1038/s41586-020-03157-9](https://doi.org/10.1038/s41586-020-03157-9).
- [75] 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.
- [76] Matthias Faessler, Flavio Fontana, Christian Forster, Elias Mueggler, Matia Pizzoli, and Davide Scaramuzza. “Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor MAV”. In: *J. Field Robot.* 33.4 (2016), pp. 431–450.
- [77] Matthias Faessler, Flavio Fontana, Christian Forster, and Davide Scaramuzza. “Automatic Re-Initialization and Failure Recovery for Aggressive Flight with a Monocular Vision-Based Quadrotor”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2015.



- [78] 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 Robot. Autom. Lett.* 3.2 (Apr. 2018), pp. 620–626. DOI: [10.1109/LRA.2017.2776353](https://doi.org/10.1109/LRA.2017.2776353).
- [79] Davide Falanga, Philipp Foehn, Peng Lu, and Davide Scaramuzza. “PAMPC: Perception-aware model predictive control for quadrotors”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2018.
- [80] 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](https://doi.org/10.1109/LRA.2019.2898117).
- [81] Davide Falanga, Kevin Kleber, Stefano Mintchev, Dario Floreano, and Davide Scaramuzza. “The Foldable Drone: A Morphing Quadrotor That Can Squeeze and Fly”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 209–216. DOI: [10.1109/LRA.2018.2885575](https://doi.org/10.1109/LRA.2018.2885575).
- [82] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. “Dynamic obstacle avoidance for quadrotors with event cameras”. In: *Science Robotics* 5.40 (2020).
- [83] Davide Falanga, Elias Mueggler, Matthias Faessler, and Davide Scaramuzza. “Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017, pp. 5774–5781. DOI: [10.1109/ICRA.2017.7989679](https://doi.org/10.1109/ICRA.2017.7989679).
- [84] Davide Falanga, Alessio Zanchettin, Alessandro Simovic, Jeffrey Delmerico, and Davide Scaramuzza. “Vision-based autonomous quadrotor landing on a moving platform”. In: *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. 2017, pp. 200–207.
- [85] David Fan, Aliakbar Aghamohammadi, and Evangelos Theodorou. “Deep learning tubes for tube MPC”. In: *Robotics: Science and Systems (RSS)* (2020).
- [86] Marius Fehr, Thomas Schneider, and Roland Siegwart. “Visual-Inertial Teach and Repeat Powered by Google Tango”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2018. DOI: [10.1109/IROS.2018.8593416](https://doi.org/10.1109/IROS.2018.8593416).
- [87] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 1126–1135.
- [88] Dario Floreano and Robert J Wood. “Science, technology and the future of small autonomous drones”. en. In: *Nature* 521.7553 (May 2015), pp. 460–466. ISSN: 0028-0836, 1476-4687. DOI: [10.1038/nature14542](https://doi.org/10.1038/nature14542).
- [89] 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.
- [90] Philipp Foehn, Davide Falanga, Naveen Kuppaswamy, Russ Tedrake, and Davide Scaramuzza. “Fast Trajectory Optimization for Agile Quadrotor Maneuvers with a Cable-Suspended Payload”. In: *Robotics: Science and Systems (RSS)*. 2017.

## Bibliography

---

- [91] Philipp Foehn, Angel Romero, and Davide Scaramuzza. “Time-optimal planning for quadrotor waypoint flight”. In: *Science Robotics* 6.56 (2021). DOI: [10.1126/scirobotics.abh1221](https://doi.org/10.1126/scirobotics.abh1221). URL: <https://robotics.sciencemag.org/content/6/56/eabh1221>.
- [92] Philipp Foehn\*, Dario Brescianini\*, Elia Kaufmann\*, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. “AlphaPilot: Autonomous Drone Racing”. In: *Robotics: Science and Systems (RSS)* (2020).
- [93] Philipp Foehn\*, Dario Brescianini\*, Elia Kaufmann\*, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. “AlphaPilot: Autonomous Drone Racing”. In: *Autonom. Rob.* (2021). DOI: [10.1007/s10514-021-10011-y](https://doi.org/10.1007/s10514-021-10011-y).
- [94] Philipp Foehn\*, Elia Kaufmann\*, Angel Romero, Robert Penicka, Sihao Sun, Leonard Bauersfeld, Thomas Laengle, Yunlong Song, Antonio Loquercio, and Davide Scaramuzza. “Agilicious: Open-Source and Open-Hardware Agile Quadrotor for Vision-Based Flight”. In: *Science Robotics* (2021). under review.
- [95] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. “On-Manifold Preintegration for Real-Time Visual-Inertial Odometry”. In: *IEEE Trans. Robot.* 33.1 (2017), pp. 1–21.
- [96] C. Forster, M. Pizzoli, and D. Scaramuzza. “SVO: Fast semi-direct monocular visual odometry”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2014. DOI: [10.1109/ICRA.2014.6906584](https://doi.org/10.1109/ICRA.2014.6906584).
- [97] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. “SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems”. In: *IEEE Trans. Robot.* 33.2 (2017), pp. 249–265.
- [98] Friedrich Fraundorfer, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. “Vision-based autonomous mapping and exploration using a quadrotor MAV”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2012.
- [99] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürri. “Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4257–4264.
- [100] Joseph Funke, Paul Theodosis, Rami Hindiyeh, Ganymed Stanek, Krisada Kritatakirana, Chris Gerdes, Dirk Langer, Marcial Hernandez, Bernhard Müller-Bessler, and Burkhard Huhnke. “Up to the limits: Autonomous Audi TTS”. In: *2012 IEEE Intelligent Vehicles Symposium*. IEEE. 2012, pp. 541–547.
- [101] Paul Furgale and Timothy D. Barfoot. “Visual teach and repeat for long-range rover autonomy”. In: *Journal of Field Robotics* 27.5 (2010), pp. 534–560.
- [102] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. “RotorS—A modular gazebo MAV simulator framework”. In: *Robot Operating System (ROS)*. Springer, 2016.
- [103] Sajad Saeedi G., Carl Thibault, Michael Trentini, and Howard Li. “3D Mapping for Autonomous Quadrotor Aircraft”. In: *Unmanned Syst.* 5.3 (2017), pp. 181–196. DOI: [10.1142/S2301385017400064](https://doi.org/10.1142/S2301385017400064). URL: <https://doi.org/10.1142/S2301385017400064>.

- 
- [104] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Joerg Conradt, Kostas Daniilidis, and Davide Scaramuzza. “Event-based Vision: A Survey”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* (2020).
- [105] Jean Gallier. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN: 1558605991.
- [106] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. “Learning to fly by crashing”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017, pp. 3948–3955.
- [107] Fei Gao, William Wu, Wenliang Gao, and Shaojie Shen. “Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments”. In: *J. Field Robot.* 36.4 (2019), pp. 710–733. DOI: <https://doi.org/10.1002/rob.21842>.
- [108] Alison L Gibbs and Francis Edward Su. “On choosing and bounding probability metrics”. In: *International Statistical Review* 70.3 (2002), pp. 419–435.
- [109] Wojciech Giernacki, Mateusz Skwarczyński, Wojciech Witwicki, Pawel Wroński, and Piotr Koziński. “Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering”. In: *International Conference on Methods and Models in Automation and Robotics (MMAR)*. 2017, pp. 37–42. DOI: [10.1109/MMAR.2017.8046794](https://doi.org/10.1109/MMAR.2017.8046794).
- [110] Rajan Gill and Raffaello D’Andrea. “Propeller thrust and drag in forward flight”. In: *2017 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE. 2017, pp. 73–79.
- [111] Rajan Gill and Raffaello D’Andrea. “Computationally Efficient Force and Moment Models for Propellers in UAV Forward Flight Applications”. In: *Drones* 3.4 (2019), p. 77.
- [112] Alessandro Giusti, Jérôme Guzzi, Dan C. Cireşan, Fang-Lin He, Juan P. Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, Davide Scaramuzza, and Luca M. Gambardella. “A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots”. In: *IEEE Robot. Autom. Lett.* 1.2 (2016).
- [113] Hector H Gonzalez-Banos and Jean-Claude Latombe. “Navigation strategies for exploring indoor environments”. In: *Int. J. Robot. Research* 21.10-11 (2002), pp. 829–848.
- [114] Melissa Greeff, SiQi Zhou, and Angela P Schoellig. “Fly Out The Window: Exploiting Discrete-Time Flatness for Fast Vision-Based Multirotor Flight”. In: *IEEE Robot. Autom. Lett.* 7.2 (2022), pp. 5023–5030.
- [115] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. “Neuroanimator: Fast neural network emulation and control of physics-based models”. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 1998, pp. 9–20.

## Bibliography

---

- [116] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gabor Bartok, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. *TF-Agents: A library for Reinforcement Learning in TensorFlow*. <https://github.com/tensorflow/agents>. [Online; accessed 25-June-2019]. 2018. URL: <https://github.com/tensorflow/agents>.
- [117] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. “FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2019.
- [118] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. “Learning invariant feature spaces to transfer skills with reinforcement learning”. In: *International Conference on Learning Representation (ICLR)* (2017).
- [119] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. “Latent space policies for hierarchical reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1851–1860.
- [120] Raia Hadsell, Pierre Sermanet, Jan Ben, Ayse Erkan, Marco Scoffier, Koray Kavukcuoglu, Urs Muller, and Yann LeCun. “Learning long-range vision for autonomous off-road driving”. In: *J. Field Robot.* 26.2 (2009), pp. 120–144.
- [121] Zhichao Han, Zhepei Wang, Neng Pan, Yi Lin, Chao Xu, and Fei Gao. “Fast-Racing: An Open-Source Strong Baseline for SE(3) Planning in Autonomous Drone Racing”. In: *IEEE Robot. Autom. Lett.* 6.4 (2021), pp. 8631–8638.
- [122] Zhichao Han, Ruibin Zhang, Neng Pan, Chao Xu, and Fei Gao. “Fast-tracker: A robust aerial system for tracking agile target in cluttered environments”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2021, pp. 328–334.
- [123] Drew Hanover, Philipp Foehn, Elia Kaufmann, and Davide Scaramuzza. “Performance, Precision, and Payloads: Adaptive Optimal Control for Quadrotors Under Uncertainty”. In: *IEEE Robot. Autom. Lett.* 2022. DOI: [10.1109/LRA.2021.3131690](https://doi.org/10.1109/LRA.2021.3131690).
- [124] Christopher G. Harris and Mike Stephens. “A combined corner and edge detector”. In: *In Proc. of Fourth Alvey Vision Conference*. 1988.
- [125] W. K. Hastings. “Monte Carlo sampling methods using Markov chains and their applications”. In: *Biometrika* 57.1 (1970), pp. 97–109.
- [126] Gautier Hattenberger, Murat Bronz, and Michel Gorraz. “Using the Paparazzi UAV System for Scientific Research”. In: *International Micro Air Vehicle Conference and Competition*. 2014, pp. 247–252.
- [127] M. Hehn, R. Ritz, and R. D’Andrea. “Performance benchmarking of quadrotor systems using time-optimal control”. In: *Auton. Robots* (Mar. 2012). DOI: [10.1007/s10514-012-9282-3](https://doi.org/10.1007/s10514-012-9282-3).
- [128] Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. “Autonomous Visual Mapping and Exploration With a Micro Aerial Vehicle”. In: *J. Field Robotics* 31.4 (2014), pp. 654–675.

- 
- [129] Daniel Hernandez-Juarez, Alejandro Chacón, Antonio Espinosa, David Vázquez, Juan Carlos Moure, and Antonio M. López. “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.
- [130] Lukas Hewing, Juraj Kabzan, and Melanie Zeilinger. “Cautious model predictive control using Gaussian process regression”. In: *IEEE Trans. Control Sys. Tech.* (2019).
- [131] Juliane Hilf and Klaus Umbach. “The Commercial Use of Drones”. In: *Computer Law Review International* 16.3 (2015).
- [132] 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.
- [133] Gabriel Hoffmann, Haomiao Huang, Steven Waslander, and Claire Tomlin. “Quadrotor helicopter flight dynamics and control: Theory and experiment”. In: *AIAA guidance, navigation and control conference and exhibit*. 2007, p. 6461.
- [134] Gabriel M Hoffmann, Haomiao Huang, Steven L Waslander, and Claire J Tomlin. “Precision flight control for a multi-vehicle quadrotor helicopter testbed”. In: *Control engineering practice* 19.9 (2011), pp. 1023–1036.
- [135] Dirk Holz, Nicola Basilico, Francesco Amigoni, and Sven Behnke. “Evaluating the efficiency of frontier-based exploration strategies”. In: *Int. Symp. Robotics (ISR)* (2010).
- [136] Namdar Homayounfar, Wei-Chiu Ma, Justin Liang, Xinyu Wu, Jack Fan, and Raquel Urtasun. “Dagmapper: Learning to map by discovering lane topology”. In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2019, pp. 2911–2920.
- [137] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. “OctoMap: An efficient probabilistic 3D mapping framework based on octrees”. In: *Autonomous Robots* (2013).
- [138] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. “ACADO toolkit—An open-source framework for automatic control and dynamic optimization”. In: *Optimal Control Applications and Methods* (2011).
- [139] Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. “Searching for MobileNetV3”. In: *International Conference on Computer Vision, ICCV*. 2019, pp. 1314–1324.
- [140] Guoquan Huang. “Visual-inertial navigation: A concise review”. In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 9572–9582.
- [141] Haomiao Huang, Gabriel M Hoffmann, Steven L Waslander, and Claire J Tomlin. “Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering”. In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 3277–3282.

## Bibliography

---

- [142] Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. “Optimizing agent behavior over long time scales by transporting value”. In: *Nature Communications* 10.1 (Nov. 2019). DOI: [10.1038/s41467-019-13073-w](https://doi.org/10.1038/s41467-019-13073-w).
- [143] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. “Learning agile and dynamic motor skills for legged robots”. In: *Science Robotics* 4.26 (2019).
- [144] 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.
- [145] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. “Control of a quadrotor with reinforcement learning”. In: *IEEE Robot. Autom. Lett.* (2017).
- [146] Intel Corporation. *Intel Movidius Myriad X Vision Processing Unit*. <https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu/movidius-myriad-x.html>. Accessed: 2021-8-2.
- [147] *Intel RealSense T265 Series Product Family*. [https://www.intelrealsense.com/wp-content/uploads/2019/09/Intel\\_RealSense\\_Tracking\\_Camera\\_Datasheet\\_Rev004\\_release.pdf](https://www.intelrealsense.com/wp-content/uploads/2019/09/Intel_RealSense_Tracking_Camera_Datasheet_Rev004_release.pdf). 2019.
- [148] 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).
- [149] Jialin Ji, Zhepei Wang, Yingjian Wang, Chao Xu, and Fei Gao. “Mapless-planner: A robust and fast planning framework for aggressive autonomous flight without map fusion”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2021, pp. 6315–6321.
- [150] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. “Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?” In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.
- [151] Miguel Juliá, Arturo Gil, and Oscar Reinoso. “A comparison of path planning strategies for autonomous exploration and mapping of unknown environments”. In: *Autonomous Robots* 33.4 (2012), pp. 427–444.
- [152] Miguel Juliá, Oscar Reinoso, Arturo Gil, Mónica Ballesta, and Luis Payá. “A hybrid solution to the multi-robot integrated exploration problem”. In: *Engineering Applications of Artificial Intelligence* (2010).
- [153] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. “Unity: A general platform for intelligent agents”. In: *arXiv e-prints* (2018).
- [154] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zidek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andrew J Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian

- Bodenstein, David Silver, Oriol Vinyals, Andrew W Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. “Highly accurate protein structure prediction with AlphaFold”. In: *Nature* 596.7873 (2021), pp. 583–589. DOI: [10.1038/s41586-021-03819-2](https://doi.org/10.1038/s41586-021-03819-2).
- [155] Sunggoo Jung, Sungwook Cho, Dasol Lee, Hanseob Lee, and David Hyunchul Shim. “A direct visual servoing-based framework for the 2016 IROS Autonomous Drone Racing Challenge”. In: *J. Field Robot.* 35.1 (2018), pp. 146–166.
- [156] Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim. “Perception, Guidance, and Navigation for Indoor Autonomous Drone Racing Using Deep Learning”. In: *IEEE Robot. Autom. Lett.* 3.3 (2018). DOI: [10.1109/LRA.2018.2808368](https://doi.org/10.1109/LRA.2018.2808368).
- [157] Juraj Kabzan, Lukas Hewing, Alexander Liniger, and Melanie N Zeilinger. “Learning-based model predictive control for autonomous racing”. In: *IEEE Robot. Autom. Lett.* (2019).
- [158] Gregory Kahn, Adam Villafior, Bosen Ding, Pieter Abbeel, and Sergey Levine. “Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018.
- [159] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. “PLATO: Policy learning using adaptive trajectory optimization”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. IEEE, 2017, pp. 3342–3349.
- [160] Jean-Marie Kai, Guillaume Allibert, Minh-Duc Hua, and Tarek Hamel. “[Nonlinear feedback control of quadrotors exploiting first-order drag effects](#)”. In: *IFAC World Congress* 50.1 (2017), pp. 8189–8195.
- [161] R. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *J. Basic Eng.* 82 (1 1960), pp. 35–45.
- [162] M. Kamel, M. Burri, and R. Siegwart. “Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles”. In: *arXiv* (2016). URL: <http://arxiv.org/abs/1611.09240>.
- [163] Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. “Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight”. In: *2019 international conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 6008–6014.
- [164] Nitin R Kapania. “Trajectory Planning and Control for an Autonomous Race Vehicle”. PhD thesis. Stanford University, 2016.
- [165] 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.
- [166] Sertac Karaman and Emilio Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *Int. J. Robot. Research* 30.7 (2011), pp. 846–894.
- [167] Konstantinos Karydis and Vijay Kumar. “Energetics in robotic flight at small scales”. en. In: *Interface Focus* 7.1 (Feb. 2017), p. 20160088. ISSN: 2042-8898. DOI: [10.1098/rsfs.2016.0088](https://doi.org/10.1098/rsfs.2016.0088).

## Bibliography

---

- [168] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Champion-Level Drone Racing using Deep Reinforcement Learning”. In: *Nature* (2023).
- [169] Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. “A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight”. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022.
- [170] Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. “Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)* (2019), pp. 690–696. DOI: [10.1109/ICRA.2019.8793631](https://doi.org/10.1109/ICRA.2019.8793631).
- [171] Elia Kaufmann\*, Antonio Loquercio\*, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. “Deep Drone Racing: Learning Agile Flight in Dynamic Environments”. In: *Conf. on Robotics Learning (CoRL)*. 2018.
- [172] Elia Kaufmann\*, Antonio Loquercio\*, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Deep Drone Acrobatics”. In: *Robotics: Science and Systems (RSS)*. 2020.
- [173] Derya Kaya and Ali T Kutay. “Aerodynamic modeling and parameter estimation of a quadrotor helicopter”. In: *AIAA Atmospheric Flight Mechanics Conference*. 2014, p. 2558.
- [174] 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.
- [175] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. “Intel RealSense Stereoscopic Depth Cameras”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2017.
- [176] Waqas Khan and Meyer Nahon. “Toward an accurate physics-based UAV thruster model”. In: *IEEE/ASME Transactions on Mechatronics* 18.4 (2013), pp. 1269–1279.
- [177] Abbas Khosravi, Saeid Nahavandi, Doug Creighton, and Amir F Atiya. “Comprehensive review of neural network-based prediction intervals and new advances”. In: *IEEE Trans. Neural Netw.* 22.9 (2011). DOI: [10.1109/TNN.2011.2162110](https://doi.org/10.1109/TNN.2011.2162110). URL: <https://doi.org/10.1109/TNN.2011.2162110>.
- [178] Dong Ki Kim and Tsuhan Chen. “Deep neural network for real-time autonomous indoor navigation”. In: *arXiv:1511.04668* (2015).
- [179] Joohwan Kim, Josef Spjut, Morgan McGuire, Alexander Majercik, Ben Boudaoud, Rachel Albert, and David Luebke. “Esports Arms Race: Latency and Refresh Rate for Competitive Gaming Tasks”. In: *Journal of Vision* 19.10 (Sept. 2019), p. 218c. DOI: [10.1167/19.10.218c](https://doi.org/10.1167/19.10.218c).
- [180] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. 2015.



- 
- [181] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. “Reinforcement learning for UAV attitude control”. In: *ACM Transactions on Cyber-Physical Systems* (2019).
- [182] Nathan Koenig and Andrew Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. Vol. 3. 2004, pp. 2149–2154.
- [183] Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, and Oskar von Stryk. “Hector open source modules for autonomous mapping and navigation with rescue robots”. In: *Robot Soccer World Cup*. Springer. 2013, pp. 624–631.
- [184] 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.
- [185] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.
- [186] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. “Rma: Rapid motor adaptation for legged robots”. In: *arXiv preprint arXiv:2107.04034* (2021).
- [187] Vijay Kumar and Nathan Michael. “Opportunities and challenges with autonomous micro aerial vehicles”. In: *The International Journal of Robotics Research* 31.11 (2012), pp. 1279–1291.
- [188] *Laird Connectivity*. <https://www.lairdconnect.com/>. Accessed: 2021-7-20.
- [189] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. “Low-level control of a quadrotor with deep model-based reinforcement learning”. In: *IEEE Robot. Autom. Lett.* (2019).
- [190] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006. URL: <http://planning.cs.uiuc.edu>.
- [191] Junseok Lee, Xiangyu Wu, Seung Jae Lee, and Mark W Mueller. “Autonomous flight through cluttered outdoor environments using a memoryless planner”. In: *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2021, pp. 1131–1138.
- [192] Keuntaek Lee, Jason Gibson, and Evangelos A Theodorou. “Aggressive Perception-Aware Navigation using Deep Optical Flow Dynamics and PixelMPC”. In: *arXiv:2001.02307* (2020).
- [193] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. “DeepMPC: Learning deep latent features for model predictive control.” In: *Robotics: Science and Systems (RSS)*. 2015.
- [194] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. “Keyframe-Based Visual-Inertial SLAM using Nonlinear Optimization”. In: *Int. J. Robot. Research* (2015).
- [195] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-end training of deep visuomotor policies”. In: *The Journal of Machine Learning Research* (2016).

## Bibliography

---

- [196] Shuo Li, Erik van der Horst, Philipp Duernay, Christophe De Wagter, and Guido de Croon. “Visual Model-predictive Localization for Computationally Efficient Autonomous Racing of a 72-gram Drone”. In: *ArXiv abs/1905.10110* (2019).
- [197] Shuo Li, Michael MOI Ozo, Christophe De Wagter, and Guido CHE de Croon. “Autonomous drone race: A computationally efficient vision-based navigation and control strategy”. In: *Robotics and Autonomous Systems* 133 (2020).
- [198] Shuo Li, Ekin Ozturk, Christophe De Wagter, Guido de Croon, and Dario Izzo. “Aggressive Online Control of a Quadrotor via Deep Network Representations of Optimality Principles”. In: *arXiv:1912.07067* (2019).
- [199] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. “Fourier Neural Operator for Parametric Partial Differential Equations”. In: *International Conference on Learning Representations*. 2020.
- [200] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. “Microsoft COCO: Common Objects in Context”. In: *European Conference on Computer Vision (ECCV)*. 2014.
- [201] Cunjia Liu, Hao Lu, and Wen-Hua Chen. “An explicit MPC for quadrotor trajectory tracking”. In: *IEEE Chin. Control Conf. (CCC)*. 2015.
- [202] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. “Search-Based Motion Planning for Aggressive Flight in SE(3)”. In: *IEEE Robot. Autom. Lett.* (2018). DOI: [10.1109/LRA.2018.2795654](https://doi.org/10.1109/LRA.2018.2795654).
- [203] G. Loianno, C. Brunner, G. McGrath, and V. Kumar. “Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU”. In: *IEEE Robot. Autom. Lett.* (2017).
- [204] G. Loianno and D. Scaramuzza. “Special issue on future challenges and opportunities in vision-based drone navigation”. In: *J. Field Robot.* (2020). DOI: [10.1002/rob.21962](https://doi.org/10.1002/rob.21962).
- [205] W. Van Loock, G. Pipeleers, and J. Swevers. “Time-optimal quadrotor flight”. In: *IEEE Eur. Control Conf. (ECC)*. 2013. DOI: [10.23919/ECC.2013.6669253](https://doi.org/10.23919/ECC.2013.6669253).
- [206] Antonio Loquercio, Ana I. Maqueda, Carlos R. del-Blanco, and Davide Scaramuzza. “DroNet: Learning to Fly by Driving”. In: *IEEE Robot. Autom. Lett.* 3.2 (2018), pp. 1088–1095.
- [207] Antonio Loquercio, Alessandro Saviolo, and Davide Scaramuzza. “Autotune: Controller tuning for high-speed flight”. In: *IEEE Robot. Autom. Lett.* 7.2 (2022), pp. 4432–4439.
- [208] 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. Robot.* 36.1 (2019), pp. 1–14. DOI: [10.1109/TRO.2019.2942989](https://doi.org/10.1109/TRO.2019.2942989).
- [209] Antonio Loquercio\*, Elia Kaufmann\*, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. “Learning High-Speed Flight in the Wild”. In: *Science Robotics*. 2021.

- [210] Bruce D. Lucas and Takeo Kanade. “An Iterative Image Registration Technique with an Application to Stereo Vision”. In: *Int. Joint Conf. Artificial Intell. (IJCAI)*. 1981.
- [211] Martin Luessi. *radix*. en. <https://www.brainfpv.com/product/radix-fc/>. Accessed: 2021-7-20. Dec. 2017.
- [212] Dario Lunni, Angel Santamaria-Navarro, Roberto Rossi, Paolo Rocco, Luca Bascetta, and Juan Andrade-Cetto. “Nonlinear model predictive control for aerial manipulation”. In: *IEEE Int. Conf. Unmanned Aircraft Syst. (ICUAS)*. 2017.
- [213] Jinglin Luo, Longfei Zhu, and Guirong Yan. “Novel quadrotor forward-flight model based on wake interference”. In: *Aiaa Journal* 53.12 (2015), pp. 3522–3533.
- [214] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D’Andrea. “A simple learning strategy for high-speed quadcopter multi-flips”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2010, pp. 1642–1648.
- [215] S. Lynen, M. Achtelik, S. Weiss, M. Chli, and R. Siegwart. “A Robust and Modular Multi-Sensor Fusion Approach Applied to MAV Navigation”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2013.
- [216] Simon Lynen, Torsten Sattler, Michael Bosse, Joel Hesch, Marc Pollefeys, and Roland Siegwart. “Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization”. In: *Robotics: Science and Systems*. 2015.
- [217] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, and Ashish Kapoor. “AirSim Drone Racing Lab”. In: *PMLR post-proceedings of the NeurIPS 2019’s Competition Track* (2020).
- [218] R. Mahony, V. Kumar, and P. Corke. “Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor”. In: *IEEE Robot. Autom. Mag.* (2012). DOI: [10.1109/MRA.2012.2206474](https://doi.org/10.1109/MRA.2012.2206474).
- [219] Alexei A Makarenko, Stefan B Williams, Frederic Bourgault, and Hugh F Durrant-Whyte. “An experiment in integrated exploration”. In: *Intelligent Robots and Systems, IEEE/RSJ International Conference on*. 2002.
- [220] Philippe Martin and Erwan Salaün. “The true role of accelerometer feedback in quadrotor control”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2010.
- [221] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from [tensorflow.org](https://www.tensorflow.org/). 2015. URL: <https://www.tensorflow.org/>.

## Bibliography

---

- [222] Helmut Maurer. “On optimal control problems with bounded state variables and control appearing linearly”. In: *SIAM J. on Control and Optimization* 15.3 (1977), pp. 345–362.
- [223] Jasna Maver and Ruzena Bajcsy. “Occlusions as a guide for planning the next view”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* (1993).
- [224] Mohit Mehndiratta and Erdal Kayacan. “Gaussian Process-based Learning Control of Aerial Robots for Precise Visualization of Geological Outcrops”. In: *IEEE Eur. Control Conf. (ECC)*. 2020.
- [225] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. “PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 6235–6240.
- [226] Lorenz Meier, Petri Tanskanen, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. “PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision”. In: *Autonom. Rob.* 33.1 (Aug. 2012), pp. 21–39. DOI: [10.1007/s10514-012-9281-4](https://doi.org/10.1007/s10514-012-9281-4).
- [227] D. Mellinger and V. Kumar. “Minimum snap trajectory generation and control for quadrotors”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2011. DOI: [10.1109/ICRA.2011.5980409](https://doi.org/10.1109/ICRA.2011.5980409).
- [228] Daniel Mellinger, Nathan Michael, and Vijay Kumar. “Trajectory generation and control for precise aggressive maneuvers with quadrotors”. In: *Int. J. Robot. Research* 31.5 (Apr. 2012), pp. 664–674. ISSN: 0278-3649. DOI: [10.1177/0278364911434236](https://doi.org/10.1177/0278364911434236).
- [229] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar Von Stryk. “Comprehensive simulation of quadrotor uavs using ros and gazebo”. In: *International conference on simulation, modeling, and programming for autonomous robots*. Springer. 2012, pp. 400–411.
- [230] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, and Georg Ostrovski. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.
- [231] Nima Mohajerin, Melissa Mozifian, and Steven Waslander. “Deep learning a quadrotor dynamic model for multi-step prediction”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2454–2459.
- [232] Nima Mohajerin and Steven Waslander. “Multistep Prediction of Dynamic Systems With Recurrent Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2019).
- [233] Kartik Mohta, Michael Watterson, Yash Mulgaonkar, Sikang Liu, Chao Qu, Anurag Makineni, Kelsey Saulnier, Ke Sun, Alex Zhu, Jeffrey Delmerico, Konstantinos Karydis, Nikolay Atanasov, Giuseppe Loianno, Davide Scaramuzza, Kostas Daniilidis, Camillo Jose Taylor, and Vijay Kumar. “Fast, autonomous flight in GPS-denied and cluttered environments”. In: *J. Field Robot.* 35.1 (2018), pp. 101–120. DOI: [10.1002/rob.21774](https://doi.org/10.1002/rob.21774).

- [234] Artem Molchanov, Tao Chen, Wolfgang Hönig, James A. Preiss, Nora Ayanian, and Gaurav S. Sukhatme. “Sim-to-(Multi)-Real: Transfer of Low-Level Robust Control Policies to Multiple Quadrotors”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2019.
- [235] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, G. de Croon, S. Hwang, S. Jung, H. Shim, H. Kim, M. Park, T.-C. Au, and S. J. Kim. “Challenges and implemented technologies used in autonomous drone racing”. In: *J. Intell. Service Robot.* (2019). DOI: [10.1007/s11370-018-00271-6](https://doi.org/10.1007/s11370-018-00271-6).
- [236] H. Moon, Y. Sun, J. Baltes, and S. J. Kim. “The IROS 2016 Competitions [Competitions]”. In: *IEEE Robotics and Automation Magazine* 24.1 (2017), pp. 20–29.
- [237] Hyungpil Moon, Yu Sun, Jacky Baltes, and Si Jung Kim. “The IROS 2016 Competitions”. In: *IEEE Robot. Autom. Mag.* 24.1 (Mar. 2017), pp. 20–29. DOI: [10.1109/MRA.2016.2646090](https://doi.org/10.1109/MRA.2016.2646090). URL: <https://ieeexplore.ieee.org/document/7886372>.
- [238] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. “A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)”. en. In: *IEEE Trans. Biomed. Circuits Syst.* 12.1 (Feb. 2018), pp. 106–122. ISSN: 1932-4545, 1940-9990. DOI: [10.1109/TBCAS.2017.2759700](https://doi.org/10.1109/TBCAS.2017.2759700).
- [239] Benjamin Morrell, Marc Rigter, Gene Merewether, Robert Reid, Rohan Thakker, Theodore Tzanetos, Vinay Rajur, and Gregory Chamitoff. “Differential flatness transformations for aggressive quadrotor flight”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018.
- [240] Benjamin Morrell, Rohan Thakker, Gene Merewether, Robert G Reid, Marc Rigter, Theodore Tzanetos, and Gregory Chamitoff. “Comparison of Trajectory Optimization Algorithms for High-Speed Quadrotor Flight Near Obstacles”. In: *IEEE Robot. Autom. Lett.* 3.4 (2018).
- [241] Anastasios I. Mourikis and Stergios I. Roumeliotis. “A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. Apr. 2007, pp. 3565–3572.
- [242] M. W. Mueller, M. Hehn, and R. D’Andrea. “A computationally efficient algorithm for state-to-state quadcopter trajectory generation and feasibility verification”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2013. DOI: [10.1109/iros.2013.6696852](https://doi.org/10.1109/iros.2013.6696852).
- [243] Mark Wilfried Mueller, Markus Hehn, and Raffaello D’Andrea. “A Computationally Efficient Motion Primitive for Quadcopter Trajectory Generation”. In: *IEEE Trans. Robot.* 31.6 (2015), pp. 1294–1310.
- [244] Matthias Müller, Vincent Casser, Jean Lahoud, Neil Smith, and Bernard Ghanem. “Sim4cv: A photo-realistic simulator for computer vision applications”. In: *Int. J. Comput. Vis.* 126.9 (2018), pp. 902–919.
- [245] Matthias Müller, Vincent Casser, Neil Smith, Dominik L Michels, and Bernard Ghanem. “Teaching UAVs to Race Using UE4Sim”. In: *arXiv:1708.05884* (2017).

## Bibliography

---

- [246] Matthias Müller, Vincent Casser, Neil Smith, Dominik L. Michels, and Bernard Ghanem. “Teaching UAVs to Race: End-to-End Regression of Agile Controls in Simulation”. In: (2018).
- [247] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun. “Driving Policy Transfer via Modularity and Abstraction”. In: *Conference on Robot Learning*. 2018, pp. 1–15.
- [248] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. “ORB-SLAM: a Versatile and Accurate Monocular SLAM System”. In: *IEEE Trans. Robot.* 31.5 (2015), pp. 1147–1163.
- [249] Raúl Mur-Artal and Juan D. Tardós. “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”. In: *IEEE Trans. Robot.* 33.5 (2017).
- [250] Fang Nan, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. “Nonlinear MPC for Quadrotor Fault-Tolerant Control”. In: *IEEE Robot. Autom. Lett.* 2022.
- [251] K.S. Narendra and K. Parthasarathy. “Identification and control of dynamical systems using neural networks”. In: *IEEE Transactions on Neural Networks* (1990).
- [252] M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli. “Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2016. DOI: [10.1109/icra.2016.7487274](https://doi.org/10.1109/icra.2016.7487274).
- [253] Huan Nguyen, Mina Kamel, Kostas Alexis, and Roland Siegwart. “Model Predictive Control for Micro Aerial Vehicles: A Survey”. In: *arXiv e-prints* (Nov. 2020). arXiv: [2011.11104](https://arxiv.org/abs/2011.11104) [[cs.R0](#)].
- [254] Barza Nisar, Philipp Foehn, Davide Falanga, and Davide Scaramuzza. “VIMO: Simultaneous Visual Inertial Model-Based Odometry and Force Estimation”. In: *IEEE Robot. Autom. Lett.* 4.3 (July 2019), pp. 2785–2792. DOI: [10.1109/LRA.2019.2918689](https://doi.org/10.1109/LRA.2019.2918689).
- [255] David A Nix and Andreas S Weigend. “Estimating the mean and variance of the target probability distribution”. In: *IEEE Int. Conf. Neural Netw.* 1994.
- [256] Helen Oleynikova, Christian Lanegger, Zachary Taylor, Michael Pantic, Alexander Millane, Roland Siegwart, and Juan Nieto. “An open-source system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments”. In: *J. Field Robot.* 37.4 (June 2020), pp. 642–666. DOI: [10.1002/rob.21950](https://doi.org/10.1002/rob.21950).
- [257] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan I. Nieto. “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.
- [258] Sammy Omari, Minh-Duc Hua, Guillaume Ducard, and Tarek Hamel. “Nonlinear control of vtol uavs incorporating flapping dynamics”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2013.
- [259] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).

- [260] Matko Orsag and Stjepan Bogdan. “Influence of forward and descent flight on quadrotor dynamics”. In: *Recent Advances in Aircraft Technology* (2012), pp. 141–156.
- [261] T. Ozaslan, G. Loianno, J. Keller, C. J. Taylor, V. Kumar, J. M. Wozencraft, and T. Hood. “Autonomous Navigation and Mapping for Inspection of Penstocks and Tunnels With MAVs”. In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1740–1747.
- [262] D. Palossi, A. Loquercio, F. Conti, F. Conti, E. Flamand, E. Flamand, D. Scaramuzza, L. Benini, and L. Benini. “A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones”. In: *IEEE Internet of Things Journal* (2019), pp. 1–1. ISSN: 2327-4662. DOI: [10.1109/JIOT.2019.2917066](https://doi.org/10.1109/JIOT.2019.2917066).
- [263] Daniele Palossi, Francesco Conti, and Luca Benini. “An Open Source and Open Hardware Deep Learning-Powered Visual Navigation Engine for Autonomous Nano-UAVs”. In: *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. May 2019, pp. 604–611. DOI: [10.1109/DCOSS.2019.00111](https://doi.org/10.1109/DCOSS.2019.00111).
- [264] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. “Agile Autonomous Driving Using End-to-End Deep Imitation Learning”. In: *Robotics: Science and Systems (RSS)*. 2018.
- [265] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P Schoellig. “Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 7512–7519.
- [266] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [267] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. “Learning Agile Robotic Locomotion Skills by Imitating Animals”. In: *Robotics: Science and Systems (RSS)*. 2020. DOI: [10.15607/RSS.2020.XVI.064](https://doi.org/10.15607/RSS.2020.XVI.064).
- [268] Xue Bin Peng and Michiel van de Panne. “Learning locomotion skills using deeprl: Does the choice of action space matter?” In: *Proceedings of the ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* 2017.
- [269] Karime Pereida, Lukas Brunke, and Angela P Schoellig. “Robust adaptive model predictive control for guaranteed fast and accurate stabilization in the presence of model errors”. In: *International Journal of Robust and Nonlinear Control* 31.18 (2021), pp. 8750–8784.

## Bibliography

---

- [270] Christian Pfeiffer and Davide Scaramuzza. “Human-piloted drone racing: Visual processing and control”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3467–3474.
- [271] Chen-Huan Pi, Kai-Chun Hu, Stone Cheng, and I-Chen Wu. “Low-level autonomous control and tracking of quadrotor using reinforcement learning”. In: *Control Engineering Practice* (2020).
- [272] Chen-Huan Pi, Wei-Yuan Ye, and Stone Cheng. “Robust quadrotor control through reinforcement learning with disturbance compensation”. In: *Applied Sciences* (2021).
- [273] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. “Asymmetric Actor Critic for Image-Based Robot Learning”. In: *Robotics: Science and Systems (RSS)*. 2018. DOI: [10.15607/RSS.2018.XIV.008](https://doi.org/10.15607/RSS.2018.XIV.008).
- [274] Gavin D Portwood, Peetak P Mitra, Mateus Dias Ribeiro, Tan Minh Nguyen, Balasubramanya T Nadiga, Juan A Saenz, Michael Chertkov, Animesh Garg, Anima Anandkumar, and Andreas Dengel. “Turbulence forecasting via Neural ODE”. In: *2nd Workshop on Machine Learning and the Physical Sciences (NeurIPS 2019)* (2019).
- [275] Caitlin Powers, Daniel Mellinger, Aleksandr Kushleyev, Bruce Kothmann, and Vijay Kumar. “Influence of aerodynamics and proximity effects in quadrotor flight”. In: *Experimental robotics*. Springer. 2013, pp. 289–302.
- [276] Raymond W Prouty. *Helicopter performance, stability, and control*. 1995.
- [277] Ali Punjani and Pieter Abbeel. “Deep learning helicopter dynamics models”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 3223–3230.
- [278] T. Qin, P. Li, and S. Shen. “VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator”. In: *IEEE Trans. Robot.* July 2018.
- [279] M Quigley, J Faust, T Foote, and J Leibs. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Dec. 2009.
- [280] René Ranftl and Vladlen Koltun. “Deep fundamental matrix estimation”. In: *European Conference on Computer Vision (ECCV)*. 2018.
- [281] Kanishka Rao, Chris Harris, Alex Irpan, Sergey Levine, Julian Ibarz, and Mohi Khansari. “RL-CycleGAN: Reinforcement Learning Aware Simulation-to-Real”. In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2020, pp. 11154–11163. DOI: [10.1109/CVPR42600.2020.01117](https://doi.org/10.1109/CVPR42600.2020.01117).
- [282] 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.
- [283] Charles Richter and Nicholas Roy. “Safe Visual Navigation via Deep Learning and Novelty Detection”. In: *Robotics: Science and Systems*. 2017.
- [284] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. “Playing for Data: Ground Truth from Computer Games”. In: *European Conference on Computer Vision (ECCV)*. 2016.



- 
- [285] 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.
- [286] Angel Romero, Sihao Sun, Philipp Foehn, and Davide Scaramuzza. “Model Predictive Contouring Control for Near-Time-Optimal Quadrotor Flight”. In: *IEEE Trans. Robot.* (2021). arXiv: [2108.13205](https://arxiv.org/abs/2108.13205) [cs.R0].
- [287] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.
- [288] Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschafer, and Davide Scaramuzza. “Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High Speed Scenarios”. In: *IEEE Robot. Autom. Lett.* 3.2 (2018), pp. 994–1001. DOI: [10.1109/LRA.2018.2793357](https://doi.org/10.1109/LRA.2018.2793357).
- [289] 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.
- [290] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J. Andrew Bagnell, and Martial Hebert. “Learning monocular reactive UAV control in cluttered natural environments”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2013, pp. 1765–1772.
- [291] Pengkai Ru and Kamesh Subbarao. “Nonlinear model predictive control for unmanned aerial vehicles”. In: *Aerospace* (2017).
- [292] Christian Rupprecht, Iro Laina, Robert S. DiPietro, and Maximilian Baust. “Learning in an Uncertain World: Representing Ambiguity Through Multiple Hypotheses”. In: *International Conference on Computer Vision, ICCV*. 2017, pp. 3611–3620.
- [293] Carl Russell, Jaewoo Jung, Gina Willink, and Brett Glasner. “Wind tunnel and hover performance test results for multicopter UAS vehicles”. In: *AHS 72nd annual forum*. 2016, pp. 16–19.
- [294] Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. “Sim-to-real robot learning from pixels with progressive nets”. In: *Conference on Robot Learning (CoRL)*. 2017.
- [295] Markus Ryll, John Ware, John Carter, and Nick Roy. “Efficient trajectory planning for high speed flight in unknown environments”. In: *2019 International conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 732–738.
- [296] Gilhyun Ryou, Ezra Tal, and Sertac Karaman. “Multi-fidelity black-box optimization for time-optimal quadrotor maneuvers”. In: *Robotics: Science and Systems (RSS)*. 2020.
- [297] Inkyu Sa, Mina Kamel, Michael Burri, Michael Blosch, Raghav Khanna, Marija Popovic, Juan Nieto, and Roland Siegwart. “Build Your Own Visual-Inertial Drone: A Cost-Effective and Open-Source Autonomous Drone”. In: *IEEE Robot. Autom. Mag.* 25.1 (Mar. 2018), pp. 89–103. ISSN: 1070-9932.

## Bibliography

---

- [298] 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, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma. 2017.
- [299] Fereshteh Sadeghi, Alexander Toshev, Eric Jang, and Sergey Levine. “Sim2Real Viewpoint Invariant Visual Servoing by Recurrent Control”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018. DOI: [10.1109/cvpr.2018.00493](https://doi.org/10.1109/cvpr.2018.00493).
- [300] Parrot Drone SAS. *Parrot ANAFI Ai*. <https://www.parrot.com/en/drones/anafi-ai>. Accessed: 2021-7-20.
- [301] Matteo Saveriano, Yuchao Yin, Pietro Falco, and Dongheui Lee. “Data-efficient control policy search using residual dynamics learning”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017.
- [302] Thomas Sayre-McCord, Winter Guerra, Amado Antonini, Jasper Arneberg, Austin Brown, Guilherme Cavalheiro, Yajun Fang, Alex Gorodetsky, Dave McCoy, Sebastian Quilter, Fabian Riether, Ezra Tal, Yunus Terzioglu, Luca Carlone, and Sertac Karaman. “Visual-Inertial Navigation Algorithm Development Using Photorealistic Camera Simulation in the Loop”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018. DOI: [10.1109/ICRA.2018.8460692](https://doi.org/10.1109/ICRA.2018.8460692).
- [303] Davide Scaramuzza, Michael Achtelik, Lefteris Doitsidis, Friedrich Fraundorfer, Elias B. Kosmatopoulos, Agostino Martinelli, Markus W. Achtelik, Margarita Chli, Savvas A. Chatzichristofis, Laurent Kneip, Daniel Gurdan, Lionel Heng, Gim Hee Lee, Simon Lynen, Marc Pollefeys, Alessandro Renzaglia, Roland Siegwart, Jan Carsten Stumpf, Petri Tanskanen, Chiara Troiani, Stephan Weiss, and Lorenz Meier. “Vision-Controlled Micro Flying Robots: From System Design to Autonomous Navigation and Mapping in GPS-Denied Environments”. In: *IEEE Robot. Autom. Mag.* 21.3 (2014), pp. 26–40.
- [304] Davide Scaramuzza and Zichao Zhang. “Visual-inertial odometry of aerial robots”. In: *Encyclopedia of Robotics* (2019).
- [305] Fabrizio Schiano, Javier Alonso-Mora, Konrad Rudin, Paul Beardsley, Roland Y Siegwart, and Bruno Sicilianok. “Towards estimation and correction of wind effects on a quadrotor UAV”. In: *IMAV 2014: International Micro Air Vehicle Conference and Competition 2014*. 2014, pp. 134–141.
- [306] T. Schneider, M. T. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart. “maplab: An Open Framework for Research in Visual-inertial Mapping and Localization”. In: *IEEE Robot. Autom. Lett.* 3.3 (2018).
- [307] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. “Mastering Atari, Go, chess and shogi by planning with a learned model”. In: *Nature* 588.7839 (2020), pp. 604–609.
- [308] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal policy optimization algorithms”. In: *arXiv e-prints* (2017).

- 
- [309] S. Shah, D. Dey, C. Lovett, and A. Kapoor. “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles”. In: *Field and Service Robot*. 2017, pp. 621–635.
- [310] Shaojie Shen, Nathan Michael, and Vijay Kumar. “Stochastic differential equation-based exploration algorithm for autonomous indoor 3d exploration with a micro-aerial vehicle”. In: *The International Journal of Robotics Research* 31.12 (2012), pp. 1431–1444.
- [311] Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. “Vision-Based State Estimation and Trajectory Control Towards High-Speed Flight with a Quadrotor”. In: *Robotics: Science and Systems (RSS)*. 2013.
- [312] Guanya Shi, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. “Neural lander: Stable drone landing control using learned dynamics”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2019. DOI: [10.1109/icra.2019.8794351](https://doi.org/10.1109/icra.2019.8794351).
- [313] Malcolm D. Shuster. “Survey of attitude representations”. In: *Journal of the Astronautical Sciences* 41.4 (Oct. 1993), pp. 439–517.
- [314] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.
- [315] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: [10.1126/science.aar6404](https://doi.org/10.1126/science.aar6404).
- [316] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, and Adrian Bolton. “Mastering the game of go without human knowledge”. In: *nature* 550.7676 (2017), pp. 354–359.
- [317] Skydio. *Skydio*. July 26, 2021.
- [318] Ewoud JJ Smeur, Qiping Chu, and Guido CHE de Croon. “Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles”. In: *Journal of Guidance, Control, and Dynamics* 39.3 (2016), pp. 450–461.
- [319] Laura Smith, J Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. “Legged Robots that Keep on Learning: Fine-Tuning Locomotion Policies in the Real World”. In: *arXiv preprint arXiv:2110.05457* (2021).
- [320] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. “Flightmare: A Flexible Quadrotor Simulator”. In: *Conference on Robot Learning*. 2020.
- [321] Yunlong Song and Davide Scaramuzza. “Learning High-Level Policies for Model Predictive Control”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2020.

## Bibliography

---

- [322] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. “Autonomous Drone Racing with Deep Reinforcement Learning”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2021.
- [323] Igor Spasojevic, Varun Murali, and Sertac Karaman. “Joint Feature Selection and Time Optimal Path Parametrization for High Speed Vision-Aided Navigation”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2020.
- [324] Riccardo Spica, Eric Cristofalo, Zijian Wang, Eduardo Montijano, and Mac Schwager. “A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing”. In: *IEEE Transactions on Robotics* 36.5 (Oct. 2020), pp. 1389–1403. DOI: [10.1109/tro.2020.2994881](https://doi.org/10.1109/tro.2020.2994881).
- [325] Nathan A Spielberg, Matthew Brown, and J Christian Gerdes. “Neural Network Model Predictive Motion Control Applied to Automated Driving With Unknown Friction”. In: *IEEE Transactions on Control Systems Technology* (2021).
- [326] Nathan A Spielberg, Matthew Brown, Nitin R Kapania, John C Kegelmann, and J Christian Gerdes. “Neural network vehicle models for high-performance automated driving”. In: *Science robotics* (2019).
- [327] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. “Navion: A 2-mW Fully Integrated Real-Time Visual-Inertial Odometry Accelerator for Autonomous Navigation of Nano Drones”. In: *IEEE J. Solid-State Circuits* 54.4 (Apr. 2019), pp. 1106–1119. ISSN: 0018-9200, 1558-173X. DOI: [10.1109/JSSC.2018.2886342](https://doi.org/10.1109/JSSC.2018.2886342).
- [328] Sihao Sun, Giovanni Cioffi, Coen De Visser, and Davide Scaramuzza. “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.
- [329] Sihao Sun, Angel Romero, Philipp Foehn, Elia Kaufmann, and Davide Scaramuzza. “A Comparative Study of Nonlinear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight”. In: *IEEE Trans. Robot.* (2022).
- [330] Sihao Sun, Leon Sijbers, Xuerui Wang, and Coen de Visser. “High-speed flight of quadrotor despite loss of single rotor”. In: *IEEE Robot. Autom. Lett.* 3.4 (2018), pp. 3201–3207.
- [331] Sihao Sun, Coen C de Visser, and Qiping Chu. “Quadrotor gray-box model identification from high-speed flight data”. In: *Journal of Aircraft* 56.2 (2019), pp. 645–661.
- [332] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [333] “Swiss Drone Industry Report 2021”. In: *Drone Industry Association Switzerland* (Nov. 1, 2021). URL: <https://droneindustry.ch/swiss-drone-industry-report-2021/> (visited on 04/16/2022).
- [334] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 2010. ISBN: 9781848829343.
- [335] 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.

- 
- [336] Ezra Tal and Sertac Karaman. “Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness”. In: *IEEE Transactions on Control Systems Technology* 29.3 (2020), pp. 1203–1218.
- [337] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. “Sim-to-real: Learning agile locomotion for quadruped robots”. In: *Robotics: Science and Systems (RSS)* (2018).
- [338] Yi-Rui Tang and Yangmin Li. “Dynamic modeling for high-performance controller design of a UAV quadrotor”. In: *2015 IEEE International Conference on Information and Automation*. IEEE, 2015, pp. 3112–3117.
- [339] The Apache Software Foundation. *NuttX*. <https://nuttx.apache.org/>. Accessed: 2021-7-20.
- [340] The Betaflight Open Source Flight Controller Firmware Project. *Betaflight*. <https://github.com/betaflight/betaflight>. Accessed: 2021-7-20.
- [341] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*. 2017.
- [342] 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.
- [343] T. Tomic, P. Lutz, K. Schmid, A. Mathers, and S. Haddadin. “Simultaneous contact and aerodynamic force estimation (s-CAFE) for aerial robots”. In: *Int. J. Robot. Research* (2020). DOI: [10.1177/0278364920904788](https://doi.org/10.1177/0278364920904788).
- [344] Erkki Tomppo. “Models and methods for analysing spatial patterns of trees”. PhD dissertation. Finnish Forest Research Institute, 1986.
- [345] 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](https://doi.org/10.1109/IROS40897.2019.8968021).
- [346] Guillem Torrente\*, Elia Kaufmann\*, Philipp Foehn, and Davide Scaramuzza. “Data-driven mpc for quadrotors”. In: *IEEE Robot. Autom. Lett.* 6.2 (2021), pp. 3769–3776.
- [347] Benjamin Tovar, Lourdes Munoz-Gomez, Rafael Murrieta-Cid, Moises Alencastre-Miranda, Raul Monroy, and Seth Hutchinson. “Planning exploration strategies for simultaneous localization and mapping”. In: *J. Robot. and Auton. Syst.* (2006).
- [348] Unity. *Unity Asset Store*. <https://assetstore.unity.com/>. Accessed: 2021-08-02.
- [349] Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. “Direct visual-inertial odometry with stereo cameras”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2016.
- [350] Patricia Ventura Diaz and Steven Yoon. “High-Fidelity Computational Aerodynamics of Multi-Rotor Unmanned Aerial Vehicles”. In: *AIAA Aerospace Sciences Meeting*. 2018.

## Bibliography

---

- [351] Jon Verbeke and Joris De Schutter. “Experimental maneuverability and agility quantification for rotary unmanned aerial vehicle”. In: *International Journal of Micro Air Vehicles* (2018). DOI: [10.1177/1756829317736204](https://doi.org/10.1177/1756829317736204).
- [352] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Niels van Duijkeren, Andrea Zanelli, Rien Quirynen, and Moritz Diehl. “Towards a modular software package for embedded optimization”. In: *IFAC-PapersOnLine* (2018).
- [353] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, and Petko Georgiev. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (2019), pp. 350–354.
- [354] Antonio Vitale<sup>1</sup>, Alpha Renner, Celine Nauer, Davide Scaramuzza, and Yulia Sandamirskaya. “Event-driven Vision and Control for UAVs on a Neuromorphic Chip”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2021.
- [355] Christophe De Wagter, Federico Paredes-Vallé, Nilay Sheth, and Guido de Croon. “The sensing, state-estimation, and control behind the winning entry to the 2019 Artificial Intelligence Robotic Racing Competition”. In: *Field Robotics* 2.1 (Mar. 2022), pp. 1263–1290. DOI: [10.55417/fr.2022042](https://doi.org/10.55417/fr.2022042).
- [356] Douglas Brent West. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, NJ, 2001.
- [357] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. The MIT Press, Cambridge, MA, 2006.
- [358] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. “Aggressive Driving with Model Predictive Path Integral Control”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. Stockholm, Sweden, May 2016, pp. 1433–1440.
- [359] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. “Information-Theoretic Model Predictive Control: Theory and Applications to Autonomous Driving”. In: *IEEE Trans. Robotics* 34.6 (2018), pp. 1603–1622.
- [360] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. “Information theoretic MPC for model-based reinforcement learning”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2017.
- [361] Dong-Ok Won, Klaus-Robert Müller, and Seong-Whan Lee. “An adaptive deep reinforcement learning framework enables curling robots with human-like performance in real-world conditions”. In: *Science Robotics* 5.46 (2020).
- [362] Markus Wulfmeier, Ingmar Posner, and Pieter Abbeel. “Mutual alignment transfer learning”. In: *Conference on Robot Learning (CoRL)*. 2017.
- [363] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. “Outracing champion Gran Turismo drivers with deep reinforcement learning”. In: *Nature* 602.7896 (2022), pp. 223–228.
- [364] Brian Yamauchi. “A frontier-based approach for autonomous exploration”. In: *IEEE Int. Symp. Comp. Intell. Robot. and Autom.* 1997. DOI: [10.1109/CIRA.1997.613851](https://doi.org/10.1109/CIRA.1997.613851).

- 
- [365] Brian Yamauchi. “Frontier-based exploration using multiple robots”. In: *ACM Int. Conf. Aut. Agents*. 1998. DOI: [10.1145/280765.280773](https://doi.org/10.1145/280765.280773).
- [366] Guang-Zhong Yang, Jim Bellingham, Pierre E Dupont, Peer Fischer, Luciano Floridi, Robert Full, Neil Jacobstein, Vijay Kumar, Marcia McNutt, and Robert Merrifield. “The grand challenges of Science Robotics”. In: *Science Robotics* 3.14 (2018), eaar7650.
- [367] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. “End-to-end interpretable neural motion planner”. In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2019, pp. 8660–8669.
- [368] Wenyuan Zeng, Shenlong Wang, Renjie Liao, Yun Chen, Bin Yang, and Raquel Urtasun. “Dsdnet: Deep structured self-driving network”. In: *Eur. Conf. Comput. Vis. (ECCV)*. 2020, pp. 156–172.
- [369] Daniel Zhang and Daniel D. Doyle. “Gate Detection Using Deep Learning”. In: *2020 IEEE Aerospace Conference*. 2020, pp. 1–11. DOI: [10.1109/AERO47225.2020.9172619](https://doi.org/10.1109/AERO47225.2020.9172619).
- [370] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. “Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2016.
- [371] Zichao Zhang and Davide Scaramuzza. “Perception-aware Receding Horizon Navigation for MAVs”. In: *IEEE Int. Conf. Robot. Autom. (ICRA)*. 2018, pp. 2534–2541.
- [372] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen. “Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight”. In: *IEEE Robot. Autom. Lett.* (2019). DOI: [10.1109/LRA.2019.2927938](https://doi.org/10.1109/LRA.2019.2927938).
- [373] Boyu Zhou, Fei Gao, Jie Pan, and Shaojie Shen. “Robust real-time uav replanning using guided gradient-based optimization and topological paths”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 1208–1214.
- [374] Boyu Zhou, Jie Pan, Fei Gao, and Shaojie Shen. “RAPTOR: Robust and Perception-aware Trajectory Replanning for Quadrotor Fast Flight”. In: *IEEE Trans. Robot.* (2021).
- [375] Brady Zhou, Philipp Krähenbühl, and Vladlen Koltun. “Does computer vision matter for action?” In: *Sci. Robotics* 4.30 (2019).
- [376] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. “On the continuity of rotation representations in neural networks”. In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*. 2019.





# Elia Kaufmann

*MSc ETH Zurich  
Robotics, Systems and Control*

*Luzernerstrasse 108  
6333 Hünenberg See  
Switzerland*

---

## Personal Details

Name **Elia Kaufmann**  
Date of Birth **03.06.1992**

---

## Education

- 11.2017– **Doctoral Program**, *University of Zurich, Department of Informatics*  
07.2022 PhD thesis in Machine Learning for Robotics: Learning Agile Flight  
Advisor: Prof. Davide Scaramuzza
- 05.2017– **Research Assistant**, *Robotics and Perception Group*,  
11.2017 Department of Neuroinformatics, Department of Informatics
- 09.2014– **MSc Robotics, Systems and Control**, *ETH Zurich*  
05.2017 Focus: Dynamic Programming and Optimal Control, Machine Learning, Recursive Estimation, System Identification, Model Predictive Control, Robot Dynamics  
Advisor: Prof. Raffaello D'Andrea  
**Master Thesis:**  
Rapid Exploration with Multi-Rotors: A Frontier Selection Method for High Speed Flight  
Advisor: Prof. Davide Scaramuzza  
**Semester Thesis:**  
Nonlinear Infinite-Horizon Model Predictive Control Using Multi-Interval Polynomial Trajectories  
Advisor: Prof. Raffaello D'Andrea, Dr. Michael Mühlebach
- 09.2011– **BSc Mechanical Engineering**, *ETH Zurich*  
05.2014 Focus: Control Systems, Robotics and Mechatronics  
**Bachelor Thesis:**  
Using Magnetometers and Barometers During Indoor Flight  
Advisor: Prof. Raffaello D'Andrea, Prof. Mark Müller